nSMPRI_EL1, bit [54]

When FEAT_SME is implemented:

Trap MRS reads of SMPRI_EL1 at EL1 using AArch64 to EL2.

nSMPRI_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of SMPRI_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of SMPRI_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nGCS_EL1, bit [53]

When FEAT_GCS is implemented:

Trap MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- GCSCR_EL1.
- GCSPR_EL1.

nGCS_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of the specified System registers are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nGCS_EL0, bit [52]

When FEAT_GCS is implemented:

Trap MRS reads at EL1 and EL0 using AArch64 of any of the following AArch64 System registers to EL2:

- GCSCRE0_EL1, at EL1 only.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- GCSPR_EL0.

nGCS_EL0	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 and EL0 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of the specified System registers are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [51]

Reserved, RES0.

nACCDATA_EL1, bit [50]

When FEAT_LS64_ACCDATA is implemented:

Trap MRS reads of ACCDATA_EL1 at EL1 using AArch64 to EL2.

nACCDATA_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ACCDATA_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.
0b1	MRS reads of ACCDATA_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ERXADDR_EL1, bit [49]

When FEAT_RAS is implemented:

Trap MRS reads of ERXADDR_EL1 at EL1 using AArch64 to EL2.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ERXADDR_EL1	Meaning
0b0	MRS reads of ERXADDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ERXADDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ERXPFGCDN_EL1, bit [48]

When FEAT_RASv1p1 is implemented:

Trap MRS reads of ERXPFGCDN_EL1 at EL1 using AArch64 to EL2.

ERXPFGCDN_EL1	Meaning
0b0	MRS reads of ERXPFGCDN_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ERXPFGCDN_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ERXPFGCTL_EL1, bit [47]

When FEAT_RASv1p1 is implemented:

Trap MRS reads of ERXPFGCTL_EL1 at EL1 using AArch64 to EL2.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ERXPFGCTL_EL1	Meaning
0b0	MRS reads of ERXPFGCTL_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ERXPFGCTL_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ERXPFGF_EL1, bit [46]

When FEAT_RASv1p1 is implemented:

Trap MRS reads of ERXPFGF_EL1 at EL1 using AArch64 to EL2.

ERXPFGF_EL1	Meaning
0b0	MRS reads of ERXPFGF_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ERXPFGF_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ERXMISCn_EL1, bit [45]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_RAS is implemented:

Trap MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- ERXMISC0_EL1.
- ERXMISC1_EL1.
- ERXMISC2_EL1.
- ERXMISC3_EL1.

ERXMISCn_EL1	Meaning
0b0	MRS reads of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ERXSTATUS_EL1, bit [44]

When FEAT_RAS is implemented:

Trap MRS reads of ERXSTATUS_EL1 at EL1 using AArch64 to EL2.

ERXSTATUS_EL1	Meaning
0b0	MRS reads of ERXSTATUS_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ERXSTATUS_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- When the highest implemented Exception level is EL2, this field resets to `'0'`.
- Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ERXCTLR_EL1, bit [43]

When FEAT_RAS is implemented:

Trap MRS reads of ERXCTLR_EL1 at EL1 using AArch64 to EL2.

ERXCTLR_EL1	Meaning
`0b0`	MRS reads of ERXCTLR_EL1 are not trapped by this mechanism.
`0b1`	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ERXCTLR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the read generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ERXFR_EL1, bit [42]

When FEAT_RAS is implemented:

Trap MRS reads of ERXFR_EL1 at EL1 using AArch64 to EL2.

ERXFR_EL1	Meaning
`0b0`	MRS reads of ERXFR_EL1 are not trapped by this mechanism.
`0b1`	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ERXFR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the read generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ERRSELR_EL1, bit [41]

When FEAT_RAS is implemented:

Trap MRS reads of ERRSELR_EL1 at EL1 using AArch64 to EL2.

ERRSELR_EL1	Meaning
`0b0`	MRS reads of ERRSELR_EL1 are not trapped by this mechanism.
`0b1`	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ERRSELR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the read generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ERRIDR_EL1, bit [40]

When FEAT_RAS is implemented:

Trap MRS reads of ERRIDR_EL1 at EL1 using AArch64 to EL2.

ERRIDR_EL1	Meaning
`0b0`	MRS reads of ERRIDR_EL1 are not trapped by this mechanism.
`0b1`	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ERRIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the read generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
- ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ICC_IGRPENn_EL1, bit [39]

When GICv3 is implemented:

Trap `MRS` reads of ICC_IGRPEN<n>_EL1 at EL1 using AArch64 to EL2.

ICC_IGRPENn_EL1	Meaning
`0b0`	`MRS` reads of ICC_IGRPEN<n>_EL1 are not trapped by this mechanism.
`0b1`	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then `MRS` reads of ICC_IGRPEN<n>_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VBAR_EL1, bit [38]

Trap `MRS` reads of VBAR_EL1 at EL1 using AArch64 to EL2.

VBAR_EL1	Meaning
`0b0`	`MRS` reads of VBAR_EL1 are not trapped by this mechanism.
`0b1`	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then `MRS` reads of VBAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TTBR1_EL1, bit [37]

Trap MRS or MRRS reads of TTBR1_EL1 at EL1 using AArch64 to EL2.

TTBR1_EL1	Meaning
0b0	MRS and MRRS reads of TTBR1_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the read generates a higher priority exception: • MRS reads of TTBR1_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • MRRS reads of TTBR1_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x14.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TTBR0_EL1, bit [36]

Trap MRS or MRRS reads of TTBR0_EL1 at EL1 using AArch64 to EL2.

TTBR0_EL1	Meaning
0b0	MRS and MRRS reads of TTBR0_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the read generates a higher priority exception: • MRS reads of TTBR0_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • MRRS reads of TTBR0_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x14.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TPIDR_EL0, bit [35]

Trap MRS reads of TPIDR_EL0 at EL1 and EL0 using AArch64 and MRC reads of TPIDRURW at EL0 using AArch32 when EL1 is using AArch64 to EL2.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TPIDR_EL0	Meaning
0b0	MRS reads of TPIDR_EL0 at EL1 and EL0 using AArch64 and MRC reads of TPIDRURW at EL0 using AArch32 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the read generates a higher priority exception: • MRS reads of TPIDR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • MRC reads of TPIDRURW at EL0 using AArch32 when EL1 is using AArch64 are trapped to EL2 and reported with EC syndrome value 0x03.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TPIDRRO_EL0, bit [34]

Trap MRS reads of TPIDRRO_EL0 at EL1 and EL0 using AArch64 and MRC reads of TPIDRURO at EL0 using AArch32 when EL1 is using AArch64 to EL2.

TPIDRRO_EL0	Meaning
0b0	MRS reads of TPIDRRO_EL0 at EL1 and EL0 using AArch64 and MRC reads of TPIDRURO at EL0 using AArch32 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the read generates a higher priority exception: • MRS reads of TPIDRRO_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • MRC reads of TPIDRURO at EL0 using AArch32 when EL1 is using AArch64 are trapped to EL2 and reported with EC syndrome value 0x03.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TPIDR_EL1, bit [33]

Trap MRS reads of TPIDR_EL1 at EL1 using AArch64 to EL2.

TPIDR_EL1	Meaning
0b0	MRS reads of TPIDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TPIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TCR_EL1, bit [32]

Trap `MRS` reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- TCR_EL1.
- TCR2_EL1, if FEAT_TCR2 is implemented.

TCR_EL1	Meaning
`0b0`	`MRS` reads of the specified System registers are not trapped by this mechanism.
`0b1`	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then `MRS` reads at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value `0x18`, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

SCXTNUM_EL0, bit [31]

When FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented:

Trap `MRS` reads of SCXTNUM_EL0 at EL1 and EL0 using AArch64 to EL2.

SCXTNUM_EL0	Meaning
`0b0`	`MRS` reads of SCXTNUM_EL0 are not trapped by this mechanism.
`0b1`	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then `MRS` reads of SCXTNUM_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SCXTNUM_EL1, bit [30]

When FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented:

Trap `MRS` reads of SCXTNUM_EL1 at EL1 using AArch64 to EL2.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SCXTNUM_EL1	Meaning
0b0	MRS reads of SCXTNUM_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of SCXTNUM_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SCTLR_EL1, bit [29]

Trap MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- SCTLR_EL1.
- SCTLR2_EL1, if FEAT_SCTLR2 is implemented.

SCTLR_EL1	Meaning
0b0	MRS reads of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

REVIDR_EL1, bit [28]

Trap MRS reads of REVIDR_EL1 at EL1 using AArch64 to EL2.

REVIDR_EL1	Meaning
0b0	MRS reads of REVIDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of REVIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

PAR_EL1, bit [27]

Trap `MRS` or `MRRS` reads of PAR_EL1 at EL1 using AArch64 to EL2.

PAR_EL1	Meaning
`0b0`	`MRS` and `MRRS` reads of PAR_EL1 are not trapped by this mechanism.
`0b1`	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the read generates a higher priority exception: • `MRS` reads of PAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`. • `MRRS` reads of PAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x14`.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

MPIDR_EL1, bit [26]

Trap `MRS` reads of MPIDR_EL1 at EL1 using AArch64 to EL2.

MPIDR_EL1	Meaning
`0b0`	`MRS` reads of MPIDR_EL1 are not trapped by this mechanism.
`0b1`	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then `MRS` reads of MPIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

MIDR_EL1, bit [25]

Trap `MRS` reads of MIDR_EL1 at EL1 using AArch64 to EL2.

MIDR_EL1	Meaning
`0b0`	`MRS` reads of MIDR_EL1 are not trapped by this mechanism.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

MIDR_EL1	Meaning
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of MIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

MAIR_EL1, bit [24]

Trap MRS reads of MAIR_EL1 at EL1 using AArch64 to EL2.

MAIR_EL1	Meaning
0b0	MRS reads of MAIR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of MAIR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

LORSA_EL1, bit [23]

When FEAT_LOR is implemented:

Trap MRS reads of LORSA_EL1 at EL1 using AArch64 to EL2.

LORSA_EL1	Meaning
0b0	MRS reads of LORSA_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of LORSA_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

LORN_EL1, bit [22]

When FEAT_LOR is implemented:

Trap MRS reads of LORN_EL1 at EL1 using AArch64 to EL2.

LORN_EL1	Meaning
0b0	MRS reads of LORN_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of LORN_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

LORID_EL1, bit [21]

When FEAT_LOR is implemented:

Trap MRS reads of LORID_EL1 at EL1 using AArch64 to EL2.

LORID_EL1	Meaning
0b0	MRS reads of LORID_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of LORID_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

LOREA_EL1, bit [20]

When FEAT_LOR is implemented:

Trap MRS reads of LOREA_EL1 at EL1 using AArch64 to EL2.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

LOREA_EL1	Meaning
0b0	MRS reads of LOREA_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of LOREA_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

LORC_EL1, bit [19]

When FEAT_LOR is implemented:

Trap MRS reads of LORC_EL1 at EL1 using AArch64 to EL2.

LORC_EL1	Meaning
0b0	MRS reads of LORC_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of LORC_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ISR_EL1, bit [18]

Trap MRS reads of ISR_EL1 at EL1 using AArch64 to EL2.

ISR_EL1	Meaning
0b0	MRS reads of ISR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ISR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

FAR_EL1, bit [17]

Trap `MRS` reads of FAR_EL1 at EL1 using AArch64 to EL2.

FAR_EL1	Meaning
`0b0`	`MRS` reads of FAR_EL1 are not trapped by this mechanism.
`0b1`	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then `MRS` reads of FAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

ESR_EL1, bit [16]

Trap `MRS` reads of ESR_EL1 at EL1 using AArch64 to EL2.

ESR_EL1	Meaning
`0b0`	`MRS` reads of ESR_EL1 are not trapped by this mechanism.
`0b1`	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then `MRS` reads of ESR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

DCZID_EL0, bit [15]

Trap `MRS` reads of DCZID_EL0 at EL1 and EL0 using AArch64 to EL2.

DCZID_EL0	Meaning
`0b0`	`MRS` reads of DCZID_EL0 are not trapped by this mechanism.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

DCZID_EL0	Meaning
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of DCZID_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

CTR_EL0, bit [14]

Trap MRS reads of CTR_EL0 at EL1 and EL0 using AArch64 to EL2.

CTR_EL0	Meaning
0b0	MRS reads of CTR_EL0 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of CTR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

CSSELR_EL1, bit [13]

Trap MRS reads of CSSELR_EL1 at EL1 using AArch64 to EL2.

CSSELR_EL1	Meaning
0b0	MRS reads of CSSELR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of CSSELR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

CPACR_EL1, bit [12]

Trap MRS reads of CPACR_EL1 at EL1 using AArch64 to EL2.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CPACR_EL1	Meaning
0b0	MRS reads of CPACR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of CPACR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

CONTEXTIDR_EL1, bit [11]

Trap MRS reads of CONTEXTIDR_EL1 at EL1 using AArch64 to EL2.

CONTEXTIDR_EL1	Meaning
0b0	MRS reads of CONTEXTIDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of CONTEXTIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

CLIDR_EL1, bit [10]

Trap MRS reads of CLIDR_EL1 at EL1 using AArch64 to EL2.

CLIDR_EL1	Meaning
0b0	MRS reads of CLIDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of CLIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CCSIDR_EL1, bit [9]

Trap MRS reads of CCSIDR_EL1 at EL1 using AArch64 to EL2.

CCSIDR_EL1	Meaning
0b0	MRS reads of CCSIDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of CCSIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

APIBKey, bit [8]

When FEAT_PAuth is implemented:

Trap MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APIBKeyHi_EL1.
- APIBKeyLo_EL1.

APIBKey	Meaning
0b0	MRS reads of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

APIAKey, bit [7]

When FEAT_PAuth is implemented:

Trap MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APIAKeyHi_EL1.
- APIAKeyLo_EL1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

APIAKey	Meaning
0b0	MRS reads of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

APGAKey, bit [6]

When FEAT_PAuth is implemented:

Trap MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APGAKeyHi_EL1.
- APGAKeyLo_EL1.

APGAKey	Meaning
0b0	MRS reads of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

APDBKey, bit [5]

When FEAT_PAuth is implemented:

Trap MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APDBKeyHi_EL1.
- APDBKeyLo_EL1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

APDBKey	Meaning
0b0	MRS reads of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

APDAKey, bit [4]

When FEAT_PAuth is implemented:

Trap MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APDAKeyHi_EL1.
- APDAKeyLo_EL1.

APDAKey	Meaning
0b0	MRS reads of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AMAIR_EL1, bit [3]

Trap MRS reads of AMAIR_EL1 at EL1 using AArch64 to EL2.

AMAIR_EL1	Meaning
0b0	MRS reads of AMAIR_EL1 are not trapped by this mechanism.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

AMAIR_EL1	Meaning
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of AMAIR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

AIDR_EL1, bit [2]

Trap MRS reads of AIDR_EL1 at EL1 using AArch64 to EL2.

AIDR_EL1	Meaning
0b0	MRS reads of AIDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of AIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

AFSR1_EL1, bit [1]

Trap MRS reads of AFSR1_EL1 at EL1 using AArch64 to EL2.

AFSR1_EL1	Meaning
0b0	MRS reads of AFSR1_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of AFSR1_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

AFSR0_EL1, bit [0]

Trap MRS reads of AFSR0_EL1 at EL1 using AArch64 to EL2.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

AFSR0_EL1	Meaning
0b0	MRS reads of AFSR0_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of AFSR0_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Accessing HFGRTR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, HFGRTR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0001	0b100

```
if !(IsFeatureImplemented(FEAT_FGT) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x1B8];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FGTEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = HFGRTR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = HFGRTR_EL2;
```

```
MSR HFGRTR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0001	0b100

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_FGT) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x1B8] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FGTEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        HFGRTR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    HFGRTR_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.74 HFGWTR2_EL2, Hypervisor Fine-Grained Write Trap Register 2

The HFGWTR2_EL2 characteristics are:

Purpose

Provides controls for traps of MSRR, MSR and MCR writes of System registers.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_FGT2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to HFGWTR2_EL2 are UNDEFINED.

Attributes

HFGWTR2_EL2 is a 64-bit register.

Field descriptions

Bits [63:15]

Reserved, RES0.

nACTLRALIAS_EL1, bit [14]

When FEAT_SRMASK is implemented:

Trap MSR writes of ACTLRALIAS_EL1 at EL1 using AArch64 to EL2.

nACTLRALIAS_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MSR writes of ACTLRALIAS_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of ACTLRALIAS_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nACTLRMASK_EL1, bit [13]

When FEAT_SRMASK is implemented:

Trap MSR writes of ACTLRMASK_EL1 at EL1 using AArch64 to EL2.

nACTLRMASK_EL1	Meaning
`0b0`	If EL2 is implemented and enabled in the current Security state, then MSR writes of ACTLRMASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the write generates a higher priority exception.
`0b1`	MSR writes of ACTLRMASK_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nTCR2ALIAS_EL1, bit [12]

When FEAT_SRMASK is implemented:

Trap MSR writes of TCR2ALIAS_EL1 at EL1 using AArch64 to EL2.

nTCR2ALIAS_EL1	Meaning
`0b0`	If EL2 is implemented and enabled in the current Security state, then MSR writes of TCR2ALIAS_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the write generates a higher priority exception.
`0b1`	MSR writes of TCR2ALIAS_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nTCRALIAS_EL1, bit [11]

When FEAT_SRMASK is implemented:

Trap MSR writes of TCRALIAS_EL1 at EL1 using AArch64 to EL2.

nTCRALIAS_EL1	Meaning
`0b0`	If EL2 is implemented and enabled in the current Security state, then MSR writes of TCRALIAS_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the write generates a higher priority exception.
`0b1`	MSR writes of TCRALIAS_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSCTLR2ALIAS_EL1, bit [10]

When FEAT_SRMASK is implemented:

Trap MSR writes of SCTLR2ALIAS_EL1 at EL1 using AArch64 to EL2.

nSCTLR2ALIAS_EL1	Meaning
`0b0`	If EL2 is implemented and enabled in the current Security state, then MSR writes of SCTLR2ALIAS_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the write generates a higher priority exception.
`0b1`	MSR writes of SCTLR2ALIAS_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSCTLRALIAS_EL1, bit [9]

When FEAT_SRMASK is implemented:

Trap MSR writes of SCTLRALIAS_EL1 at EL1 using AArch64 to EL2.

nSCTLRALIAS_EL1	Meaning
`0b0`	If EL2 is implemented and enabled in the current Security state, then MSR writes of SCTLRALIAS_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the write generates a higher priority exception.
`0b1`	MSR writes of SCTLRALIAS_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nCPACRALIAS_EL1, bit [8]

When FEAT_SRMASK is implemented:

Trap MSR writes of CPACRALIAS_EL1 at EL1 using AArch64 to EL2.

nCPACRALIAS_EL1	Meaning
`0b0`	If EL2 is implemented and enabled in the current Security state, then MSR writes of CPACRALIAS_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the write generates a higher priority exception.
`0b1`	MSR writes of CPACRALIAS_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nTCR2MASK_EL1, bit [7]

When FEAT_SRMASK is implemented:

Trap MSR writes of TCR2MASK_EL1 at EL1 using AArch64 to EL2.

nTCR2MASK_EL1	Meaning
`0b0`	If EL2 is implemented and enabled in the current Security state, then MSR writes of TCR2MASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the write generates a higher priority exception.
`0b1`	MSR writes of TCR2MASK_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nTCRMASK_EL1, bit [6]

When FEAT_SRMASK is implemented:

Trap MSR writes of TCRMASK_EL1 at EL1 using AArch64 to EL2.

nTCRMASK_EL1	Meaning
`0b0`	If EL2 is implemented and enabled in the current Security state, then MSR writes of TCRMASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the write generates a higher priority exception.
`0b1`	MSR writes of TCRMASK_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSCTLR2MASK_EL1, bit [5]

When FEAT_SRMASK is implemented:

Trap MSR writes of SCTLR2MASK_EL1 at EL1 using AArch64 to EL2.

nSCTLR2MASK_EL1	Meaning
`0b0`	If EL2 is implemented and enabled in the current Security state, then MSR writes of SCTLR2MASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the write generates a higher priority exception.
`0b1`	MSR writes of SCTLR2MASK_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nSCTLRMASK_EL1, bit [4]

When FEAT_SRMASK is implemented:

Trap MSR writes of SCTLRMASK_EL1 at EL1 using AArch64 to EL2.

nSCTLRMASK_EL1	Meaning
`0b0`	If EL2 is implemented and enabled in the current Security state, then MSR writes of SCTLRMASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the write generates a higher priority exception.
`0b1`	MSR writes of SCTLRMASK_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nCPACRMASK_EL1, bit [3]

When FEAT_SRMASK is implemented:

Trap `MSR` writes of CPACRMASK_EL1 at EL1 using AArch64 to EL2.

nCPACRMASK_EL1	Meaning
`0b0`	If EL2 is implemented and enabled in the current Security state, then `MSR` writes of CPACRMASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the write generates a higher priority exception.
`0b1`	`MSR` writes of CPACRMASK_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nRCWSMASK_EL1, bit [2]

When FEAT_THE is implemented:

Trap `MSR` or `MSRR` writes of RCWSMASK_EL1 at EL1 using AArch64 to EL2.

nRCWSMASK_EL1	Meaning
`0b0`	If EL2 is implemented and enabled in the current Security state, then unless the write generates a higher priority exception: • `MSR` writes of RCWSMASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`. • `MSRR` writes of RCWSMASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x14`.
`0b1`	`MSR` and `MSRR` writes of RCWSMASK_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [1]

Reserved, RES0.

nPFAR_EL1, bit [0]

When FEAT_PFAR is implemented:

Trap MSR writes of PFAR_EL1 at EL1 using AArch64 to EL2.

nPFAR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, then MSR writes of PFAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of PFAR_EL1 are not trapped by this mechanism.

This field is ignored by the PE and treated as zero when all of the following are true:

- EL3 is implemented.
- SCR_EL3.FGTEn2 == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing HFGWTR2_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, HFGWTR2_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0011	0b0001	0b011

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_FGT2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x2C8];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FGTEn2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = HFGWTR2_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = HFGWTR2_EL2;
```

MSR HFGWTR2_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0011	0b0001	0b011

```
if !(IsFeatureImplemented(FEAT_FGT2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x2C8] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FGTEn2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        HFGWTR2_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    HFGWTR2_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.75 HFGWTR_EL2, Hypervisor Fine-Grained Write Trap Register

The HFGWTR_EL2 characteristics are:

Purpose

Provides controls for traps of MSRR, MSR and MCR writes of System registers.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_FGT is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to HFGWTR_EL2 are UNDEFINED.

Attributes

HFGWTR_EL2 is a 64-bit register.

Field descriptions

nAMAIR2_EL1, bit [63]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_AIE is implemented:

Trap MSR writes of AMAIR2_EL1 at EL1 using AArch64 to EL2.

nAMAIR2_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of AMAIR2_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of AMAIR2_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nMAIR2_EL1, bit [62]

When FEAT_AIE is implemented:

Trap MSR writes of MAIR2_EL1 at EL1 using AArch64 to EL2.

nMAIR2_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of MAIR2_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of MAIR2_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nS2POR_EL1, bit [61]

When FEAT_S2POE is implemented:

Trap MSR writes of S2POR_EL1 at EL1 using AArch64 to EL2.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

nS2POR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of S2POR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of S2POR_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nPOR_EL1, bit [60]

When FEAT_S1POE is implemented:

Trap MSR writes of POR_EL1 at EL1 using AArch64 to EL2.

nPOR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of POR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of POR_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nPOR_EL0, bit [59]

When FEAT_S1POE is implemented:

Trap MSR writes of POR_EL0 at EL1 and EL0 using AArch64 to EL2.

nPOR_EL0	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of POR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

nPOR_EL0	Meaning
0b1	MSR writes of POR_EL0 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nPIR_EL1, bit [58]

When FEAT_S1PIE is implemented:

Trap MSR writes of PIR_EL1 at EL1 using AArch64 to EL2.

nPIR_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of PIR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of PIR_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nPIRE0_EL1, bit [57]

When FEAT_S1PIE is implemented:

Trap MSR writes of PIRE0_EL1 at EL1 using AArch64 to EL2.

nPIRE0_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of PIRE0_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of PIRE0_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- When the highest implemented Exception level is EL2, this field resets to `'0'`.
- Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nRCWMASK_EL1, bit [56]

When FEAT_THE is implemented:

Trap `MSR` or `MSRR` writes of RCWMASK_EL1 at EL1 using AArch64 to EL2.

nRCWMASK_EL1	Meaning
`0b0`	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the write generates a higher priority exception: • `MSR` writes of RCWMASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`. • `MSRR` writes of RCWMASK_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x14`.
`0b1`	`MSR` and `MSRR` writes of RCWMASK_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nTPIDR2_EL0, bit [55]

When FEAT_SME is implemented:

Trap `MSR` writes of TPIDR2_EL0 at EL1 and EL0 using AArch64 to EL2.

nTPIDR2_EL0	Meaning
`0b0`	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then `MSR` writes of TPIDR2_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the write generates a higher priority exception.
`0b1`	`MSR` writes of TPIDR2_EL0 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

nSMPRI_EL1, bit [54]

When FEAT_SME is implemented:

Trap MSR writes of SMPRI_EL1 at EL1 using AArch64 to EL2.

nSMPRI_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of SMPRI_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of SMPRI_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nGCS_EL1, bit [53]

When FEAT_GCS is implemented:

Trap MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- GCSCR_EL1.
- GCSPR_EL1.

nGCS_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of the specified System registers are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nGCS_EL0, bit [52]

When FEAT_GCS is implemented:

Trap MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- GCSCRE0_EL1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- GCSPR_EL0.

nGCS_EL0	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of the specified System registers are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [51]

Reserved, RES0.

nACCDATA_EL1, bit [50]

When FEAT_LS64_ACCDATA is implemented:

Trap MSR writes of ACCDATA_EL1 at EL1 using AArch64 to EL2.

nACCDATA_EL1	Meaning
0b0	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ACCDATA_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.
0b1	MSR writes of ACCDATA_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ERXADDR_EL1, bit [49]

When FEAT_RAS is implemented:

Trap MSR writes of ERXADDR_EL1 at EL1 using AArch64 to EL2.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ERXADDR_EL1	Meaning
0b0	MSR writes of ERXADDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ERXADDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ERXPFGCDN_EL1, bit [48]

When FEAT_RASv1p1 is implemented:

Trap MSR writes of ERXPFGCDN_EL1 at EL1 using AArch64 to EL2.

ERXPFGCDN_EL1	Meaning
0b0	MSR writes of ERXPFGCDN_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ERXPFGCDN_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ERXPFGCTL_EL1, bit [47]

When FEAT_RASv1p1 is implemented:

Trap MSR writes of ERXPFGCTL_EL1 at EL1 using AArch64 to EL2.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ERXPFGCTL_EL1	Meaning
0b0	MSR writes of ERXPFGCTL_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ERXPFGCTL_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [46]

Reserved, RES0.

ERXMISCn_EL1, bit [45]

When FEAT_RAS is implemented:

Trap MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- ERXMISC0_EL1.
- ERXMISC1_EL1.
- ERXMISC2_EL1.
- ERXMISC3_EL1.

ERXMISCn_EL1	Meaning
0b0	MSR writes of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ERXSTATUS_EL1, bit [44]

When FEAT_RAS is implemented:

Trap MSR writes of ERXSTATUS_EL1 at EL1 using AArch64 to EL2.

ERXSTATUS_EL1	Meaning
`0b0`	MSR writes of ERXSTATUS_EL1 are not trapped by this mechanism.
`0b1`	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ERXSTATUS_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the write generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ERXCTLR_EL1, bit [43]

When FEAT_RAS is implemented:

Trap MSR writes of ERXCTLR_EL1 at EL1 using AArch64 to EL2.

ERXCTLR_EL1	Meaning
`0b0`	MSR writes of ERXCTLR_EL1 are not trapped by this mechanism.
`0b1`	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ERXCTLR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the write generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [42]

Reserved, RES0.

ERRSELR_EL1, bit [41]

When FEAT_RAS is implemented:

Trap MSR writes of ERRSELR_EL1 at EL1 using AArch64 to EL2.

ERRSELR_EL1	Meaning
`0b0`	MSR writes of ERRSELR_EL1 are not trapped by this mechanism.
`0b1`	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ERRSELR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the write generates a higher priority exception.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [40]

Reserved, RES0.

ICC_IGRPENn_EL1, bit [39]

When GICv3 is implemented:

Trap MSR writes of ICC_IGRPEN<n>_EL1 at EL1 using AArch64 to EL2.

ICC_IGRPENn_EL1	Meaning
`0b0`	MSR writes of ICC_IGRPEN<n>_EL1 are not trapped by this mechanism.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ICC_IGRPENn_EL1	Meaning
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ICC_IGRPEN<n>_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VBAR_EL1, bit [38]

Trap MSR writes of VBAR_EL1 at EL1 using AArch64 to EL2.

VBAR_EL1	Meaning
0b0	MSR writes of VBAR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of VBAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TTBR1_EL1, bit [37]

Trap MSR or MSRR writes of TTBR1_EL1 at EL1 using AArch64 to EL2.

TTBR1_EL1	Meaning
0b0	MSR and MSRR writes of TTBR1_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the write generates a higher priority exception: • MSR writes of TTBR1_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • MSRR writes of TTBR1_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x14.

The reset behavior of this field is:

- On a Warm reset:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- When the highest implemented Exception level is EL2, this field resets to `'0'`.
- Otherwise, this field resets to an architecturally UNKNOWN value.

TTBR0_EL1, bit [36]

Trap `MSR` or `MSRR` writes of TTBR0_EL1 at EL1 using AArch64 to EL2.

TTBR0_EL1	Meaning
`0b0`	`MSR` and `MSRR` writes of TTBR0_EL1 are not trapped by this mechanism.
`0b1`	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the write generates a higher priority exception: • `MSR` writes of TTBR0_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`. • `MSRR` writes of TTBR0_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x14`.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TPIDR_EL0, bit [35]

Trap `MSR` writes of TPIDR_EL0 at EL1 and EL0 using AArch64 and `MCR` writes of TPIDRURW at EL0 using AArch32 when EL1 is using AArch64 to EL2.

TPIDR_EL0	Meaning
`0b0`	`MSR` writes of TPIDR_EL0 at EL1 and EL0 using AArch64 and `MCR` writes of TPIDRURW at EL0 using AArch32 are not trapped by this mechanism.
`0b1`	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the write generates a higher priority exception: • `MSR` writes of TPIDR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`. • `MCR` writes of TPIDRURW at EL0 using AArch32 when EL1 is using AArch64 are trapped to EL2 and reported with EC syndrome value `0x03`.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TPIDRRO_EL0, bit [34]

Trap `MSR` writes of TPIDRRO_EL0 at EL1 using AArch64 to EL2.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TPIDRRO_EL0	Meaning
0b0	MSR writes of TPIDRRO_EL0 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TPIDRRO_EL0 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TPIDR_EL1, bit [33]

Trap MSR writes of TPIDR_EL1 at EL1 using AArch64 to EL2.

TPIDR_EL1	Meaning
0b0	MSR writes of TPIDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TPIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TCR_EL1, bit [32]

Trap MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- TCR_EL1.
- TCR2_EL1, if FEAT_TCR2 is implemented.

TCR_EL1	Meaning
0b0	MSR writes of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

SCXTNUM_EL0, bit [31]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented:

Trap MSR writes of SCXTNUM_EL0 at EL1 and EL0 using AArch64 to EL2.

SCXTNUM_EL0	Meaning
0b0	MSR writes of SCXTNUM_EL0 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of SCXTNUM_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SCXTNUM_EL1, bit [30]

When FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented:

Trap MSR writes of SCXTNUM_EL1 at EL1 using AArch64 to EL2.

SCXTNUM_EL1	Meaning
0b0	MSR writes of SCXTNUM_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of SCXTNUM_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SCTLR_EL1, bit [29]

Trap MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- SCTLR_EL1.
- SCTLR2_EL1, if FEAT_SCTLR2 is implemented.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SCTLR_EL1	Meaning
0b0	MSR writes of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [28]

Reserved, RES0.

PAR_EL1, bit [27]

Trap MSR or MSRR writes of PAR_EL1 at EL1 using AArch64 to EL2.

PAR_EL1	Meaning
0b0	MSR and MSRR writes of PAR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then unless the write generates a higher priority exception: • MSR writes of PAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. • MSRR writes of PAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x14.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bits [26:25]

Reserved, RES0.

MAIR_EL1, bit [24]

Trap MSR writes of MAIR_EL1 at EL1 using AArch64 to EL2.

MAIR_EL1	Meaning
0b0	MSR writes of MAIR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of MAIR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

LORSA_EL1, bit [23]

When FEAT_LOR is implemented:

Trap MSR writes of LORSA_EL1 at EL1 using AArch64 to EL2.

LORSA_EL1	Meaning
`0b0`	MSR writes of LORSA_EL1 are not trapped by this mechanism.
`0b1`	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of LORSA_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

LORN_EL1, bit [22]

When FEAT_LOR is implemented:

Trap MSR writes of LORN_EL1 at EL1 using AArch64 to EL2.

LORN_EL1	Meaning
`0b0`	MSR writes of LORN_EL1 are not trapped by this mechanism.
`0b1`	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of LORN_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value `0x18`, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [21]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

LOREA_EL1, bit [20]

When FEAT_LOR is implemented:

Trap MSR writes of LOREA_EL1 at EL1 using AArch64 to EL2.

LOREA_EL1	Meaning
0b0	MSR writes of LOREA_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of LOREA_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

LORC_EL1, bit [19]

When FEAT_LOR is implemented:

Trap MSR writes of LORC_EL1 at EL1 using AArch64 to EL2.

LORC_EL1	Meaning
0b0	MSR writes of LORC_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of LORC_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [18]

Reserved, RES0.

FAR_EL1, bit [17]

Trap MSR writes of FAR_EL1 at EL1 using AArch64 to EL2.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FAR_EL1	Meaning
0b0	MSR writes of FAR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of FAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

ESR_EL1, bit [16]

Trap MSR writes of ESR_EL1 at EL1 using AArch64 to EL2.

ESR_EL1	Meaning
0b0	MSR writes of ESR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ESR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bits [15:14]

Reserved, RES0.

CSSELR_EL1, bit [13]

Trap MSR writes of CSSELR_EL1 at EL1 using AArch64 to EL2.

CSSELR_EL1	Meaning
0b0	MSR writes of CSSELR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of CSSELR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CPACR_EL1, bit [12]

Trap MSR writes of CPACR_EL1 at EL1 using AArch64 to EL2.

CPACR_EL1	Meaning
0b0	MSR writes of CPACR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of CPACR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

CONTEXTIDR_EL1, bit [11]

Trap MSR writes of CONTEXTIDR_EL1 at EL1 using AArch64 to EL2.

CONTEXTIDR_EL1	Meaning
0b0	MSR writes of CONTEXTIDR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of CONTEXTIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bits [10:9]

Reserved, RES0.

APIBKey, bit [8]

When FEAT_PAuth is implemented:

Trap MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APIBKeyHi_EL1.
- APIBKeyLo_EL1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

APIBKey	Meaning
0b0	MSR writes of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

APIAKey, bit [7]

When FEAT_PAuth is implemented:

Trap MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APIAKeyHi_EL1.
- APIAKeyLo_EL1.

APIAKey	Meaning
0b0	MSR writes of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

APGAKey, bit [6]

When FEAT_PAuth is implemented:

Trap MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APGAKeyHi_EL1.
- APGAKeyLo_EL1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

APGAKey	Meaning
0b0	MSR writes of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

APDBKey, bit [5]

When FEAT_PAuth is implemented:

Trap MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APDBKeyHi_EL1.
- APDBKeyLo_EL1.

APDBKey	Meaning
0b0	MSR writes of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

APDAKey, bit [4]

When FEAT_PAuth is implemented:

Trap MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APDAKeyHi_EL1.
- APDAKeyLo_EL1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

APDAKey	Meaning
0b0	MSR writes of the specified System registers are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the specified System registers are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AMAIR_EL1, bit [3]

Trap MSR writes of AMAIR_EL1 at EL1 using AArch64 to EL2.

AMAIR_EL1	Meaning
0b0	MSR writes of AMAIR_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of AMAIR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [2]

Reserved, RES0.

AFSR1_EL1, bit [1]

Trap MSR writes of AFSR1_EL1 at EL1 using AArch64 to EL2.

AFSR1_EL1	Meaning
0b0	MSR writes of AFSR1_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of AFSR1_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- When the highest implemented Exception level is EL2, this field resets to `'0'`.
- Otherwise, this field resets to an architecturally UNKNOWN value.

AFSR0_EL1, bit [0]

Trap MSR writes of AFSR0_EL1 at EL1 using AArch64 to EL2.

AFSR0_EL1	Meaning
0b0	MSR writes of AFSR0_EL1 are not trapped by this mechanism.
0b1	If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of AFSR0_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Accessing HFGWTR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, HFGWTR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0001	0b101

```
if !(IsFeatureImplemented(FEAT_FGT) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x1C0];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FGTEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = HFGWTR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = HFGWTR_EL2;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MSR HFGWTR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0001	0b101

```
if !(IsFeatureImplemented(FEAT_FGT) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x1C0] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FGTEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        HFGWTR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    HFGWTR_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.76 HPFAR_EL2, Hypervisor IPA Fault Address Register

The HPFAR_EL2 characteristics are:

Purpose

Holds the faulting IPA for some aborts on a stage 2 translation taken to EL2 and GPC exceptions due to a fault on an access for a stage 2 translation.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

The HPFAR_EL2 is written for:

- A Translation fault, Access Flag fault, or Address Size fault on a stage 2 translation not on a stage 1 translation table walk.
- A Translation fault, Access Flag fault, Address Size fault, or Permission fault on stage 2 translation of an address accessed in a stage 1 translation table walk.
- If FEAT_RME is implemented, a Granule Protection Check fault in the second stage of translation.

For all other exceptions taken to EL2, this register is UNKNOWN.

Note

The address held in this register is an address accessed by the instruction fetch or data access that caused the exception that gave rise to the Instruction Abort exception or Data Abort exception. It is the lower address that gave rise to the fault that is reported. Where different faults from different addresses arise from the same instruction, such as for an instruction that loads or stores an unaligned address that crosses a page boundary, the architecture does not prioritize which fault is reported.

AArch64 System register HPFAR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HPFAR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to HPFAR_EL2 are UNDEFINED.

Attributes

HPFAR_EL2 is a 64-bit register.

Field descriptions

Execution at EL1 or EL0 makes HPFAR_EL2 become UNKNOWN.

NS, bit [63]

When FEAT_SEL2 is implemented:

Faulting IPA address space.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

NS	Meaning
0b0	Faulting IPA is from the Secure IPA space.
0b1	Faulting IPA is from the Non-secure IPA space.

For Data Abort exceptions or Instruction Abort exceptions taken to Non-secure EL2:

- This field is RES0.
- The address is from the Non-secure IPA space.

If FEAT_RME is implemented, for Data Abort exceptions or Instruction Abort exceptions taken to Realm EL2:

- This field is RES0.
- The address is from the Realm IPA space.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

FIPA, bits [47:4]

When FEAT_D128 is implemented:

FIPA, bits [43:0]

Bits [55:12] of the Faulting Intermediate Physical Address.

For implementations with fewer than 55 physical address bits, the corresponding upper bits in this field are RES0.

When FEAT_MOPS is implemented, the value presented in FIPA on a synchronous exception that set the HPFAR_EL2 from any of the Memory Copy and Memory Set instructions is within the address range of the current stage 2 translation granule, aligned to the size of the current stage 2 translation granule, of the address that generated the Data abort. In this case, bits[(n-1):0] of the value are UNKNOWN, where 2^n is the current stage 2 translation granule size in bytes.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_LPA is implemented and FEAT_D128 is not implemented:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bits [43:40]

Reserved, RES0.

FIPA, bits [39:0]

Bits [51:12] of the Faulting Intermediate Physical Address.

For implementations with fewer than 52 physical address bits, the corresponding upper bits in this field are RES0.

When FEAT_MOPS is implemented, the value presented in FIPA on a synchronous exception that set the HPFAR_EL2 from any of the Memory Copy and Memory Set instructions is within the address range of the current stage 2 translation granule, aligned to the size of the current stage 2 translation granule, of the address that generated the Data abort. In this case, bits[(n-1):0] of the value are UNKNOWN, where 2^n is the current stage 2 translation granule size in bytes.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_LPA is not implemented:

Bits [43:36]

Reserved, RES0.

FIPA, bits [35:0]

Bits[47:12] Faulting Intermediate Physical Address.

For implementations with fewer than 48 physical address bits, the corresponding upper bits in this field are RES0.

When FEAT_MOPS is implemented, the value presented in FIPA on a synchronous exception that set the HPFAR_EL2 from any of the Memory Copy and Memory Set instructions is within the address range of the current stage 2 translation granule, aligned to the size of the current stage 2 translation granule, of the address that generated the Data abort. In this case, bits[(n-1):0] of the value are UNKNOWN, where 2^n is the current stage 2 translation granule size in bytes.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [3:0]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing HPFAR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, HPFAR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0110	0b0000	0b100

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = HPFAR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = HPFAR_EL2;
```

```
MSR HPFAR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0110	0b0000	0b100

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HPFAR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    HPFAR_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.77 HSTR_EL2, Hypervisor System Trap Register

The HSTR_EL2 characteristics are:

Purpose

Controls trapping to EL2 of EL1 or lower AArch32 accesses to the System register in the coproc == `0b1111` encoding space, by the CRn value used to access the register using MCR or MRC instruction. When the register is accessible using an MCRR or MRRC instruction, this is the CRm value used to access the register.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

AArch64 System register HSTR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HSTR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to HSTR_EL2 are UNDEFINED.

Attributes

HSTR_EL2 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:

Bits [63:16, 14, 4]

Reserved, RES0.

T<n> , bits [n], for n = 15 to 15, 13 to 5, 3 to 0

The remaining fields control whether EL0 and EL1 accesses, using MCR, MRC, MCRR, and MRRC instructions, to the System registers in the coproc == `0b1111` encoding space, are trapped to EL2 as follows:

- MCR or MRC accesses to these registers that are trapped to EL2 are reported using EC syndrome value `0x03`, unless the access is UNDEFINED.
- MCRR or MRRC accesses to these registers that are trapped to EL2 are reported using EC syndrome value `0x04`, unless the access is UNDEFINED.

T<n>	Meaning
`0b0`	This control has no effect on EL0 or EL1 accesses to System registers.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T<n>	Meaning
0b1	System registers in the coproc == 0b1111 encoding space and CRn == <n> or CRm == <n> where T<n> is the name of this field, are trapped as follows: • An EL1 MCR or MRC access is trapped to EL2. • An EL0 MCR or MRC access is trapped to EL2, if the access is not UNDEFINED when the value of this field is 0. • An EL1 MCRR or MRRC access is trapped to EL2. • An EL0 MCRR or MRRC access is trapped to EL2, if the access is not UNDEFINED when the value of this field is 0. It is IMPLEMENTATION DEFINED whether an EL0 access using AArch32 is trapped to EL2, or is UNDEFINED. If the access is UNDEFINED, and generates an exception that is taken to EL1 or EL2 using AArch64, this is reported with EC syndrome value 0x00. **Note** Arm expects that trapping to EL2 of EL0 accesses to these registers is unusual and used only when the hypervisor must virtualize EL0 operation. Arm recommends that, whenever possible, EL0 accesses to these registers behave as they would if the implementation did not include EL2. This means that, if the architecture does not support the EL0 access, then the register access instruction is treated as UNDEFINED and generates an exception that is taken to EL1.

For example, when HSTR_EL2.T7 is 1, for instructions executed at EL1:

- An MCR or MRC instruction with coproc set to 0b1111 and <CRn> set to c7 is trapped to EL2.
- An MCRR or MRRC instruction with coproc set to 0b1111 and <CRm> set to c7 is trapped to EL2.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Bits [63:0]

Reserved, RES0.

Accessing HSTR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, HSTR_EL2
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0001	0b011

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x080];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = HSTR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = HSTR_EL2;
```

MSR HSTR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0001	0b011

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x080] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HSTR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    HSTR_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.78 ID_AA64AFR0_EL1, AArch64 Auxiliary Feature Register 0

The ID_AA64AFR0_EL1 characteristics are:

Purpose

Provides information about the IMPLEMENTATION DEFINED features of the PE in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64AFR0_EL1 are UNDEFINED.

Attributes

ID_AA64AFR0_EL1 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

IMPLEMENTATION DEFINED, bits [31:28]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [27:24]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [23:20]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [19:16]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [15:12]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [11:8]

IMPLEMENTATION DEFINED.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

IMPLEMENTATION DEFINED, bits [7:4]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [3:0]

IMPLEMENTATION DEFINED.

Accessing ID_AA64AFR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64AFR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0101	0b100

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64AFR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64AFR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64AFR0_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.79 ID_AA64AFR1_EL1, AArch64 Auxiliary Feature Register 1

The ID_AA64AFR1_EL1 characteristics are:

Purpose

Reserved for future expansion of information about the IMPLEMENTATION DEFINED features of the PE in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64AFR1_EL1 are UNDEFINED.

Attributes

ID_AA64AFR1_EL1 is a 64-bit register.

Field descriptions

Bits [63:0]

Reserved, RES0.

Accessing ID_AA64AFR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64AFR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0101	0b101

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
        ↪then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64AFR1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64AFR1_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64AFR1_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.80 ID_AA64DFR0_EL1, AArch64 Debug Feature Register 0

The ID_AA64DFR0_EL1 characteristics are:

Purpose

Provides top-level information about the debug system in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

The external register EDDFR gives information from this register.

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64DFR0_EL1 are UNDEFINED.

Attributes

ID_AA64DFR0_EL1 is a 64-bit register.

Field descriptions

HPMN0, bits [63:60]

Zero PMU event counters for a Guest operating system.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HPMN0	Meaning
0b0000	Setting MDCR_EL2.HPMN to zero has CONSTRAINED UNPREDICTABLE behavior.
0b0001	Setting MDCR_EL2.HPMN to zero has defined behavior.

All other values are reserved.

FEAT_HPMN0 implements the functionality identified by the value 0b0001.

From Armv8.8, in an implementation that includes FEAT_PMUv3, FEAT_FGT, and EL2, the value 0b0000 is not permitted.

Access to this field is RO.

ExtTrcBuff, bits [59:56]

Trace Buffer External Mode Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ExtTrcBuff	Meaning
0b0000	Trace Buffer External Mode not implemented.
0b0001	Trace Buffer External Mode implemented.

All other values are reserved.

FEAT_TRBE_EXT implements the functionality identified by the value 0b0001.

Access to this field is RO.

BRBE, bits [55:52]

Branch Record Buffer Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BRBE	Meaning
0b0000	Branch Record Buffer Extension not implemented.
0b0001	Branch Record Buffer Extension implemented.
0b0010	As 0b0001, and adds support for branch recording at EL3.

All other values are reserved.

FEAT_BRBE implements the functionality identified by the value 0b0001.

FEAT_BRBEv1p1 implements the functionality identified by the value 0b0010.

From Armv9.3, if FEAT_BRBE is implemented, the value 0b0001 is not permitted.

Access to this field is RO.

MTPMU, bits [51:48]

Multi-threaded PMU extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MTPMU	Meaning
0b0000	FEAT_MTPMU not implemented. If FEAT_PMUv3 is implemented, it is IMPLEMENTATION DEFINED whether PMEVTYPER<n>_EL0.MT and PMEVTYPER<n>.MT are read/write or RES0.
0b0001	FEAT_MTPMU and FEAT_PMUv3 implemented. PMEVTYPER<n>_EL0.MT and PMEVTYPER<n>.MT are read/write. When FEAT_MTPMU is disabled, the Effective values of PMEVTYPER<n>_EL0.MT and PMEVTYPER<n>.MT are 0.
0b1111	FEAT_MTPMU not implemented. If FEAT_PMUv3 is implemented, PMEVTYPER<n>_EL0.MT and PMEVTYPER<n>.MT are RES0.

All other values are reserved.

FEAT_MTPMU implements the functionality identified by the value 0b0001.

From Armv8.6, in an implementation that includes FEAT_PMUv3, the value 0b0000 is not permitted.

In an implementation that does not include FEAT_PMUv3, the value 0b0001 is not permitted.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Access to this field is RO.

TraceBuffer, bits [47:44]

Trace Buffer Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TraceBuffer	Meaning
0b0000	Trace Buffer Extension not implemented.
0b0001	Trace Buffer Extension implemented.
0b0010	As 0b0001, and adds: • If EL2 and FEAT_FGT are implemented, a fine-grained trap on the TSB CSYNC instruction. • If EL2 is implemented, an EL2 control to override TRBLIMITR_EL1.nVM. • The TRBE Profiling exception extension, FEAT_TRBE_EXC.

All other values are reserved.

FEAT_TRBE implements the functionality identified by the value 0b0001.

FEAT_TRBEv1p1 implements the functionality identified by the value 0b0010.

In any Armv9 implementation, if FEAT_ETE is implemented, the value 0b0000 is not permitted.

From Armv9.6, if FEAT_TRBE is implemented, the value 0b0001 is not permitted.

Access to this field is RO.

TraceFilt, bits [43:40]

Armv8.4 Self-hosted Trace Extension version.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TraceFilt	Meaning
0b0000	Armv8.4 Self-hosted Trace Extension not implemented.
0b0001	Armv8.4 Self-hosted Trace Extension implemented.

All other values are reserved.

FEAT_TRF implements the functionality identified by the value 0b0001.

From Armv8.4, if FEAT_ETMv4 is implemented, the value 0b0000 is not permitted.

If FEAT_ETE is implemented, the value 0b0000 is not permitted.

Access to this field is RO.

DoubleLock, bits [39:36]

OS Double Lock implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

DoubleLock	Meaning
0b0000	OS Double Lock implemented. OSDLR_EL1 is RW.
0b1111	OS Double Lock not implemented. OSDLR_EL1 is RAZ/WI.

All other values are reserved.

FEAT_DoubleLock implements the functionality identified by the value 0b0000.

In Armv8.0, the only permitted value is 0b0000.

If FEAT_Debugv8p2 is implemented and FEAT_DoPD is not implemented, the permitted values are 0b0000 and 0b1111.

If FEAT_DoPD is implemented, the only permitted value is 0b1111.

Access to this field is RO.

PMSVer, bits [35:32]

Statistical Profiling Extension version.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PMSVer	Meaning
0b0000	Statistical Profiling Extension not implemented.
0b0001	Statistical Profiling Extension implemented.
0b0010	As 0b0001, and adds: • Support for the Events packet Alignment flag. • If FEAT_SVE is implemented, support for the Scalable Vector extensions to Statistical Profiling.
0b0011	As 0b0010, and adds: • Discard mode. • Extended event filtering, including the PMSNEVFR_EL1 System register. • Support for the OPTIONAL previous branch target Address packet. • If FEAT_PMUv3 is implemented, controls to freeze the PMU event counters after an SPE buffer management event occurs. • If FEAT_PMUv3 is implemented, the SAMPLE_FEED_BR, SAMPLE_FEED_EVENT, SAMPLE_FEED_LAT, SAMPLE_FEED_LD, SAMPLE_FEED_OP, and SAMPLE_FEED_ST PMU events.
0b0100	As 0b0011, and adds: • If FEAT_MOPS is implemented, Operation Type packet encodings for Memory Copy and Set operations. • If FEAT_MTE is implemented, Operation Type packet encodings for loads and stores of Allocation Tags.
0b0101	As 0b0100, and adds: • Support for the Events packet Level 2 Data cache access, Level 2 Data cache miss, Cached data modified, Recently fetched cache line, and Cache snoop flags. • Support for Data Source filtering.
0b0110	As 0b0101, and adds: • If EL2 and FEAT_FGT are implemented, a fine-grained trap on the PSB CSYNC instruction. • The SPE Profiling exception extension, FEAT_SPE_EXC. • The Statistical Profiling physical addressing mode extension, FEAT_SPE_nVM.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

All other values are reserved.

FEAT_SPE implements the functionality identified by the value `0b0001`.

FEAT_SPEv1p1 implements the functionality identified by the value `0b0010`.

FEAT_SPEv1p2 implements the functionality identified by the value `0b0011`.

FEAT_SPEv1p3 implements the functionality identified by the value `0b0100`.

FEAT_SPEv1p4 implements the functionality identified by the value `0b0101`.

FEAT_SPEv1p5 implements the functionality identified by the value `0b0110`.

From Armv8.5, if FEAT_SPE is implemented, the value `0b0001` is not permitted.

From Armv8.7, if FEAT_SPE is implemented, the value `0b0010` is not permitted.

From Armv8.8, if FEAT_SPE is implemented, the value `0b0011` is not permitted.

From Armv8.9, if FEAT_SPE is implemented, the value `0b0100` is not permitted.

From Armv9.6, if FEAT_SPE is implemented, the value `0b0101` is not permitted.

Access to this field is RO.

CTX_CMPs, bits [31:28]

Number of context-aware breakpoints, minus 1.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CTX_CMPs	Meaning
`0b0000..0b1110`	The number of context-aware breakpoints, minus 1.
`0b1111`	If ID_AA64DFR1_EL1.CTX_CMPs is zero, then 16 context-aware breakpoints are implemented. Otherwise, 16 or more context-aware breakpoints are implemented and ID_AA64DFR1_EL1.CTX_CMPs indicates how many.

Values greater than ID_AA64DFR0_EL1.BRPs are not permitted.

Note

If AArch32 is supported at EL1, then the PE does not implement more than 16 breakpoints.

Access to this field is RO.

SEBEP, bits [27:24]

Synchronous-exception-based event profiling.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SEBEP	Meaning
`0b0000`	Synchronous-exception-based event profiling not implemented.
`0b0001`	Synchronous-exception-based event profiling implemented.

All other values are reserved.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FEAT_SEBEP implements the functionality identified by the value `0b0001`.

Access to this field is RO.

WRPs, bits [23:20]

Number of watchpoints, minus 1.

The value of this field is an IMPLEMENTATION DEFINED choice of:

WRPs	Meaning
`0b0001..0b1111`	The number of watchpoints, minus 1.

If FEAT_Debugv8p9 is implemented and 16 or more watchpoints are implemented, then this field reads as `0b1111` and ID_AA64DFR1_EL1.WRPs indicates the number of watchpoints.

Note

If AArch32 is supported at EL1, then the PE does not implement more than 16 watchpoints.

The value `0b0000` is reserved.

Access to this field is RO.

PMSS, bits [19:16]

PMU Snapshot extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PMSS	Meaning
`0b0000`	PMU snapshot extension not implemented.
`0b0001`	PMU snapshot extension implemented.

All other values are reserved.

FEAT_PMUv3_SS implements the functionality identified by the value `0b0001`.

Access to this field is RO.

BRPs, bits [15:12]

Index of the highest numbered context-aware breakpoint. Identifies which of the implemented breakpoints are context-aware breakpoints.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BRPs	Meaning
`0b0001..0b1110`	Index of the highest numbered context-aware breakpoint.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

BRPs	Meaning
0b1111	If ID_AA64DFR0_EL1.CTX_CMPs and ID_AA64DFR1_EL1.CTX_CMPs indicate that 16 or fewer breakpoints are context-aware, then the index of the highest context-aware breakpoint is 15. Otherwise, the context-aware breakpoints are the lowest numbered breakpoints.

If ID_AA64DFR1_EL1.BRPs is zero, then this is also the number of implemented breakpoints, minus 1, meaning the context-aware breakpoints are the highest numbered breakpoints.

The value of this field is always less than the number of implemented breakpoints.

Note

If AArch32 is supported at EL1, then the PE does not implement more than 16 breakpoints.

The value 0b0000 is reserved.

Access to this field is RO.

PMUVer, bits [11:8]

Performance Monitors Extension version.

This field does not follow the standard ID scheme, but uses the alternative ID scheme described in 'Alternative ID scheme used for the Performance Monitors Extension version'.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PMUVer	Meaning
0b0000	Performance Monitors Extension not implemented.
0b0001	Performance Monitors Extension, PMUv3 implemented.
0b0100	PMUv3 for Armv8.1. As 0b0001, and adds support for: • Extended 16-bit PMEVTYPER<n>_EL0.evtCount field. • If EL2 is implemented, the MDCR_EL2.HPMD control.
0b0101	PMUv3 for Armv8.4. As 0b0100, and adds support for the PMMIR_EL1 register.
0b0110	PMUv3 for Armv8.5. As 0b0101, and adds support for: • 64-bit event counters. • If EL2 is implemented, the MDCR_EL2.HCCD control. • If EL3 is implemented, the MDCR_EL3.SCCD control.
0b0111	PMUv3 for Armv8.7. As 0b0110, and adds support for: • The PMCR_EL0.FZO and, if EL2 is implemented, MDCR_EL2.HPMFZO controls. • If EL3 is implemented, the MDCR_EL3.{MPMX,MCCD} controls.
0b1000	PMUv3 for Armv8.8. As 0b0111, and: • Extends the Common event number space to include 0x0040 to 0x00BF and 0x4040 to 0x40BF. • Removes the CONSTRAINED UNPREDICTABLE behaviors if a reserved or unimplemented PMU event number is selected.
0b1001	PMUv3 for Armv8.9. As 0b1000, and: • Updates the definitions of existing PMU events. • Adds support for the PMUSERENR_EL0.UEN control and the PMUACR_EL1 register. • Adds support for the EDECR.PME control.
0b1111	IMPLEMENTATION DEFINED form of performance monitors supported, PMUv3 not supported. Arm does not recommend this value for new implementations.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

All other values are reserved.

FEAT_PMUv3 implements the functionality identified by the value `0b0001`.

FEAT_PMUv3p1 implements the functionality identified by the value `0b0100`.

FEAT_PMUv3p4 implements the functionality identified by the value `0b0101`.

FEAT_PMUv3p5 implements the functionality identified by the value `0b0110`.

FEAT_PMUv3p7 implements the functionality identified by the value `0b0111`.

FEAT_PMUv3p8 implements the functionality identified by the value `0b1000`.

FEAT_PMUv3p9 implements the functionality identified by the value `0b1001`.

From Armv8.1, if FEAT_PMUv3 is implemented, the value `0b0001` is not permitted.

From Armv8.4, if FEAT_PMUv3 is implemented, the value `0b0100` is not permitted.

From Armv8.5, if FEAT_PMUv3 is implemented, the value `0b0101` is not permitted.

From Armv8.7, if FEAT_PMUv3 is implemented, the value `0b0110` is not permitted.

From Armv8.8, if FEAT_PMUv3 is implemented, the value `0b0111` is not permitted.

From Armv8.9, if FEAT_PMUv3 is implemented, the value `0b1000` is not permitted.

Access to this field is RO.

TraceVer, bits [7:4]

Trace support. Indicates whether System register interface to a trace unit is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TraceVer	Meaning
`0b0000`	Trace unit System registers not implemented.
`0b0001`	Trace unit System registers implemented.

All other values are reserved.

When trace unit System registers are implemented, see TRCIDR1 for tracing capabilities of the trace unit.

Access to this field is RO.

DebugVer, bits [3:0]

Debug architecture version. Indicates presence of Armv8 debug architecture.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DebugVer	Meaning
`0b0110`	Armv8.0 debug architecture.
`0b0111`	Armv8.1 debug architecture, FEAT_Debugv8p1.
`0b1000`	Armv8.2 debug architecture, FEAT_Debugv8p2.
`0b1001`	Armv8.4 debug architecture, FEAT_Debugv8p4.
`0b1010`	Armv8.8 debug architecture, FEAT_Debugv8p8.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

DebugVer	Meaning
0b1011	Armv8.9 debug architecture, FEAT_Debugv8p9.

All other values are reserved.

FEAT_Debugv8p1 implements the functionality identified by the value 0b0111.

FEAT_Debugv8p2 implements the functionality identified by the value 0b1000.

FEAT_Debugv8p4 implements the functionality identified by the value 0b1001.

FEAT_Debugv8p8 implements the functionality identified by the value 0b1010.

FEAT_Debugv8p9 implements the functionality identified by the value 0b1011.

From Armv8.1, when FEAT_Debugv8p1 is implemented the value 0b0110 is not permitted.

From Armv8.2, the values 0b0110 and 0b0111 are not permitted.

From Armv8.4, the value 0b1000 is not permitted.

From Armv8.8, the value 0b1001 is not permitted.

From Armv8.9, the value 0b1010 is not permitted.

Access to this field is RO.

Accessing ID_AA64DFR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64DFR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64DFR0_EL1;
elsif PSTATE.EL == EL2 then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
         ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64DFR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64DFR0_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.81 ID_AA64DFR1_EL1, AArch64 Debug Feature Register 1

The ID_AA64DFR1_EL1 characteristics are:

Purpose

Provides top-level information about the debug system in AArch64.

Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64DFR1_EL1 are UNDEFINED.

Attributes

ID_AA64DFR1_EL1 is a 64-bit register.

Field descriptions

ABL_CMPs, bits [63:56]

When FEAT_ABLE is implemented:

Number of breakpoints that support address linking, minus 1.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ABL_CMPs	Meaning
0x00..0x3F	Number of breakpoints that support address linking minus 1.

All other values are reserved.

The number of breakpoints that support address linking is never more than either the number of breakpoints or the number of watchpoints.

Access to this field is RO.

Otherwise:

Reserved, RES0.

DPFZS, bits [55:52]

Behavior of the cycle counter when event counting is frozen by a Statistical Profiling management event.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

DPFZS	Meaning
0b0000	The cycle counter PMCCNTR_EL0 is never affected by PMCR_EL0.FZS.
0b0001	The cycle counter PMCCNTR_EL0 does not count when PMCR_EL0.DP is 1 and counting by event counters accessible to EL1 is frozen by the PMCR_EL0.FZS mechanism.

FEAT_SPE_DPFZS implements the functionality identified by the value 0b0001.

Access to this field is RO.

EBEP, bits [51:48]

Exception-based event profiling.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EBEP	Meaning
0b0000	Exception-based event profiling not implemented.
0b0001	Exception-based event profiling implemented.

All other values are reserved.

FEAT_EBEP implements the functionality identified by the value 0b0001.

Access to this field is RO.

ITE, bits [47:44]

Instrumentation Trace Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ITE	Meaning
0b0000	Instrumentation Trace Extension not implemented.
0b0001	Instrumentation Trace Extension implemented.

All other values are reserved.

FEAT_ITE implements the functionality identified by the value 0b0001.

Access to this field is RO.

ABLE, bits [43:40]

Address Breakpoint Linking Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ABLE	Meaning
0b0000	Address Breakpoint Linking Extension not implemented.
0b0001	Address Breakpoint Linking Extension implemented.

All other values are reserved.

FEAT_BWE implements the address range breakpoints and mismatch breakpoints part of the functionality identified by the value 0b0001.

FEAT_ABLE implements the functionality identified by the value 0b0001.

Access to this field is RO.

PMICNTR, bits [39:36]

PMU fixed-function instruction counter.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PMICNTR	Meaning
0b0000	PMU fixed-function instruction counter not implemented.
0b0001	PMU fixed-function instruction counter implemented.

All other values are reserved.

FEAT_PMUv3_ICNTR implements the functionality identified by the value 0b0001.

If FEAT_PMUv3 is not implemented, then the only permitted value is 0b0000.

Access to this field is RO.

SPMU, bits [35:32]

System PMU extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SPMU	Meaning
0b0000	System PMU extension not implemented.
0b0001	System PMU extension implemented.
0b0010	As 0b0001, and adds support for SPMZR_EL0.

All other values are reserved.

FEAT_SPMU implements the functionality identified by the value 0b0001.

FEAT_SPMU2 implements the functionality identified by the value 0b0010.

From Armv9.5, the value 0b0001 is not permitted.

Access to this field is RO.

CTX_CMPs, bits [31:24]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Context-aware breakpoints.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CTX_CMPs	Meaning
`0x00`	ID_AA64DFR0_EL1.CTX_CMPs is the number of context-aware breakpoints, minus 1.
`0x01..0x3F`	Number of context-aware breakpoints minus 1.

All other values are reserved.

If 16 or fewer context-aware breakpoints are implemented, then the value of this field is either `0x00` or the same as ID_AA64DFR0_EL1.CTX_CMPs. If more than 16 context-aware breakpoints are implemented, then the value `0x00` is not permitted.

Values greater than ID_AA64DFR1_EL1.BRPs are not permitted.

Access to this field is RO.

WRPs, bits [23:16]

Watchpoints.

The value of this field is an IMPLEMENTATION DEFINED choice of:

WRPs	Meaning
`0x00`	ID_AA64DFR0_EL1.WRPs is the number of watchpoints, minus 1.
`0x01..0x3F`	Number of watchpoints minus 1.

All other values are reserved.

Access to this field is RO.

BRPs, bits [15:8]

Breakpoints.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BRPs	Meaning
`0x00`	ID_AA64DFR0_EL1.BRPs is the number of breakpoints, minus 1.
`0x01..0x3F`	Number of breakpoints minus 1.

All other values are reserved.

If more than 16 breakpoints are implemented, or the index of the highest numbered context-aware breakpoint is not the number of breakpoints minus 1, then the value `0x00` is not permitted.

Nonzero values less than ID_AA64DFR0_EL1.BRPs are not permitted.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SYSPMUID, bits [7:0]

When FEAT_SPMU is implemented:

System PMU ID. Indicates the largest value that can be written to SPMSELR_EL0.SYSPMUSEL.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SYSPMUID	Meaning
0x00..0x1F	The largest supported value that can be written to SPMSELR_EL0.SYSPMUSEL.

All other values are reserved.

Since System PMUs might not be contiguously accessible, this field does not necessarily indicate the total number of accessible System PMUs.

Access to this field is RO.

Otherwise:

Reserved, RES0.

Accessing ID_AA64DFR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64DFR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64DFR1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64DFR1_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64DFR1_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.82 ID_AA64DFR2_EL1, AArch64 Debug Feature Register 2

The ID_AA64DFR2_EL1 characteristics are:

Purpose

Provides top-level information about the debug system in AArch64.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64DFR2_EL1 are UNDEFINED.

Attributes

ID_AA64DFR2_EL1 is a 64-bit register.

Field descriptions

Bits [63:28]

Reserved, RES0.

TRBE_EXC, bits [27:24]

TRBE Profiling exception extension. Describes support for reporting trace buffer management events as TRBE Profiling exceptions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TRBE_EXC	Meaning
0b0000	TRBE Profiling exception not implemented.
0b0001	TRBE Profiling exception implemented.

All other values are reserved.

FEAT_TRBE_EXC implements the functionality identified by the value 0b0001.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SPE_nVM, bits [23:20]

Profiling Buffer physical address mode supported. Describes support for defining the Profiling Buffer using physical addresses or intermediate physical addresses.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SPE_nVM	Meaning
0b0000	Profiling Buffer physical address mode not implemented.
0b0001	Profiling Buffer physical address mode implemented.

All other values are reserved.

FEAT_SPE_nVM implements the functionality identified by the value 0b0001.

Access to this field is RO.

SPE_EXC, bits [19:16]

SPE Profiling exception extension. Describes support for reporting Profiling Buffer management events as SPE Profiling exceptions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SPE_EXC	Meaning
0b0000	SPE Profiling exception not implemented.
0b0001	SPE Profiling exception implemented.

All other values are reserved.

FEAT_SPE_EXC implements the functionality identified by the value 0b0001.

Access to this field is RO.

Bits [15:8]

Reserved, RES0.

BWE, bits [7:4]

Breakpoints and watchpoint enhancements.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BWE	Meaning
0b0000	This field does not indicate whether DBGBCR<n>_EL1.MASK and address mismatch breakpoints are implemented.
0b0001	DBGBCR<n>_EL1.MASK and address mismatch breakpoints are implemented.
0b0010	As 0b0001, and address mismatch watchpoints are implemented.

All other values are reserved.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FEAT_BWE implements the functionality identified by the value `0b0001`.

FEAT_BWE2 implements the functionality identified by the value `0b0010`.

When this field is `0b0000`, ID_AA64DFR1_EL1.ABLE might indicate the presence of support for DBGBCR<n>_EL1.MASK and address mismatch breakpoints.

From Armv9.5, the value `0b0001` is not permitted.

Access to this field is RO.

STEP, bits [3:0]

Enhanced Software Step Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

STEP	Meaning
`0b0000`	Execution from MDSTEPOP_EL1 is not supported for Software Step.
`0b0001`	Execution from MDSTEPOP_EL1 is supported for Software Step.

All other values are reserved.

FEAT_STEP2 implements the functionality identified by the value `0b0001`.

Access to this field is RO.

Accessing ID_AA64DFR2_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64DFR2_EL1
```

op0	op1	CRn	CRm	op2
`0b11`	`0b000`	`0b0000`	`0b0101`	`0b010`

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
         ↪then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_AA64DFR2_EL1) || boolean
         ↪IMPLEMENTATION_DEFINED "ID_AA64DFR2_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64DFR2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64DFR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64DFR2_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.83 ID_AA64FPFR0_EL1, AArch64 Floating-point Feature Register 0

The ID_AA64FPFR0_EL1 characteristics are:

Purpose

Provides information about floating-point formats and instructions implemented in AArch64 state.

The fields in this register do not follow the standard ID scheme. See Alternative ID scheme used for ID_AA64SMFR0_EL1 and ID_AA64FPFR0_EL1.

Configuration

Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64FPFR0_EL1 are UNDEFINED.

Attributes

ID_AA64FPFR0_EL1 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

F8CVT, bit [31]

Indicates support for the following instructions:

- The Advanced SIMD floating-point scaling instruction `FSCALE`.
- The Advanced SIMD FP8 convert instructions `BF1CVTL`, `BF1CVTL2`, `BF2CVTL`, `BF2CVTL2`, `F1CVTL`, `F1CVTL2`, `F2CVTL`, `F2CVTL2`, `FCVTN`, and `FCVTN2`.
- When FEAT_SVE2 or FEAT_SME2 is implemented, the SVE2 FP8 convert instructions `BF1CVT`, `BF1CVTLT`, `BF2CVT`, `BF2CVTLT`, `F1CVT`, `F1CVTLT`, `F2CVT`, `F2CVTLT`, `BFCVTN`, `FCVTN`, `FCVTNB`, and `FCVTNT`.
- When FEAT_SME2 is implemented, the SME2 multi-vector floating-point scaling instruction `FSCALE` and the SME2 FP8 convert instructions `BF1CVT`, `BF1CVTL`, `BF2CVT`, `BF2CVTL`, `F1CVT`, `F1CVTL`, `F2CVT`, `F2CVTL`, `BFCVT`, `FCVT`, and `FCVTN`.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F8CVT	Meaning
0b0	The specified instructions are not implemented.
0b1	The specified instructions are implemented.

FEAT_FP8 implements the functionality identified by the value 1.

Access to this field is RO.

F8FMA, bit [30]

Indicates support for the following instructions:

- The Advanced SIMD FP8 to single-precision and half-precision multiply-accumulate instructions FMLALB, FMLALT, FMLALLBB, FMLALLBT, FMLALLTB, and FMLALLTT.
- When FEAT_SVE2 is implemented and the PE is not in Streaming SVE mode, the SVE2 FP8 to single-precision and half-precision multiply-accumulate instructions FMLALB, FMLALT, FMLALLBB, FMLALLBT, FMLALLTB, and FMLALLTT.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F8FMA	Meaning
0b0	The specified instructions are not implemented.
0b1	The specified instructions are implemented.

FEAT_FP8FMA implements the functionality identified by the value 1.

Access to this field is RO.

F8DP4, bit [29]

Indicates support for the following instructions:

- Advanced SIMD FP8 to single-precision 4-way dot product FDOT (4-way) instructions.
- When FEAT_SVE2 is implemented and the PE is not in Streaming SVE mode, SVE FP8 to single-precision 4-way dot product FDOT (4-way) instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F8DP4	Meaning
0b0	The specified instructions are not implemented by this feature.
0b1	The specified instructions are implemented.

Note

Other features may implement some of the specified instructions.

All other values are reserved.

FEAT_FP8DOT4 implements the functionality identified by the value 1.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F8DP2, bit [28]

Indicates support for the following instructions:

- Advanced SIMD FP8 to half-precision 2-way dot product `FDOT` (2-way) instructions.
- When FEAT_SVE2 is implemented and the PE is not in Streaming SVE mode, SVE FP8 to half-precision 2-way dot product `FDOT` (2-way) instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F8DP2	Meaning
`0b0`	The specified instructions are not implemented by this feature.
`0b1`	The specified instructions are implemented.

Note

Other features may implement some of the specified instructions.

FEAT_FP8DOT2 implements the functionality identified by the value 1.

Access to this field is RO.

F8MM8, bit [27]

Indicates support for the following instructions:

- Advanced SIMD FP8 to single-precision matrix multiply `FMMLA` (8-way, FP8 to FP32) instruction.
- If FEAT_SVE2 is implemented, SVE FP8 to single-precision matrix multiply `FMMLA` (widening, FP8 to FP32) instruction is implemented when the PE is not in Streaming SVE mode.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F8MM8	Meaning
`0b0`	Advanced SIMD and SVE FP8 to single-precision matrix multiply instructions are not implemented by this feature.
`0b1`	The specified Advanced SIMD and SVE FP8 to single-precision matrix multiply instructions are implemented.

FEAT_F8F32MM implements the functionality identified by the value `0b0001`

Access to this field is RO.

F8MM4, bit [26]

Indicates support for the following instructions:

- Advanced SIMD FP8 to half-precision matrix multiply `FMMLA` (4-way, FP8 to FP16) instruction.
- If FEAT_SVE2 is implemented, SVE FP8 to half-precision matrix multiply `FMMLA` (widening, FP8 to FP16) instruction is implemented when the PE is not in Streaming SVE mode.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F8MM4	Meaning
0b0	Advanced SIMD and SVE FP8 to half-precision matrix multiply instructions are not implemented by this feature.
0b1	The specified Advanced SIMD and SVE FP8 to half-precision matrix multiply instructions are implemented.

FEAT_F8F16MM implements the functionality identified by the value 0b0001

Access to this field is RO.

Bits [25:8]

Reserved, RES0.

Bits [7:2]

Reserved for data formats 2 to 7.

Reserved, RAZ.

F8E4M3, bit [1]

Indicates support for OFP8 E4M3 format and behavior for FP8 instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F8E4M3	Meaning
0b0	OFP8 E4M3 format is not supported.
0b1	OFP8 E4M3 format is supported.

If FEAT_FP8 is implemented, the only permitted value is 1.

Otherwise, the only permitted value is 0.

For more information on OFP8 formats, see the Open Compute Project, OCP 8-bit Floating Point Specification (OFP8).

Access to this field is RO.

F8E5M2, bit [0]

Indicates support for OFP8 E5M2 format and behavior for FP8 instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F8E5M2	Meaning
0b0	OFP8 E5M2 format is not supported.
0b1	OFP8 E5M2 format is supported.

If FEAT_FP8 is implemented, the only permitted value is 1.

Otherwise, the only permitted value is 0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

For more information on OFP8 formats, see the Open Compute Project, OCP 8-bit Floating Point Specification (OFP8).

Access to this field is RO.

Accessing ID_AA64FPFR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64FPFR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0100	0b111

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_AA64FPFR0_EL1) || boolean
     ↪IMPLEMENTATION_DEFINED "ID_AA64FPFR0_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64FPFR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64FPFR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64FPFR0_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.84 ID_AA64ISAR0_EL1, AArch64 Instruction Set Attribute Register 0

The ID_AA64ISAR0_EL1 characteristics are:

Purpose

Provides information about the instructions implemented in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64ISAR0_EL1 are UNDEFINED.

Attributes

ID_AA64ISAR0_EL1 is a 64-bit register.

Field descriptions

RNDR, bits [63:60]

Indicates support for Random Number instructions in AArch64 state.

When FEAT_RNG_TRAP is implemented, the value returned by a direct read of ID_AA64ISAR0_EL1.RNDR is further controlled by the value of SCR_EL3.TRNDR.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RNDR	Meaning
0b0000	No Random Number instructions are implemented.
0b0001	RNDR and RNDRRS registers are implemented.

All other values are reserved.

FEAT_RNG implements the functionality identified by the value 0b0001.

Access to this field is RO.

TLB, bits [59:56]

Indicates support for Outer Shareable and TLB range maintenance instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TLB	Meaning
0b0000	Outer Shareable and TLB range maintenance instructions are not implemented.
0b0001	Outer Shareable TLB maintenance instructions are implemented.
0b0010	Outer Shareable and TLB range maintenance instructions are implemented.

All other values are reserved.

FEAT_TLBIOS implements the functionality identified by the values 0b0001 and 0b0010.

FEAT_TLBIRANGE implements the functionality identified by the value 0b0010.

From Armv8.4, the values 0b0000 and 0b0001 are not permitted.

Access to this field is RO.

TS, bits [55:52]

Indicates support for flag manipulation instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TS	Meaning
0b0000	No flag manipulation instructions are implemented.
0b0001	CFINV, RMIF, SETF16, and SETF8 instructions are implemented.
0b0010	CFINV, RMIF, SETF16, SETF8, AXFLAG, and XAFLAG instructions are implemented.

All other values are reserved.

FEAT_FlagM implements the functionality identified by the value 0b0001.

FEAT_FlagM2 implements the functionality identified by the value 0b0010.

In Armv8.4, the value 0b0000 is not permitted.

From Armv8.5, the value 0b0001 is not permitted.

Access to this field is RO.

FHM, bits [51:48]

Indicates support for the Advanced SIMD half-precision to single-precision multiply instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FHM	Meaning
0b0000	The Advanced SIMD half-precision to single-precision multiply instructions are not implemented.
0b0001	FMLAL and FMLSL instructions are implemented.

All other values are reserved.

FEAT_FHM implements the functionality identified by the value 0b0001.

From Armv8.4, if FEAT_FP16 is implemented, the value 0b0000 is not permitted.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

From Armv9.7, if FEAT_AdvSIMD and FEAT_SVE_F16F32MM are both implemented, the value `0b0010` is not permitted.

Access to this field is RO.

DP, bits [47:44]

Indicates support for Dot Product instructions in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DP	Meaning
`0b0000`	No Dot Product instructions implemented.
`0b0001`	`UDOT` and `SDOT` instructions implemented.

All other values are reserved.

FEAT_DotProd implements the functionality identified by the value `0b0001`.

From Armv8.4, if FEAT_AdvSIMD is implemented, the value `0b0000` is not permitted.

Access to this field is RO.

SM4, bits [43:40]

Indicates support for SM4 instructions in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SM4	Meaning
`0b0000`	No SM4 instructions implemented.
`0b0001`	`SM4E` and `SM4EKEY` instructions implemented.

All other values are reserved.

If FEAT_SM4 is not implemented, the value `0b0001` is reserved.

FEAT_SM4 implements the functionality identified by the value `0b0001`.

This field must have the same value as ID_AA64ISAR0_EL1.SM3.

Access to this field is RO.

SM3, bits [39:36]

Indicates support for the following SM3 instructions `SM3SS1`, `SM3TT1A`, `SM3TT1B`, `SM3TT2A`, `SM3TT2B`, `SM3PARTW1`, and `SM3PARTW2` in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SM3	Meaning
`0b0000`	The specified instructions are not implemented.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SM3	Meaning
0b0001	The specified instructions are implemented.

All other values are reserved.

If FEAT_SM3 is not implemented, the value 0b0001 is reserved.

FEAT_SM3 implements the functionality identified by the value 0b0001.

This field must have the same value as ID_AA64ISAR0_EL1.SM4.

Access to this field is RO.

SHA3, bits [35:32]

Indicates support for the following SHA3 instructions EOR3, RAX1, XAR, and BCAX in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SHA3	Meaning
0b0000	The specified instructions are not implemented.
0b0001	The specified instructions are implemented.

All other values are reserved.

If FEAT_SHA3 is not implemented, the value 0b0001 is reserved.

FEAT_SHA3 implements the functionality identified by the value 0b0001.

If the value of ID_AA64ISAR0_EL1.SHA1 is 0b0000, this field must have the value 0b0000.

If the value of this field is 0b0001, ID_AA64ISAR0_EL1.SHA2 must have the value 0b0010.

Access to this field is RO.

RDM, bits [31:28]

Indicates support for SQRDMLAH and SQRDMLSH instructions in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RDM	Meaning
0b0000	No RDMA instructions implemented.
0b0001	SQRDMLAH and SQRDMLSH instructions implemented.

All other values are reserved.

FEAT_RDM implements the functionality identified by the value 0b0001.

From Armv8.1, the value 0b0000 is not permitted.

Access to this field is RO.

Bits [27:24]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Atomic, bits [23:20]

Indicates support for Atomic instructions in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Atomic	Meaning
0b0000	No Atomic instructions implemented.
0b0010	LDADD, LDCLR, LDEOR, LDSET, LDSMAX, LDSMIN, LDUMAX, LDUMIN, CAS, CASP, and SWP instructions implemented.
0b0011	As for 0b0010, plus 128-bit instructions LDCLRP, LDSETP, and SWPP.

All other values are reserved.

FEAT_LSE implements the functionality identified by the value 0b0010.

FEAT_LSE128 implements the functionality identified by the value 0b0011.

From Armv8.1, the value 0b0000 is not permitted.

Access to this field is RO.

CRC32, bits [19:16]

Indicates support for the following CRC32 instructions CRC32B, CRC32H, CRC32W, CRC32X, CRC32CB, CRC32CH, CRC32CW, and CRC32CX in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CRC32	Meaning
0b0000	The specified instructions are not implemented.
0b0001	The specified instructions are implemented.

All other values are reserved.

FEAT_CRC32 implements the functionality identified by the value 0b0001.

From Armv8.1, the value 0b0000 is not permitted.

Access to this field is RO.

SHA2, bits [15:12]

Indicates support for SHA2 instructions in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SHA2	Meaning
0b0000	No SHA2 instructions implemented.
0b0001	Implements instructions: SHA256H, SHA256H2, SHA256SU0, and SHA256SU1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SHA2	Meaning
0b0010	Implements instructions: • SHA256H, SHA256H2, SHA256SU0, and SHA256SU1. • SHA512H, SHA512H2, SHA512SU0, and SHA512SU1.

All other values are reserved.

FEAT_SHA256 implements the functionality identified by the value 0b0001.

FEAT_SHA512 implements the functionality identified by the value 0b0010.

If the value of ID_AA64ISAR0_EL1.SHA1 is 0b0000, this field must have the value 0b0000.

If the value of this field is 0b0010, ID_AA64ISAR0_EL1.SHA3 must have the value 0b0001.

Access to this field is RO.

SHA1, bits [11:8]

Indicates support for the following SHA1 instructions SHA1C, SHA1P, SHA1M, SHA1H, SHA1SU0, and SHA1SU1 in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SHA1	Meaning
0b0000	The specified instructions are not implemented.
0b0001	The specified instructions are implemented.

All other values are reserved.

FEAT_SHA1 implements the functionality identified by the value 0b0001.

If the value of ID_AA64ISAR0_EL1.SHA2 is 0b0000, this field must have the value 0b0000.

Access to this field is RO.

AES, bits [7:4]

Indicates support for AES instructions in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AES	Meaning
0b0000	No AES instructions implemented.
0b0001	AESE, AESD, AESMC, and AESIMC instructions implemented.
0b0010	As for 0b0001, plus PMULL and PMULL2 instructions operating on 64-bit source elements.

FEAT_AES implements the functionality identified by the value 0b0001.

FEAT_PMULL implements the functionality identified by the value 0b0010.

All other values are reserved.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bits [3:0]

Reserved, RES0.

Accessing ID_AA64ISAR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64ISAR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0110	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64ISAR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64ISAR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64ISAR0_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.85 ID_AA64ISAR1_EL1, AArch64 Instruction Set Attribute Register 1

The ID_AA64ISAR1_EL1 characteristics are:

Purpose

Provides information about the features and instructions implemented in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64ISAR1_EL1 are UNDEFINED.

Attributes

ID_AA64ISAR1_EL1 is a 64-bit register.

Field descriptions

63:60	59:56	55:52	51:48	47:44	43:40	39:36	35:32
LS64	XS	I8MM	DGH	BF16	SPECRES	SB	FRINTTS

31:28	27:24	23:20	19:16	15:12	11:8	7:4	3:0
GPI	GPA	LRCPC	FCMA	JSCVT	API	APA	DPB

LS64, bits [63:60]

Indicates support for `LD64B` and `ST64B*` instructions, and the ACCDATA_EL1 register.

The value of this field is an IMPLEMENTATION DEFINED choice of:

LS64	Meaning
`0b0000`	No `LD64B` or `ST64B` instructions are supported.
`0b0001`	The `LD64B` and `ST64B` instructions are supported.
`0b0010`	As `0b0001`, and adds the `ST64BV` instruction and traps.
`0b0011`	As `0b0010`, and adds the `ST64BV0` instruction, ACCDATA_EL1 register, and traps.
`0b0100`	As `0b0011`, and adds support for atomic accesses to Write-back Cacheable, Shareable memory using one of the following: • `LD64B` and `ST64B` instructions. • SIMD&FP instructions that load or store a pair of 128-bit registers by generating 32-byte single-copy atomic accesses.

All other values are reserved.

FEAT_LS64 implements the functionality identified by `0b0001`.

FEAT_LS64_V implements the functionality identified by `0b0010`.

FEAT_LS64_ACCDATA implements the functionality identified by `0b0011`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FEAT_LS64WB implements the functionality identified by `0b0100`.

Access to this field is RO.

XS, bits [59:56]

Indicates support for the XS attribute, the TLBI and DSB instructions with the nXS qualifier, and the HCRX_EL2.{FGTnXS, FnXS} fields in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

XS	Meaning
`0b0000`	The XS attribute, the TLBI and DSB instructions with the nXS qualifier, and the HCRX_EL2.{FGTnXS, FnXS} fields are not supported.
`0b0001`	The XS attribute, the TLBI and DSB instructions with the nXS qualifier, and the HCRX_EL2.{FGTnXS, FnXS} fields are supported.

All other values are reserved.

FEAT_XS implements the functionality identified by `0b0001`.

From Armv8.7, the value `0b0000` is not permitted.

Access to this field is RO.

I8MM, bits [55:52]

Indicates support for the following Advanced SIMD Int8 matrix multiplication instructions `SMMLA`, `SUDOT`, `UMMLA`, `USMMLA`, and `USDOT` in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

I8MM	Meaning
`0b0000`	The specified instructions are not implemented.
`0b0001`	The specified instructions are implemented.

All other values are reserved.

FEAT_I8MM implements the functionality identified by `0b0001`.

When Advanced SIMD and SVE are both implemented, this field must return the same value as ID_AA64ZFR0_EL1.I8MM.

From Armv8.6, the value `0b0000` is not permitted.

Access to this field is RO.

DGH, bits [51:48]

Indicates support for the Data Gathering Hint instruction.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

DGH	Meaning
0b0000	Data Gathering Hint is not implemented.
0b0001	Data Gathering Hint is implemented.

All other values are reserved.

FEAT_DGH implements the functionality identified by 0b0001.

From Armv8.0, the permitted values are 0b0000 and 0b0001.

If the DGH instruction has no effect in preventing the merging of memory accesses, the value of this field is 0b0000.

Access to this field is RO.

BF16, bits [47:44]

Indicates support for Advanced SIMD and floating-point BFloat16 instructions in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BF16	Meaning
0b0000	BFloat16 instructions are not implemented.
0b0001	BFCVT, BFCVTN, BFCVTN2, BFDOT, BFMLALB, BFMLALT, and BFMMLA instructions are implemented.
0b0010	As 0b0001, but the FPCR.EBF field is also supported.

All other values are reserved.

FEAT_BF16 implements the functionality identified by 0b0001.

FEAT_EBF16 implements the functionality identified by 0b0010.

When FEAT_SVE or FEAT_SME is implemented, this field must return the same value as ID_AA64ZFR0_EL1.BF16.

From Armv8.6 and Armv9.1, the value 0b0000 is not permitted.

Access to this field is RO.

SPECRES, bits [43:40]

Indicates support for prediction invalidation instructions in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SPECRES	Meaning
0b0000	Prediction invalidation instructions are not implemented.
0b0001	CFP RCTX, DVP RCTX and CPP RCTX instructions are implemented.
0b0010	As 0b0001, and COSP RCTX instruction is implemented.

All other values are reserved.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FEAT_SPECRES implements the functionality identified by `0b0001`.

FEAT_SPECRES2 implements the functionality identified by `0b0010`.

From Armv8.5, the value `0b0000` is not permitted.

From Armv8.9, the value `0b0001` is not permitted.

Access to this field is RO.

SB, bits [39:36]

Indicates support for `SB` instruction in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SB	Meaning
`0b0000`	`SB` instruction is not implemented.
`0b0001`	`SB` instruction is implemented.

All other values are reserved.

FEAT_SB implements the functionality identified by `0b0001`.

From Armv8.5, the value `0b0000` is not permitted.

Access to this field is RO.

FRINTTS, bits [35:32]

Indicates support for `FRINT32Z`, `FRINT32X`, `FRINT64Z`, and `FRINT64X` instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FRINTTS	Meaning
`0b0000`	The specified instructions are not implemented.
`0b0001`	The specified instructions are implemented.

All other values are reserved.

FEAT_FRINTTS implements the functionality identified by `0b0001`.

From Armv8.5, the value `0b0000` is not permitted.

Access to this field is RO.

GPI, bits [31:28]

Indicates support for an IMPLEMENTATION DEFINED algorithm is implemented in the PE for generic code authentication in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

GPI	Meaning
0b0000	Generic Authentication using an IMPLEMENTATION DEFINED algorithm is not implemented.
0b0001	Generic Authentication using an IMPLEMENTATION DEFINED algorithm is implemented. This includes the PACGA instruction.

All other values are reserved.

When this field is nonzero, FEAT_PACIMP is implemented.

From Armv8.3, the permitted values are 0b0000 and 0b0001.

If the value of ID_AA64ISAR1_EL1.GPA is nonzero, or the value of ID_AA64ISAR2_EL1.GPA3 is nonzero, this field must have the value 0b0000.

Access to this field is RO.

GPA, bits [27:24]

Indicates whether the QARMA5 algorithm is implemented in the PE for generic code authentication in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

GPA	Meaning
0b0000	Generic Authentication using the QARMA5 algorithm is not implemented.
0b0001	Generic Authentication using the QARMA5 algorithm is implemented. This includes the PACGA instruction.

All other values are reserved.

FEAT_PACQARMA5 implements the functionality identified by 0b0001.

When this field is nonzero, FEAT_PACQARMA5 is implemented.

From Armv8.3, the permitted values are 0b0000 and 0b0001.

If the value of ID_AA64ISAR1_EL1.GPI is nonzero, or the value of ID_AA64ISAR2_EL1.GPA3 is nonzero, this field must have the value 0b0000.

Access to this field is RO.

LRCPC, bits [23:20]

Indicates support for weaker release consistency, RCpc, based model.

The value of this field is an IMPLEMENTATION DEFINED choice of:

LRCPC	Meaning
0b0000	RCpc instructions are not implemented.
0b0001	The no offset LDAPR, LDAPRB, and LDAPRH instructions are implemented.
0b0010	As 0b0001, and the LDAPR (unscaled immediate) and STLR (unscaled immediate) instructions are implemented.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

LRCPC	Meaning
0b0011	As 0b0010, and the post-index LDAPR, LDIAPP, STILP, and pre-index STLR instructions are implemented. If Advanced SIMD and floating-point is implemented, then the LDAPUR (SIMD&FP), LDAP1 (SIMD&FP), STLUR (SIMD&FP), and STL1 (SIMD&FP) instructions are implemented in Advanced SIMD and floating-point.

All other values are reserved.

FEAT_LRCPC implements the functionality identified by the value 0b0001.

FEAT_LRCPC2 implements the functionality identified by the value 0b0010.

FEAT_LRCPC3 implements the functionality identified by the value 0b0011.

From Armv8.3, the value 0b0000 is not permitted.

From Armv8.4, the value 0b0001 is not permitted.

Access to this field is RO.

FCMA, bits [19:16]

Indicates support for complex number addition and multiplication, where numbers are stored in vectors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FCMA	Meaning
0b0000	The FCMLA and FCADD instructions are not implemented.
0b0001	The FCMLA and FCADD instructions are implemented.

All other values are reserved.

FEAT_FCMA implements the functionality identified by the value 0b0001.

From Armv8.3, if Advanced SIMD or floating-point is implemented, the value 0b0000 is not permitted.

From Armv8.3, if Advanced SIMD or floating-point is not implemented, the only permitted value is 0b0000.

Access to this field is RO.

JSCVT, bits [15:12]

Indicates support for JavaScript conversion from double-precision floating-point values to integers in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

JSCVT	Meaning
0b0000	The FJCVTZS instruction is not implemented.
0b0001	The FJCVTZS instruction is implemented.

All other values are reserved.

FEAT_JSCVT implements the functionality identified by 0b0001.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

From Armv8.3, if Advanced SIMD or floating-point is implemented, the value `0b0000` is not permitted.

From Armv8.3, if Advanced SIMD or floating-point is not implemented, the only permitted value is `0b0000`.

Access to this field is RO.

API, bits [11:8]

Indicates whether an IMPLEMENTATION DEFINED algorithm is implemented in the PE for address authentication, in AArch64 state. This applies to all Pointer Authentication instructions other than the `PACGA` instruction.

The value of this field is an IMPLEMENTATION DEFINED choice of:

API	Meaning
`0b0000`	Address Authentication using an IMPLEMENTATION DEFINED algorithm is not implemented.
`0b0001`	Address Authentication using an IMPLEMENTATION DEFINED algorithm is implemented, FEAT_EPAC and FEAT_PAuth2 are not implemented.
`0b0010`	Address Authentication using an IMPLEMENTATION DEFINED algorithm is implemented, FEAT_EPAC is implemented, and FEAT_PAuth2 is not implemented.
`0b0011`	Address Authentication using an IMPLEMENTATION DEFINED algorithm is implemented, FEAT_EPAC is not implemented, and FEAT_PAuth2 is implemented.
`0b0100`	Address Authentication using an IMPLEMENTATION DEFINED algorithm is implemented, FEAT_EPAC is not implemented, FEAT_PAuth2 and FEAT_FPAC are implemented.
`0b0101`	Address Authentication using an IMPLEMENTATION DEFINED algorithm is implemented, FEAT_EPAC is not implemented, FEAT_PAuth2, FEAT_FPAC, and FEAT_FPACCOMBINE are implemented.
`0b0110`	Address Authentication using an IMPLEMENTATION DEFINED algorithm is implemented, including instructions that allow signing of LR using SP and PC as diversifiers, FEAT_EPAC is not implemented, FEAT_PAuth2, FEAT_FPAC, FEAT_FPACCOMBINE, and FEAT_PAuth_LR are implemented.

All other values are reserved.

FEAT_PAuth implements the functionality identified by `0b0001`.

FEAT_EPAC implements the functionality identified by `0b0010`.

FEAT_PAuth2 implements the functionality identified by `0b0011`.

FEAT_FPAC implements the functionality identified by `0b0100`.

FEAT_FPACCOMBINE implements the functionality identified by `0b0101`.

FEAT_PAuth_LR implements the functionality identified by `0b0110`.

When this field is nonzero, FEAT_PACIMP is implemented.

In Armv8.3, the permitted values are `0b0000`, `0b0001`, `0b0010`, `0b0011`, `0b0100`, and `0b0101`.

From Armv8.6, the permitted values are `0b0000`, `0b0011`, `0b0100`, and `0b0101`.

From Armv9.5, the permitted values are `0b0000`, `0b0011`, `0b0100`, `0b0101`, and `0b0110`.

If the value of ID_AA64ISAR1_EL1.APA is nonzero, or the value of ID_AA64ISAR2_EL1.APA3 is nonzero, this field must have the value `0b0000`.

Access to this field is RO.

APA, bits [7:4]

Indicates whether the QARMA5 algorithm is implemented in the PE for address authentication, in AArch64 state. This applies to all Pointer Authentication instructions other than the `PACGA` instruction.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The value of this field is an IMPLEMENTATION DEFINED choice of:

APA	Meaning
0b0000	Address Authentication using the QARMA5 algorithm is not implemented.
0b0001	Address Authentication using the QARMA5 algorithm is implemented, FEAT_EPAC and FEAT_PAuth2 are not implemented.
0b0010	Address Authentication using the QARMA5 algorithm is implemented, FEAT_EPAC is implemented, and FEAT_PAuth2 is not implemented.
0b0011	Address Authentication using the QARMA5 algorithm is implemented, FEAT_EPAC is not implemented, and FEAT_PAuth2 is implemented.
0b0100	Address Authentication using the QARMA5 algorithm is implemented, FEAT_EPAC is not implemented, FEAT_PAuth2 and FEAT_FPAC are implemented.
0b0101	Address Authentication using the QARMA5 algorithm is implemented, FEAT_EPAC is not implemented, FEAT_PAuth2, FEAT_FPAC, and FEAT_FPACCOMBINE are implemented.
0b0110	Address Authentication using the QARMA5 algorithm is implemented, including instructions that allow signing of LR using SP and PC as diversifiers, FEAT_EPAC is not implemented, FEAT_PAuth2, FEAT_FPAC, FEAT_FPACCOMBINE, and FEAT_PAuth_LR are implemented.

All other values are reserved.

FEAT_PAuth implements the functionality identified by 0b0001.

FEAT_EPAC implements the functionality identified by 0b0010.

FEAT_PAuth2 implements the functionality identified by 0b0011.

FEAT_FPAC implements the functionality identified by 0b0100.

FEAT_FPACCOMBINE implements the functionality identified by 0b0101.

FEAT_PAuth_LR implements the functionality identified by 0b0110.

When this field is nonzero, FEAT_PACQARMA5 is implemented.

In Armv8.3, the permitted values are 0b0000, 0b0001, 0b0010, 0b0011, 0b0100, and 0b0101.

From Armv8.6, the permitted values are 0b0000, 0b0011, 0b0100, and 0b0101.

From Armv9.5, the permitted values are 0b0000, 0b0011, 0b0100, 0b0101, and 0b0110.

If the value of ID_AA64ISAR1_EL1.API is nonzero, or the value of ID_AA64ISAR2_EL1.APA3 is nonzero, this field must have the value 0b0000.

Access to this field is RO.

DPB, bits [3:0]

Data Persistence writeback. Indicates support for the DC CVAP and DC CVADP instructions in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DPB	Meaning
0b0000	DC CVAP not supported.
0b0001	DC CVAP supported.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

DPB	Meaning
0b0010	DC CVAP and DC CVADP supported.

All other values are reserved.

FEAT_DPB implements the functionality identified by the value 0b0001.

FEAT_DPB2 implements the functionality identified by the value 0b0010.

From Armv8.2, the value 0b0000 is not permitted.

From Armv8.5, the value 0b0001 is not permitted.

Access to this field is RO.

Accessing ID_AA64ISAR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64ISAR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0110	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64ISAR1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64ISAR1_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64ISAR1_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.86 ID_AA64ISAR2_EL1, AArch64 Instruction Set Attribute Register 2

The ID_AA64ISAR2_EL1 characteristics are:

Purpose

Provides information about the features and instructions implemented in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64ISAR2_EL1 are UNDEFINED.

Attributes

ID_AA64ISAR2_EL1 is a 64-bit register.

Field descriptions

63:60	59:56	55:52	51:48	47:44	43:40	39:36	35:32
ATS1A	LUT	CSSC	RPRFM	PCDPHINT	PRFMSLC	SYSINSTR_128	SYSREG_128

31:28	27:24	23:20	19:16	15:12	11:8	7:4	3:0
CLRBHB	PAC_frac	BC	MOPS	APA3	GPA3	RPRES	WFxT

ATS1A, bits [63:60]

Indicates support for address translation instructions, which perform stage 1 address translation for the given virtual address without checking for stage 1 permissions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ATS1A	Meaning
0b0000	Address Translate Stage 1 instructions without Permissions Checks are not implemented
0b0001	Address Translate Stage 1 instructions without Permissions Checks are implemented.

All other values are reserved.

Access to this field is RO.

LUT, bits [59:56]

Indicates support for:

- Advanced SIMD lookup table instructions with 2-bit and 4-bit indices.
- If FEAT_SVE2 or FEAT_SME2 is implemented, SVE lookup table instructions with 2-bit and 4-bit indices.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The value of this field is an IMPLEMENTATION DEFINED choice of:

LUT	Meaning
0b0000	The specified instructions are not implemented.
0b0001	Lookup table instructions with 2-bit indices LUTI2 and 4-bit indices LUTI4 are implemented.

All other values are reserved.

FEAT_LUT implements the functionality identified by the value 0b0001.

From Armv9.5, if FEAT_AdvSIMD is implemented, the value 0b0000 is not permitted.

Access to this field is RO.

CSSC, bits [55:52]

Indicates support for common short sequence compression instructions.

If FEAT_CMPBR is implemented, indicates support for compare and branch instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CSSC	Meaning
0b0000	Common short sequence compression instructions are not implemented.
0b0001	The following common short sequence compression instructions are implemented: • 32-bit and 64-bit ABS, CNT, CTZ • 32-bit SMAX, UMAX, SMIN, UMIN (immediate). • 64-bit SMAX, UMAX, SMIN, UMIN (immediate). • 32-bit SMAX, UMAX, SMIN, UMIN (register). • 64-bit SMAX, UMAX, SMIN, UMIN (register).
0b0010	As 0b0001, and adds compare and branch instructions: • CBB<cc>. • CB<cc> (immediate), CB<cc> (register). • CBH<cc>.

All other values are reserved.

FEAT_CSSC implements the functionality identified by the value 0b0001.

FEAT_CMPBR implements the functionality identified by the value 0b0010.

From Armv9.4, the value 0b0000 is not permitted.

From Armv9.6, the value 0b0001 is not permitted.

Access to this field is RO.

RPRFM, bits [51:48]

RPRFM hint instruction.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

RPRFM	Meaning
0b0000	RPRFM hint instruction is not implemented and is treated as a NOP.
0b0001	RPRFM hint instruction is implemented.

All other values are reserved.

FEAT_RPRFM implements the functionality identified by the value 0b0001.

Access to this field is RO.

PCDPHINT, bits [47:44]

Indicates support for producer-consumer data placement hints.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PCDPHINT	Meaning
0b0000	The STSHH and PRFM IR hint instructions are not implemented.
0b0001	The STSHH and PRFM IR hint instructions are implemented.

FEAT_PCDPHINT implements the functionality identified by the value 0b0001.

Access to this field is RO.

PRFMSLC, bits [43:40]

Indicates whether the PRFM and PRFUM instructions support a system level cache option.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PRFMSLC	Meaning
0b0000	The PRFM and PRFUM instructions do not support the SLC target.
0b0001	The PRFM and PRFUM instructions support the SLC target.

All other values are reserved.

FEAT_PRFMSLC implements the functionality identified by the value 0b0001.

Access to this field is RO.

SYSINSTR_128, bits [39:36]

SYSINSTR_128. Indicates support for System instructions that can take 128-bit inputs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SYSINSTR_128	Meaning
0b0000	System instructions that can take 128-bit inputs are not supported.
0b0001	System instructions that can take 128-bit inputs are supported.

All other values are reserved.

FEAT_SYSINSTR128 implements the functionality identified by the value 0b0001.

Access to this field is RO.

SYSREG_128, bits [35:32]

SYSREG_128. Indicates support for instructions to access 128-bit System Registers.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SYSREG_128	Meaning
0b0000	Instructions to access 128-bit System Registers are not supported.
0b0001	Instructions to access 128-bit System Registers are supported.

All other values are reserved.

FEAT_SYSREG128 implements the functionality identified by the value 0b0001.

Access to this field is RO.

CLRBHB, bits [31:28]

Indicates support for the CLRBHB instruction in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CLRBHB	Meaning
0b0000	CLRBHB instruction is not implemented.
0b0001	CLRBHB instruction is implemented.

All other values are reserved.

FEAT_CLRBHB implements the functionality identified by the value 0b0001.

From Armv8.9, the value 0b0000 is not permitted.

Access to this field is RO.

PAC_frac, bits [27:24]

Indicates which address bit is used to determine the size of the PAC field.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

PAC_frac	Meaning
0b0000	The address bit which is used to define the size of the PAC field is dependent on whether address tagging is used.
0b0001	The address bit which is used to define the size of the PAC field is fixed.

All other values are reserved.

FEAT_CONSTPACFIELD implements the functionality identified by the value 0b0001.

From Armv8.3, the permitted values are 0b0000 and 0b0001.

Access to this field is RO.

BC, bits [23:20]

Indicates support for the BC instruction in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BC	Meaning
0b0000	BC instruction is not implemented.
0b0001	BC instruction is implemented.

All other values are reserved.

FEAT_HBC implements the functionality identified by the value 0b0001.

From Armv8.8, the value 0b0000 is not permitted.

Access to this field is RO.

MOPS, bits [19:16]

Indicates support for the Memory Copy and Memory Set instructions in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MOPS	Meaning
0b0000	The Memory Copy and Memory Set instructions are not implemented in AArch64 state.
0b0001	The Memory Copy and Memory Set instructions are implemented in AArch64 state with the following exception. If FEAT_MTE is implemented, then SETGP*, SETGM* and SETGE* instructions are also supported.

All other values are reserved.

FEAT_MOPS implements the functionality identified by the value 0b0001.

From Armv8.8, the value 0b0000 is not permitted.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

APA3, bits [15:12]

Indicates whether the QARMA3 algorithm is implemented in the PE for address authentication in AArch64 state. This applies to all Pointer Authentication instructions other than the `PACGA` instruction.

The value of this field is an IMPLEMENTATION DEFINED choice of:

APA3	Meaning
`0b0000`	Address Authentication using the QARMA3 algorithm is not implemented.
`0b0001`	Address Authentication using the QARMA3 algorithm is implemented, FEAT_EPAC and FEAT_PAuth2 are not implemented.
`0b0010`	Address Authentication using the QARMA3 algorithm is implemented, FEAT_EPAC is implemented, and FEAT_PAuth2 is not implemented.
`0b0011`	Address Authentication using the QARMA3 algorithm is implemented, FEAT_EPAC is not implemented, and FEAT_PAuth2 is implemented.
`0b0100`	Address Authentication using the QARMA3 algorithm is implemented, FEAT_EPAC is not implemented, FEAT_PAuth2 and FEAT_FPAC are implemented.
`0b0101`	Address Authentication using the QARMA3 algorithm is implemented, FEAT_EPAC is not implemented, FEAT_PAuth2, FEAT_FPAC, and FEAT_FPACCOMBINE are implemented.
`0b0110`	Address Authentication using the QARMA3 algorithm is implemented, including instructions that allow signing of LR using SP and PC as diversifiers, FEAT_EPAC is not implemented, FEAT_PAuth2, FEAT_FPAC, FEAT_FPACCOMBINE, and FEAT_PAuth_LR are implemented.

All other values are reserved.

FEAT_PAuth implements the functionality identified by the value `0b0001`.

FEAT_EPAC implements the functionality identified by the value `0b0010`.

FEAT_PAuth2 implements the functionality identified by the value `0b0011`.

FEAT_FPAC implements the functionality identified by the value `0b0100`.

FEAT_FPACCOMBINE implements the functionality identified by the value `0b0101`.

FEAT_PAuth_LR implements the functionality identified by the value `0b0110`.

When this field is nonzero, FEAT_PACQARMA3 is implemented.

In Armv8.3, the permitted values are `0b0000`, `0b0001`, `0b0010`, `0b0011`, `0b0100`, and `0b0101`.

From Armv8.6, the permitted values are `0b0000`, `0b0011`, `0b0100`, and `0b0101`.

From Armv9.5, the permitted values are `0b0000`, `0b0011`, `0b0100`, `0b0101`, and `0b0110`.

If the value of ID_AA64ISAR1_EL1.API is nonzero, or the value of ID_AA64ISAR1_EL1.APA is nonzero, this field must have the value `0b0000`.

Access to this field is RO.

GPA3, bits [11:8]

Indicates whether the QARMA3 algorithm is implemented in the PE for generic code authentication in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

GPA3	Meaning
0b0000	Generic Authentication using the QARMA3 algorithm is not implemented.
0b0001	Generic Authentication using the QARMA3 algorithm is implemented. This includes the PACGA instruction.

All other values are reserved.

FEAT_PACQARMA3 implements the functionality identified by the value 0b0001.

When this field is nonzero, FEAT_PACQARMA3 is implemented.

From Armv8.3, the permitted values are 0b0000 and 0b0001.

If the value of ID_AA64ISAR1_EL1.GPI is nonzero, or the value of ID_AA64ISAR1_EL1.GPA is nonzero, this field must have the value 0b0000.

Access to this field is RO.

RPRES, bits [7:4]

Indicates support for 12 bits of mantissa in single-precision reciprocal and reciprocal square root instructions in AArch64 state, when FPCR.AH is 1.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RPRES	Meaning
0b0000	Single-precision reciprocal and reciprocal square root estimates give 8 bits of mantissa, when FPCR.AH is 1.
0b0001	Single-precision reciprocal and reciprocal square root estimates give 12 bits of mantissa, when FPCR.AH is 1.

All other values are reserved.

FEAT_RPRES implements the functionality identified by the value 0b0001.

When FPCR.AH is 0, the floating-point reciprocal estimate and reciprocal square root estimate instructions give 8 bits of mantissa.

Access to this field is RO.

WFxT, bits [3:0]

Indicates support for the WFET and WFIT instructions in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

WFxT	Meaning
0b0000	WFET and WFIT are not supported.
0b0010	WFET and WFIT are supported, and the register number is reported in the ESR_ELx on exceptions.

All other values are reserved.

FEAT_WFxT implements the functionality identified by the value 0b0010.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

From Armv8.7, the only permitted value is `0b0010`.

Access to this field is RO.

Accessing ID_AA64ISAR2_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64ISAR2_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0110	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_AA64ISAR2_EL1) || boolean
     ↪IMPLEMENTATION_DEFINED "ID_AA64ISAR2_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64ISAR2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64ISAR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64ISAR2_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.87 ID_AA64ISAR3_EL1, AArch64 Instruction Set Attribute Register 3

The ID_AA64ISAR3_EL1 characteristics are:

Purpose

Provides information about the features and instructions implemented in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64ISAR3_EL1 are UNDEFINED.

Attributes

ID_AA64ISAR3_EL1 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

FPRCVT, bits [31:28]

Indicates support for the following conversion between floating-point and integer instructions with only scalar SIMD&FP register operands and results, and with different input and output register sizes, when the PE is not in Streaming SVE mode:

- FCVTAS (scalar SIMD&FP), FCVTAU (scalar SIMD&FP).
- FCVTMS (scalar SIMD&FP), FCVTMU (scalar SIMD&FP).
- FCVTNS (scalar SIMD&FP), FCVTNU (scalar SIMD&FP).
- FCVTPS (scalar SIMD&FP), FCVTPU (scalar SIMD&FP).
- FCVTZS (scalar SIMD&FP), FCVTZU (scalar SIMD&FP).
- SCVTF (scalar SIMD&FP), UCVTF (scalar SIMD&FP).

If FEAT_SME is implemented, indicates support for the above instructions, as well as the following conversion between floating-point and integer instructions with only scalar SIMD&FP register operands and results, and with the same input and output register sizes, when the PE is in Streaming SVE mode:

- FCVTAS (vector), FCVTAU (vector).
- FCVTMS (vector), FCVTMU (vector).
- FCVTNS (vector), FCVTNU (vector).
- FCVTPS (vector), FCVTPU (vector).
- FCVTZS (vector), FCVTZU (vector).
- SCVTF (vector), UCVTF (vector).

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPRCVT	Meaning
0b0000	The specified instructions are not implemented.
0b0001	The specified instructions are implemented.

FEAT_FPRCVT implements the functionality identified by the value 0b0001.

From Armv9.6, if FEAT_FP is implemented, the value 0b0000 is not permitted.

Access to this field is RO.

LSUI, bits [27:24]

Support for the following load and store unprivileged instructions:

- LDTXR, STTXR.
- LDATXR, STLTXR.
- CAST, CASAT, CASALT, CASLT.
- CASPT, CASPAT, CASPALT, CASPLT.
- LDTP, STTP.
- LDTP (SIMD&FP), STTP (SIMD&FP).
- LDTNP, STTNP.
- LDTNP (SIMD&FP), STTNP (SIMD&FP).
- SWPT, SWPTA, SWPTL, SWPTAL.
- LDTADD, LDTADDA, LDTADDAL, LDTADDL.
- LDTSET, LDTSETA, LDTSETAL, LDTSETL.
- LDTCLR, LDTCLRA, LDTCLRAL, LDTCLRL.
- STTADD, STTADDL.
- STTSET, STTSETL.
- STTCLR, STTCLRL.

The value of this field is an IMPLEMENTATION DEFINED choice of:

LSUI	Meaning
0b0000	The specified instructions are not implemented.
0b0001	The specified instructions are implemented.

FEAT_LSUI implements the functionality identified by the value 0b0001.

Access to this field is RO.

OCCMO, bits [23:20]

Indicates support for the following cache maintenance operation to the Outer cache level instructions:

- DC CIVAOC.
- DC CVAOC.

If FEAT_MTE is implemented:

- DC CIGDVAOC.
- DC CGDVAOC.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

OCCMO	Meaning
0b0000	The specified instructions are not implemented.
0b0001	The specified instructions are implemented.

FEAT_OCCMO implements the functionality identified by the value 0b0001.

From Armv9.6, the value 0b0000 is not permitted.

Access to this field is RO.

LSFE, bits [19:16]

Indicates support for the following A64 Base atomic floating-point in-memory instructions:

- Floating-point atomic add in memory: LDFADD, LDFADDA, LDFADDAL, and LDFADDL.
- BFloat16 atomic add in memory: LDBFADD, LDBFADDA, LDBFADDAL, and LDBFADDL.
- Floating-point atomic maximum in memory: LDFMAX, LDFMAXA, LDFMAXAL, and LDFMAXL.
- BFloat16 atomic maximum in memory: LDBFMAX, LDBFMAXA, LDBFMAXAL, and LDBFMAXL.
- Floating-point atomic maximum number in memory: LDFMAXNM, LDFMAXNMA, LDFMAXNMAL, and LDFMAXNML.
- BFloat16 atomic maximum number in memory: LDBFMAXNM, LDBFMAXNMA, LDBFMAXNMAL, and LDBFMAXNML.
- Floating-point atomic minimum in memory: LDFMIN, LDFMINA, LDFMINAL, and LDFMINL.
- BFloat16 atomic minimum in memory: LDBFMIN, LDBFMINA, LDBFMINAL, and LDBFMINL.
- Floating-point atomic minimum number in memory: LDFMINNM, LDFMINNMA, LDFMINNMAL, and LDFMINNML.
- BFloat16 atomic minimum number in memory: LDBFMINNM, LDBFMINNMA, LDBFMINNMAL, and LDBFMINNML.
- Floating-point atomic add in memory, without return: STFADD and STFADDL.
- BFloat16 atomic add in memory, without return: STBFADD and STBFADDL.
- Floating-point atomic maximum in memory, without return: STFMAX and STFMAXL.
- BFloat16 atomic maximum in memory, without return: STBFMAX and STBFMAXL.
- Floating-point atomic maximum number in memory, without return: STFMAXNM and STFMAXNML.
- BFloat16 atomic maximum number in memory, without return: STBFMAXNM and STBFMAXNML.
- Floating-point atomic minimum in memory, without return: STFMIN and STFMINL.
- BFloat16 atomic minimum in memory, without return: STBFMIN and STBFMINL.
- Floating-point atomic minimum number in memory, without return: STFMINNM and STFMINNML.
- BFloat16 atomic minimum number in memory, without return: STBFMINNM and STBFMINNML.

The value of this field is an IMPLEMENTATION DEFINED choice of:

LSFE	Meaning
0b0000	Atomic floating-point in-memory instructions are not implemented.
0b0001	The specified Atomic floating-point in-memory instructions are implemented.

FEAT_LSFE implements the functionality identified by the value 0b0001

Access to this field is RO.

PACM, bits [15:12]

Indicates the implementation of PSTATE.PACM.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

PACM	Meaning
0b0000	PSTATE.PACM is not implemented.
0b0001	Trivial implementation of PSTATE.PACM.
0b0010	Full implementation of PSTATE.PACM.

All other values are reserved.

FEAT_PAuth_LR implements the functionality identified by the values 0b0001 and 0b0010.

If FEAT_PAuth_LR is implemented, the value 0b0000 is not permitted.

If ID_AA64ISAR1_EL1.API is 0b0000, the value 0b0001 is not permitted.

If one of FEAT_PACQARMA3 or FEAT_PACQARMA5 are implemented, the value 0b0001 is not permitted.

Access to this field is RO.

TLBIW, bits [11:8]

Support for TLBI VMALL for Dirty state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TLBIW	Meaning
0b0000	TLBI VMALL for Dirty state is not supported.
0b0001	TLBI VMALL for Dirty state is supported.

All other values are reserved.

FEAT_TLBIW implements the functionality identified by the value 0b0001.

Access to this field is RO.

FAMINMAX, bits [7:4]

Indicates support for the following Advanced SIMD, SVE2, and SME2 instructions that compute maximum and minimum absolute value:

- When Advanced SIMD is implemented, the Advanced SIMD instructions FAMAX and FAMIN.
- When FEAT_SVE2 or FEAT_SME2 is implemented, the SVE2 instructions FAMAX and FAMIN.
- When FEAT_SME2 is implemented, the SME2 instructions FAMAX (multiple and single vectors), FAMIN (multiple and single vectors), FAMAX (multiple vectors), and FAMIN (multiple vectors).

The value of this field is an IMPLEMENTATION DEFINED choice of:

FAMINMAX	Meaning
0b0000	The specified instructions are not implemented.
0b0001	The specified instructions are implemented.

All other values are reserved.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FEAT_FAMINMAX implements the functionality identified by the value `0b0001`.

Access to this field is RO.

CPA, bits [3:0]

Indicates support for Checked Pointer Arithmetic instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CPA	Meaning
`0b0000`	Checked Pointer Arithmetic instructions are not implemented.
`0b0001`	Checked Pointer Arithmetic instructions are implemented.
`0b0010`	Checked Pointer Arithmetic instructions are implemented, and Checked Pointer Arithmetic can be enabled.

All other values are reserved.

FEAT_CPA implements the functionality identified by the value `0b0001`.

FEAT_CPA2 implements the functionality identified by the value `0b0010`.

From Armv9.5, the value `0b0000` is not permitted.

Access to this field is RO.

Accessing ID_AA64ISAR3_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64ISAR3_EL1
```

op0	op1	CRn	CRm	op2
`0b11`	`0b000`	`0b0000`	`0b0110`	`0b011`

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
        ↪then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_AA64ISAR3_EL1) || boolean
        ↪IMPLEMENTATION_DEFINED "ID_AA64ISAR3_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64ISAR3_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64ISAR3_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64ISAR3_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.88 ID_AA64MMFR0_EL1, AArch64 Memory Model Feature Register 0

The ID_AA64MMFR0_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64MMFR0_EL1 are UNDEFINED.

Attributes

ID_AA64MMFR0_EL1 is a 64-bit register.

Field descriptions

ECV, bits [63:60]

Indicates presence of Enhanced Counter Virtualization.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ECV	Meaning
`0b0000`	Enhanced Counter Virtualization is not implemented.
`0b0001`	Enhanced Counter Virtualization is implemented. Supports CNTHCTL_EL2.{EL1TVT, EL1TVCT, EL1NVPCT, EL1NVVCT, EVNTIS}, CNTKCTL_EL1.EVNTIS, CNTPCTSS_EL0 counter views, and CNTVCTSS_EL0 counter views. Extends the PMSCR_EL1.PCT, PMSCR_EL2.PCT, TRFCR_EL1.TS, and TRFCR_EL2.TS fields.
`0b0010`	As `0b0001`, and the CNTPOFF_EL2 register and the CNTHCTL_EL2.ECV and SCR_EL3.ECVEn fields are implemented.

All other values are reserved.

FEAT_ECV implements the functionality identified by the value `0b0001`.

FEAT_ECV_POFF implements the functionality identified by the value `0b0010`.

From Armv8.6, the value `0b0000` is not permitted.

Access to this field is RO.

FGT, bits [59:56]

Indicates presence of the Fine-Grained Trap controls.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The value of this field is an IMPLEMENTATION DEFINED choice of:

FGT	Meaning
0b0000	Fine-grained trap controls are not implemented.
0b0001	Fine-grained trap controls are implemented. Supports: • If EL2 is implemented, the HAFGRTR_EL2, HDFGRTR_EL2, HDFGWTR_EL2, HFGRTR_EL2, HFGITR_EL2 and HFGWTR_EL2 registers, and their associated traps. • If EL2 is implemented, MDCR_EL2.TDCC. • If EL3 is implemented, MDCR_EL3.TDCC. • If both EL2 and EL3 are implemented, SCR_EL3.FGTEn.
0b0010	As 0b0001, and also includes support for: • If EL2 is implemented, the HDFGRTR2_EL2, HDFGWTR2_EL2, HFGITR2_EL2, HFGRTR2_EL2, and HFGWTR2_EL2 registers, and their associated traps. • If both EL2 and EL3 are implemented, SCR_EL3.FGTEn2.

All other values are reserved.

FEAT_FGT implements the functionality identified by the value 0b0001.

FEAT_FGT2 implements the functionality identified by the value 0b0010.

From Armv8.6, the value 0b0000 is not permitted.

From Armv8.9, the value 0b0001 is not permitted.

Access to this field is RO.

Bits [55:48]

Reserved, RES0.

ExS, bits [47:44]

Indicates support for disabling context synchronizing exception entry and exit.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ExS	Meaning
0b0000	All exception entries and exits are context synchronization events.
0b0001	Non-context synchronizing exception entry and exit are supported.

All other values are reserved.

FEAT_ExS implements the functionality identified by the value 0b0001.

Access to this field is RO.

TGran4_2, bits [43:40]

Indicates support for 4KB memory granule size at stage 2.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TGran4_2	Meaning	Applies when
0b0000	Support for 4KB granule at stage 2 is identified in the ID_AA64MMFR0_EL1.TGran4 field.	
0b0001	4KB granule not supported at stage 2.	
0b0010	4KB granule supported at stage 2.	
0b0011	4KB granule at stage 2 supports 52-bit input addresses and can describe 52-bit output addresses.	FEAT_LPA2 is implemented

All other values are reserved.

If EL2 is not implemented, this field is 0b0000. Otherwise, the value 0b0000 is deprecated.

Note

This field does not follow the standard ID scheme. See Alternative ID scheme used for ID_AA64MMFR0_EL1 stage 2 granule sizes for more information.

Access to this field is RO.

TGran64_2, bits [39:36]

Indicates support for 64KB memory granule size at stage 2.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TGran64_2	Meaning
0b0000	Support for 64KB granule at stage 2 is identified in the ID_AA64MMFR0_EL1.TGran64 field.
0b0001	64KB granule not supported at stage 2.
0b0010	64KB granule supported at stage 2.

All other values are reserved.

If EL2 is not implemented, this field is 0b0000. Otherwise, the value 0b0000 is deprecated.

Note

This field does not follow the standard ID scheme. See Alternative ID scheme used for ID_AA64MMFR0_EL1 stage 2 granule sizes for more information.

Access to this field is RO.

TGran16_2, bits [35:32]

Indicates support for 16KB memory granule size at stage 2.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TGran16_2	Meaning	Applies when
0b0000	Support for 16KB granule at stage 2 is identified in the ID_AA64MMFR0_EL1.TGran16 field.	
0b0001	16KB granule not supported at stage 2.	
0b0010	16KB granule supported at stage 2.	
0b0011	16KB granule at stage 2 supports 52-bit input addresses and can describe 52-bit output addresses.	FEAT_LPA2 is implemented

All other values are reserved.

If EL2 is not implemented, this field is 0b0000. Otherwise, the value 0b0000 is deprecated.

Note

This field does not follow the standard ID scheme. See Alternative ID scheme used for ID_AA64MMFR0_EL1 stage 2 granule sizes for more information.

Access to this field is RO.

TGran4, bits [31:28]

Indicates support for 4KB memory translation granule size.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TGran4	Meaning	Applies when
0b0000	4KB granule supported.	
0b0001	4KB granule supports 52-bit input addresses and can describe 52-bit output addresses.	FEAT_LPA2 is implemented
0b1111	4KB granule not supported.	

All other values are reserved.

Access to this field is RO.

TGran64, bits [27:24]

Indicates support for 64KB memory translation granule size.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TGran64	Meaning
0b0000	64KB granule supported.
0b1111	64KB granule not supported.

All other values are reserved.

Access to this field is RO.

TGran16, bits [23:20]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Indicates support for 16KB memory translation granule size.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TGran16	Meaning	Applies when
0b0000	16KB granule not supported.	
0b0001	16KB granule supported.	
0b0010	16KB granule supports 52-bit input addresses and can describe 52-bit output addresses.	FEAT_LPA2 is implemented

All other values are reserved.

Access to this field is RO.

BigEndEL0, bits [19:16]

Indicates support for mixed-endian at EL0 only.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BigEndEL0	Meaning
0b0000	No mixed-endian support at EL0. The SCTLR_EL1.E0E bit has a fixed value.
0b0001	Mixed-endian support at EL0. The SCTLR_EL1.E0E bit can be configured.

FEAT_MixedEndEL0 implements the functionality identified by the value 0b0001.

All other values are reserved.

This field is invalid and is RES0 if ID_AA64MMFR0_EL1.BigEnd is not 0b0000.

Access to this field is RO.

SNSMem, bits [15:12]

Indicates support for a distinction between Secure and Non-secure Memory.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SNSMem	Meaning
0b0000	Does not support a distinction between Secure and Non-secure Memory.
0b0001	Does support a distinction between Secure and Non-secure Memory.

Note

If EL3 is implemented, the value 0b0000 is not permitted.

All other values are reserved.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

BigEnd, bits [11:8]

Indicates support for mixed-endian configuration.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BigEnd	Meaning
0b0000	No mixed-endian support. The SCTLR_ELx.EE bits have a fixed value. See the BigEndEL0 field, bits[19:16], for whether EL0 supports mixed-endian.
0b0001	Mixed-endian support. The SCTLR_ELx.EE and SCTLR_EL1.E0E bits can be configured.

FEAT_MixedEnd implements the functionality identified by the value 0b0001.

If this field is 0b0001, FEAT_MixedEndEL0 is also implemented.

All other values are reserved.

Access to this field is RO.

ASIDBits, bits [7:4]

Number of ASID bits.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ASIDBits	Meaning
0b0000	8 bits.
0b0010	16 bits.

All other values are reserved.

Access to this field is RO.

PARange, bits [3:0]

Physical Address range supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PARange	Meaning	Applies when
0b0000	32 bits, 4GB.	
0b0001	36 bits, 64GB.	
0b0010	40 bits, 1TB.	
0b0011	42 bits, 4TB.	
0b0100	44 bits, 16TB.	
0b0101	48 bits, 256TB.	
0b0110	52 bits, 4PB.	FEAT_LPA is implemented

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

0b0111	56 bits, 64PB.	FEAT_D128 is implemented

All other values are reserved.

Access to this field is RO.

Accessing ID_AA64MMFR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64MMFR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0111	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64MMFR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64MMFR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64MMFR0_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.89 ID_AA64MMFR1_EL1, AArch64 Memory Model Feature Register 1

The ID_AA64MMFR1_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64MMFR1_EL1 are UNDEFINED.

Attributes

ID_AA64MMFR1_EL1 is a 64-bit register.

Field descriptions

ECBHB, bits [63:60]

Indicates support for restrictions on branch history speculation around exceptions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ECBHB	Meaning
0b0000	The implementation does not disclose whether restrictions are imposed on branch history speculation around exceptions.
0b0001	The implementation imposes restrictions on branch history speculation around exceptions.

All other values are reserved.

FEAT_ECBHB implements the functionality identified by the value 0b0001.

From Armv8.9, the value 0b0000 is not permitted.

Access to this field is RO.

CMOW, bits [59:56]

Indicates support for cache maintenance instruction permission.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CMOW	Meaning
0b0000	SCTLR_EL1.CMOW, SCTLR_EL2.CMOW, and HCRX_EL2.CMOW bits are not implemented.
0b0001	SCTLR_EL1.CMOW is implemented. If EL2 is implemented, SCTLR_EL2.CMOW and HCRX_EL2.CMOW bits are implemented.

All other values are reserved.

FEAT_CMOW implements the functionality identified by the value 0b0001.

From Armv8.8, the value 0b0000 is not permitted.

Access to this field is RO.

TIDCP1, bits [55:52]

Indicates whether SCTLR_EL1.TIDCP and SCTLR_EL2.TIDCP are implemented in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TIDCP1	Meaning
0b0000	SCTLR_EL1.TIDCP and SCTLR_EL2.TIDCP bits are not implemented and are RES0.
0b0001	SCTLR_EL1.TIDCP bit is implemented. If EL2 is implemented, SCTLR_EL2.TIDCP bit is implemented.

All other values are reserved.

FEAT_TIDCP1 implements the functionality identified by the value 0b0001.

From Armv8.8, the value 0b0000 is not permitted.

Access to this field is RO.

nTLBPA, bits [51:48]

Indicates support for intermediate caching of translation table walks.

The value of this field is an IMPLEMENTATION DEFINED choice of:

nTLBPA	Meaning
0b0000	The intermediate caching of translation table walks might include non-coherent physical translation caches.
0b0001	The intermediate caching of translation table walks does not include non-coherent physical translation caches.

Non-coherent physical translation caches are non-coherent caches of previous valid translation table entries since the last completed relevant TLBI applicable to the PE, where either:

- The caching is indexed by the physical address of the location holding the translation table entry.
- The caching is used for stage 1 translations and is indexed by the intermediate physical address of the location holding the translation table entry.

All other values are reserved.

FEAT_nTLBPA implements the functionality identified by the value 0b0001.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Access to this field is RO.

AFP, bits [47:44]

Indicates support for FPCR.{AH, FIZ, NEP}.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AFP	Meaning
0b0000	The FPCR.{AH, FIZ, NEP} fields are not supported.
0b0001	The FPCR.{AH, FIZ, NEP} fields are supported.

All other values are reserved.

FEAT_AFP implements the functionality identified by the value 0b0001.

From Armv8.7, if Advanced SIMD and floating-point is implemented, the value 0b0000 is not permitted.

Access to this field is RO.

HCX, bits [43:40]

Indicates support for HCRX_EL2 and its associated EL3 trap.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HCX	Meaning
0b0000	HCRX_EL2 and its associated EL3 trap are not supported.
0b0001	HCRX_EL2 and its associated EL3 trap are supported.

All other values are reserved.

FEAT_HCX implements the functionality identified by the value 0b0001.

From Armv8.7, if EL2 is implemented, the value 0b0000 is not permitted.

Access to this field is RO.

ETS, bits [39:36]

Indicates support for Enhanced Translation Synchronization.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ETS	Meaning
0b0000	Enhanced Translation Synchronization is not supported.
0b0001	Enhanced Translation Synchronization is not supported.
0b0010	FEAT_ETS2 is implemented.
0b0011	FEAT_ETS3 is implemented.

All other values are reserved.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FEAT_ETS2 implements the functionality identified by the value `0b0010`.

FEAT_ETS3 implements the functionality identified by the value `0b0011`.

From Armv8.8, the values `0b0000` and `0b0001` are not permitted.

From Armv9.5, the value `0b0010` is not permitted.

Access to this field is RO.

TWED, bits [35:32]

Indicates support for the configurable delayed trapping of WFE and WFET.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TWED	Meaning
`0b0000`	Configurable delayed trapping of WFE and WFET are not supported.
`0b0001`	Configurable delayed trapping of WFE and WFET are supported.

All other values are reserved.

FEAT_TWED implements the functionality identified by the value `0b0001`.

Access to this field is RO.

XNX, bits [31:28]

Indicates support for execute-never control distinction by Exception level at stage 2.

The value of this field is an IMPLEMENTATION DEFINED choice of:

XNX	Meaning
`0b0000`	Distinction between EL0 and EL1 execute-never control at stage 2 not supported.
`0b0001`	Distinction between EL0 and EL1 execute-never control at stage 2 supported.

All other values are reserved.

FEAT_XNX implements the functionality identified by the value `0b0001`.

From Armv8.2, the value `0b0000` is not permitted.

Access to this field is RO.

SpecSEI, bits [27:24]

When FEAT_RAS is implemented:

Describes whether the PE can generate SError exceptions from speculative reads of memory, including speculative instruction fetches.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SpecSEI	Meaning
0b0000	The PE never generates an SError exception due to an External abort on a speculative read.
0b0001	The PE might generate an SError exception due to an External abort on a speculative read.

All other values are reserved.

FEAT_SpecSEI implements the functionality identified by the value 0b0001.

Access to this field is RO.

Otherwise:

Reserved, RES0.

PAN, bits [23:20]

Privileged Access Never. Indicates support for the PAN bit in PSTATE, SPSR_EL1, SPSR_EL2, SPSR_EL3, and DSPSR_EL0.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PAN	Meaning
0b0000	PAN not supported.
0b0001	PAN supported.
0b0010	PAN supported and AT S1E1RP and AT S1E1WP instructions supported.
0b0011	PAN supported, AT S1E1RP and AT S1E1WP instructions supported, and SCTLR_EL1.EPAN and SCTLR_EL2.EPAN bits supported.

All other values are reserved.

FEAT_PAN implements the functionality identified by the value 0b0001.

FEAT_PAN2 implements the functionality added by the value 0b0010.

FEAT_PAN3 implements the functionality added by the value 0b0011.

From Armv8.1, the value 0b0000 is not permitted.

From Armv8.2, the value 0b0001 is not permitted.

From Armv8.7, the value 0b0010 is not permitted.

Access to this field is RO.

LO, bits [19:16]

LORegions. Indicates support for LORegions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

LO	Meaning
0b0000	LORegions not supported.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

LO	Meaning
0b0001	LORegions supported.

All other values are reserved.

FEAT_LOR implements the functionality identified by the value 0b0001.

From Armv8.1, the value 0b0000 is not permitted.

Access to this field is RO.

HPDS, bits [15:12]

Hierarchical Permission Disables. Indicates support for disabling hierarchical controls in translation tables.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HPDS	Meaning
0b0000	Disabling of hierarchical controls not supported.
0b0001	Disabling of hierarchical controls supported with the TCR_EL1.{HPD1, HPD0}, TCR_EL2.HPD or TCR_EL2.{HPD1, HPD0}, and TCR_EL3.HPD bits.
0b0010	As for value 0b0001, and adds possible hardware allocation of bits[62:59] of the Translation table descriptors from the final lookup level for IMPLEMENTATION DEFINED use.

All other values are reserved.

FEAT_HPDS implements the functionality identified by the value 0b0001.

FEAT_HPDS2 implements the functionality identified by the value 0b0010.

From Armv8.1, the value 0b0000 is not permitted.

Access to this field is RO.

VH, bits [11:8]

Virtualization Host Extensions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

VH	Meaning
0b0000	Virtualization Host Extensions not supported.
0b0001	Virtualization Host Extensions supported.

All other values are reserved.

FEAT_VHE implements the functionality identified by the value 0b0001.

From Armv8.1, the value 0b0000 is not permitted.

Access to this field is RO.

VMIDBits, bits [7:4]

Number of VMID bits.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The value of this field is an IMPLEMENTATION DEFINED choice of:

VMIDBits	Meaning
0b0000	8 bits
0b0010	16 bits

All other values are reserved.

FEAT_VMID16 implements the functionality identified by the value 0b0010.

Access to this field is RO.

HAFDBS, bits [3:0]

Hardware updates to Access flag and Dirty state in translation tables.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAFDBS	Meaning
0b0000	Hardware update of the Access flag and dirty state are not supported.
0b0001	Support for hardware update of the Access flag for Block and Page descriptors.
0b0010	As 0b0001, and adds support for hardware update of dirty state.
0b0011	As 0b0010, and adds support for hardware update of the Access flag for Table descriptors.
0b0100	As 0b0011, and adds support for hardware tracking of Dirty state Structure.

All other values are reserved.

FEAT_HAFDBS implements the functionality identified by the values 0b0001 and 0b0010.

FEAT_HAFT implements the functionality identified by the value 0b0011.

FEAT_HDBSS implements the functionality identified by the value 0b0100.

Access to this field is RO.

Accessing ID_AA64MMFR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64MMFR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0111	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
        ↪then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64MMFR1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
        ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64MMFR1_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64MMFR1_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.90 ID_AA64MMFR2_EL1, AArch64 Memory Model Feature Register 2

The ID_AA64MMFR2_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64MMFR2_EL1 are UNDEFINED.

Attributes

ID_AA64MMFR2_EL1 is a 64-bit register.

Field descriptions

63:60	59:56	55:52	51:48	47:44	43:40	39:36	35:32
E0PD	EVT	BBM	TTL	RES0	FWB	IDS	AT

31:28	27:24	23:20	19:16	15:12	11:8	7:4	3:0
ST	NV	CCIDX	VARange	IESB	LSM	UAO	CnP

E0PD, bits [63:60]

Indicates support for the E0PD mechanism.

The value of this field is an IMPLEMENTATION DEFINED choice of:

E0PD	Meaning
0b0000	E0PDx mechanism is not implemented.
0b0001	E0PDx mechanism is implemented.

All other values are reserved.

FEAT_E0PD implements the functionality identified by the value 0b0001.

In Armv8.4, the permitted values are 0b0000 and 0b0001.

From Armv8.5, the only permitted value is 0b0001.

If FEAT_E0PD is implemented, FEAT_CSV3 must be implemented.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EVT, bits [59:56]

Enhanced Virtualization Traps. If EL2 is implemented, indicates support for the HCR_EL2.{TTLBOS, TTLBIS, TOCU, TICAB, TID4} traps.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EVT	Meaning
0b0000	HCR_EL2.{TTLBOS, TTLBIS, TOCU, TICAB, TID4} traps are not supported.
0b0001	HCR_EL2.{TOCU, TICAB, TID4} traps are supported. HCR_EL2.{TTLBOS, TTLBIS} traps are not supported.
0b0010	HCR_EL2.{TTLBOS, TTLBIS, TOCU, TICAB, TID4} traps are supported.

All other values are reserved.

FEAT_EVT implements the functionality identified by the values 0b0001 and 0b0010.

If EL2 is not implemented, the only permitted value is 0b0000.

From Armv8.5, if EL2 is implemented, the value 0b0001 is not permitted.

Access to this field is RO.

BBM, bits [55:52]

Allows identification of the requirements of the hardware to have break-before-make sequences when changing block or table size for a translation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BBM	Meaning
0b0000	Break-before-make sequence must be used.
0b0001	Level 1 support for changing block size is supported.
0b0010	Level 2 support for changing block size is supported.

All other values are reserved.

FEAT_BBML1 implements the functionality identified by the value 0b0001.

FEAT_BBML2 implements the functionality identified by the value 0b0010.

Access to this field is RO.

TTL, bits [51:48]

Indicates support for TTL field in address operations.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TTL	Meaning
0b0000	TLB maintenance instructions by address have bits[47:44] as RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TTL	Meaning
0b0001	TLB maintenance instructions by address have bits[47:44] holding the TTL field.

All other values are reserved.

FEAT_TTL implements the functionality identified by the value 0b0001.

This field affects TLBI IPAS2E1, TLBI IPAS2E1IS, TLBI IPAS2E1OS, TLBI IPAS2LE1, TLBI IPAS2LE1IS, TLBI IPAS2LE1OS, TLBI VAAE1, TLBI VAAE1IS, TLBI VAAE1OS, TLBI VAALE1, TLBI VAALE1IS, TLBI VAALE1OS, TLBI VAE1, TLBI VAE1IS, TLBI VAE1OS, TLBI VAE2, TLBI VAE2IS, TLBI VAE2OS, TLBI VAE3, TLBI VAE3IS, TLBI VAE3OS,TLBI VALE1, TLBI VALE1IS, TLBI VALE1OS, TLBI VALE2, TLBI VALE2IS, TLBI VALE2OS, TLBI VALE3, TLBI VALE3IS, TLBI VALE3OS.

From Armv8.4, the value 0b0000 is not permitted.

Access to this field is RO.

Bits [47:44]

Reserved, RES0.

FWB, bits [43:40]

Indicates support for HCR_EL2.FWB.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FWB	Meaning
0b0000	HCR_EL2.FWB bit is not supported.
0b0001	HCR_EL2.FWB is supported.

All other values reserved.

FEAT_S2FWB implements the functionality identified by the value 0b0001.

From Armv8.4, the value 0b0000 is not permitted.

Access to this field is RO.

IDS, bits [39:36]

Indicates the EC syndrome value in ESR_ELx.EC that is reported if an exception is generated by a read access to the feature ID space.

The value of this field is an IMPLEMENTATION DEFINED choice of:

IDS	Meaning
0b0000	An exception which is generated by a read access to the feature ID space, other than a trap caused by HCR_EL2.TIDx, SCTLR_EL1.UCT, or SCTLR_EL2.UCT is reported by ESR_ELx.EC == 0x0.
0b0001	All exceptions generated by an AArch64 read access to the feature ID space are reported by ESR_ELx.EC == 0x18.
0b0010	As 0b0001 and introduces support for trapping ID register accesses to EL3.

All other values are reserved.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The Feature ID space is defined as the System register space in AArch64 with op0==3, op1=={0, 1, 3}, CRn==0, CRm=={0-7}, op2=={0-7}.

FEAT_IDST implements the functionality identified by the value `0b0001`.

FEAT_IDTE3 implements the functionality identified by `0b0010`.

From Armv8.4, the value `0b0000` is not permitted.

From Armv9.6, if EL3 is implemented, the value `0b0001` is not permitted.

Access to this field is RO.

AT, bits [35:32]

Identifies support for unaligned single-copy atomicity and atomic functions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AT	Meaning
`0b0000`	Unaligned single-copy atomicity and atomic functions are not supported.
`0b0001`	Unaligned single-copy atomicity and atomic functions with a 16-byte address range aligned to 16-bytes are supported.

All other values are reserved.

FEAT_LSE2 implements the functionality identified by the value `0b0001`.

From Armv8.4, the value `0b0000` is not permitted.

Access to this field is RO.

ST, bits [31:28]

Identifies support for small translation tables.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ST	Meaning
`0b0000`	The maximum value of the TCR_ELx.{T0SZ,T1SZ} and VTCR_EL2.T0SZ fields is 39.
`0b0001`	The maximum value of the TCR_ELx.{T0SZ,T1SZ} and VTCR_EL2.T0SZ fields is 48 for 4KB and 16KB granules, and 47 for 64KB granules.

All other values are reserved.

FEAT_TTST implements the functionality identified by the value `0b0001`.

When FEAT_SEL2 is implemented, the value `0b0000` is not permitted.

Access to this field is RO.

NV, bits [27:24]

Nested Virtualization. If EL2 is implemented, indicates support for the use of nested virtualization.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

NV	Meaning
0b0000	If ID_AA64MMFR4_EL1.NV_frac != 0b0000, support for nested virtualization is described in ID_AA64MMFR4_EL1.NV_frac. Otherwise, nested virtualization is not supported.
0b0001	The HCR_EL2.{AT, NV1, NV} bits are implemented.
0b0010	The VNCR_EL2 register and the HCR_EL2.{NV2, AT, NV1, NV} bits are implemented.

All other values are reserved.

If EL2 is not implemented, the only permitted value is 0b0000.

FEAT_NV implements the functionality identified by the value 0b0001.

FEAT_NV2 implements the functionality identified by the value 0b0010.

In Armv8.3, if EL2 is implemented, the permitted values are 0b0000 and 0b0001.

From Armv8.4, if EL2 is implemented, the permitted values are 0b0000, 0b0001, and 0b0010.

Access to this field is RO.

CCIDX, bits [23:20]

Support for the use of revised CCSIDR_EL1 register format.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CCIDX	Meaning
0b0000	32-bit format implemented for all levels of the CCSIDR_EL1.
0b0001	64-bit format implemented for all levels of the CCSIDR_EL1.

All other values are reserved.

FEAT_CCIDX implements the functionality identified by the value 0b0001.

From Armv8.3, the permitted values are 0b0000 and 0b0001.

Access to this field is RO.

VARange, bits [19:16]

Indicates support for a larger virtual address.

The value of this field is an IMPLEMENTATION DEFINED choice of:

VARange	Meaning	Applies when
0b0000	VMSAv8-64 supports 48-bit VAs.	
0b0001	VMSAv8-64 supports 52-bit VAs when using the 64KB translation granule. The size for other translation granules is not defined by this field.	
0b0010	VMSAv9-128 supports 56-bit VAs.	FEAT_D128 is implemented

All other values are reserved.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FEAT_LVA implements the functionality identified by the value `0b0001`.

FEAT_LVA3 implements the functionality identified by the value `0b0010`.

Access to this field is RO.

IESB, bits [15:12]

Indicates support for the IESB bit in the SCTLR_ELx registers.

The value of this field is an IMPLEMENTATION DEFINED choice of:

IESB	Meaning
`0b0000`	IESB bit in the SCTLR_ELx registers is not supported.
`0b0001`	IESB bit in the SCTLR_ELx registers is supported.

All other values are reserved.

FEAT_IESB implements the functionality identified by the value `0b0001`.

Access to this field is RO.

LSM, bits [11:8]

Indicates support for LSMAOE and nTLSMD bits in SCTLR_EL1 and SCTLR_EL2.

The value of this field is an IMPLEMENTATION DEFINED choice of:

LSM	Meaning
`0b0000`	LSMAOE and nTLSMD bits not supported.
`0b0001`	LSMAOE and nTLSMD bits supported.

All other values are reserved.

FEAT_LSMAOC implements the functionality identified by the value `0b0001`.

Access to this field is RO.

UAO, bits [7:4]

User Access Override.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UAO	Meaning
`0b0000`	UAO not supported.
`0b0001`	UAO supported.

All other values are reserved.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FEAT_UAO implements the functionality identified by the value `0b0001`.

From Armv8.2, the value `0b0000` is not permitted.

Access to this field is RO.

CnP, bits [3:0]

Indicates support for Common not Private translations.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CnP	Meaning
`0b0000`	Common not Private translations not supported.
`0b0001`	Common not Private translations supported.

All other values are reserved.

FEAT_TTCNP implements the functionality identified by the value `0b0001`.

From Armv8.2, the value `0b0000` is not permitted.

Access to this field is RO.

Accessing ID_AA64MMFR2_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64MMFR2_EL1
```

op0	op1	CRn	CRm	op2
`0b11`	`0b000`	`0b0000`	`0b0111`	`0b010`

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
        ↪then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_AA64MMFR2_EL1) || boolean
        ↪IMPLEMENTATION_DEFINED "ID_AA64MMFR2_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        X[t, 64] = ID_AA64MMFR2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64MMFR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64MMFR2_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.91 ID_AA64MMFR3_EL1, AArch64 Memory Model Feature Register 3

The ID_AA64MMFR3_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch64 state.

Configuration

Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64MMFR3_EL1 are UNDEFINED.

Attributes

ID_AA64MMFR3_EL1 is a 64-bit register.

Field descriptions

Spec_FPACC, bits [63:60]

When FEAT_FPACCOMBINE is implemented:

Speculative behavior in the event of a PAC authentication failure in an implementation that includes FEAT_FPACCOMBINE.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Spec_FPACC	Meaning
0b0000	The implementation does not disclose whether the speculative use of pointers processed by a PAC Authentication is materially different in terms of the impact on cached microarchitectural state between passing and failing of the PAC Authentication.
0b0001	The speculative use of pointers processed by a PAC Authentication is not materially different in terms of the impact on cached microarchitectural state between passing and failing of the PAC Authentication.

All other values are reserved.

For the purpose of this definition, cached microarchitecture state is the state of caching agents such as instruction caches, data caches and TLBs which can be altered as a result of speculation caused by a mispredicted execution, but is not restored to the state prior to the speculation when the misprediction is corrected.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

ADERR, bits [59:56]

Asynchronous Device error exceptions. With ID_AA64MMFR3_EL1.SDERR, describes the PE behavior for External aborts signaled on Device memory loads.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ADERR	Meaning
0b0000	If FEAT_RASv2 is not implemented and ID_AA64MMFR3_EL1.SDERR is 0b0000, then the behavior is not described. Otherwise, the behavior is described by ID_AA64MMFR3_EL1.SDERR.
0b0001	All External aborts on Device memory loads are handled asynchronously.
0b0010	FEAT_ADERR is implemented. SCTLR2_ELx.EnADERR and HCRX_EL2.EnSDERR are implemented. If FEAT_ANERR is also implemented, then software should ensure that all the following apply: • SCTLR2_ELx.EnADERR is set to the value of SCTLR2_ELx.EnANERR. • HCRX_EL2.EnSDERR is set to the value of HCRX_EL2.EnSNERR.
0b0011	FEAT_ADERR is implemented. SCTLR2_ELx.EnADERR and HCRX_EL2.EnSDERR are implemented. If FEAT_ANERR is also implemented, then SCTLR2_ELx.EnADERR and HCRX_EL2.EnSDERR operate independently of SCTLR2_ELx.EnANERR and HCRX_EL2.EnSNERR.

All other values are reserved.

Implementation-specific exceptions to applications of this field are described in ‘Taking error exceptions’.

When FEAT_RASv2 is implemented and ID_AA64MMFR3_EL1.SDERR is 0b0000, the value of this field is 0b0001.

When ID_AA64MMFR3_EL1.SDERR is 0b0001, the value of this field is 0b0000.

When ID_AA64MMFR3_EL1.SDERR is 0b0010, the value of this field is 0b0010.

When ID_AA64MMFR3_EL1.SDERR is 0b0011, the value of this field is 0b0011.

FEAT_ADERR implements the functionality described by the value 0b0010.

Access to this field is RO.

SDERR, bits [55:52]

Synchronous Device error exceptions. With ID_AA64MMFR3_EL1.ADERR, describes the PE behavior for External aborts signaled on Device memory loads.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SDERR	Meaning
0b0000	If FEAT_RASv2 is not implemented and ID_AA64MMFR3_EL1.ADERR is 0b0000, then the behavior is not described. Otherwise, the behavior is described by ID_AA64MMFR3_EL1.ADERR.
0b0001	All External aborts on Device memory loads are handled synchronously.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SDERR	Meaning
0b0010	FEAT_ADERR is implemented. SCTLR2_ELx.EnADERR and HCRX_EL2.EnSDERR are implemented. If FEAT_ANERR is also implemented, then software should ensure that all the following apply: • SCTLR2_ELx.EnADERR is set to the value of SCTLR2_ELx.EnANERR. • HCRX_EL2.EnSDERR is set to the value of HCRX_EL2.EnSNERR.
0b0011	FEAT_ADERR is implemented. SCTLR2_ELx.EnADERR and HCRX_EL2.EnSDERR are implemented. If FEAT_ANERR is also implemented, then SCTLR2_ELx.EnADERR and HCRX_EL2.EnSDERR operate independently of SCTLR2_ELx.EnANERR and HCRX_EL2.EnSNERR.

All other values are reserved.

Implementation-specific exceptions to applications of this field are described in 'Taking error exceptions'.

When FEAT_RASv2 is implemented and ID_AA64MMFR3_EL1.ADERR is 0b0000, the value of this field is 0b0001.

When ID_AA64MMFR3_EL1.ADERR is 0b0001, the value of this field is 0b0000.

When ID_AA64MMFR3_EL1.ADERR is 0b0010, the value of this field is 0b0010.

When ID_AA64MMFR3_EL1.ADERR is 0b0011, the value of this field is 0b0011.

FEAT_ADERR implements the functionality described by the value 0b0010.

Access to this field is RO.

Bits [51:48]

Reserved, RES0.

ANERR, bits [47:44]

Asynchronous Normal error exceptions. With ID_AA64MMFR3_EL1.SNERR, describes the PE behavior for External aborts signaled on Normal memory loads.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ANERR	Meaning
0b0000	If FEAT_RASv2 is not implemented and ID_AA64MMFR3_EL1.SNERR is 0b0000, then the behavior is not described. Otherwise, the behavior is described by ID_AA64MMFR3_EL1.SNERR.
0b0001	All External aborts on Normal memory loads are handled asynchronously.
0b0010	FEAT_ANERR is implemented. SCTLR2_ELx.EnANERR and HCRX_EL2.EnSNERR are implemented. If FEAT_ADERR is also implemented, then software should ensure that all the following apply: • SCTLR2_ELx.EnANERR is set to the value of SCTLR2_ELx.EnADERR. • HCRX_EL2.EnSNERR is set to the value of HCRX_EL2.EnSDERR.
0b0011	FEAT_ANERR is implemented. SCTLR2_ELx.EnANERR and HCRX_EL2.EnSNERR are implemented. If FEAT_ADERR is also implemented, then SCTLR2_ELx.EnANERR and HCRX_EL2.EnSNERR operate independently of SCTLR2_ELx.EnADERR and HCRX_EL2.EnSDERR.

All other values are reserved.

Implementation-specific exceptions to applications of this field are described in 'Taking error exceptions'.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_RASv2 is implemented and ID_AA64MMFR3_EL1.SNERR is `0b0000`, the value of this field is `0b0001`.

When ID_AA64MMFR3_EL1.SNERR is `0b0001`, the value of this field is `0b0000`.

When ID_AA64MMFR3_EL1.SNERR is `0b0010`, the value of this field is `0b0010`.

When ID_AA64MMFR3_EL1.SNERR is `0b0011`, the value of this field is `0b0011`.

FEAT_ANERR implements the functionality described by the value `0b0010`.

Access to this field is RO.

SNERR, bits [43:40]

Synchronous Normal error exceptions. With ID_AA64MMFR3_EL1.ANERR, describes the PE behavior for External aborts signaled on Normal memory loads.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SNERR	Meaning
`0b0000`	If FEAT_RASv2 is not implemented and ID_AA64MMFR3_EL1.ANERR is `0b0000`, then the behavior is not described. Otherwise, the behavior is described by ID_AA64MMFR3_EL1.ANERR.
`0b0001`	All External aborts on Normal memory loads are handled synchronously.
`0b0010`	FEAT_ANERR is implemented. SCTLR2_ELx.EnANERR and HCRX_EL2.EnSNERR are implemented. If FEAT_ADERR is also implemented, then software should ensure that all the following apply: • SCTLR2_ELx.EnANERR is set to the value of SCTLR2_ELx.EnADERR. • HCRX_EL2.EnSNERR is set to the value of HCRX_EL2.EnSDERR.
`0b0011`	FEAT_ANERR is implemented. SCTLR2_ELx.EnANERR and HCRX_EL2.EnSNERR are implemented. If FEAT_ADERR is also implemented, then SCTLR2_ELx.EnANERR and HCRX_EL2.EnSNERR operate independently of SCTLR2_ELx.EnADERR and HCRX_EL2.EnSDERR.

All other values are reserved.

Implementation-specific exceptions to applications of this field are described in ‘Taking error exceptions’.

When FEAT_RASv2 is implemented and ID_AA64MMFR3_EL1.ANERR is `0b0000`, the value of this field is `0b0001`.

When ID_AA64MMFR3_EL1.ANERR is `0b0001`, the value of this field is `0b0000`.

When ID_AA64MMFR3_EL1.ANERR is `0b0010`, the value of this field is `0b0010`.

When ID_AA64MMFR3_EL1.ANERR is `0b0011`, the value of this field is `0b0011`.

FEAT_ANERR implements the functionality described by the value `0b0010`.

Access to this field is RO.

D128_2, bits [39:36]

128-bit translation table descriptor at stage 2. Indicates support for 128-bit translation table descriptor at stage 2.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D128_2	Meaning
0b0000	128-bit translation table descriptor Extension at stage 2 is not supported.
0b0001	128-bit translation table descriptor Extension at stage 2 is supported.

All other values are reserved.

Access to this field is RO.

D128, bits [35:32]

128-bit translation table descriptor. Indicates support for 128-bit translation table descriptor.

The value of this field is an IMPLEMENTATION DEFINED choice of:

D128	Meaning
0b0000	128-bit translation table descriptor Extension is not supported.
0b0001	128-bit translation table descriptor Extension is supported.

All other values are reserved.

Access to this field is RO.

MEC, bits [31:28]

Indicates support for Memory Encryption Contexts.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MEC	Meaning
0b0000	Memory Encryption Contexts is not supported.
0b0001	Memory Encryption Contexts is supported, with multiple contexts in the Realm physical address space.

All other values are reserved.

FEAT_MEC implements the functionality identified by the value 0b0001.

Access to this field is RO.

AIE, bits [27:24]

Attribute Indexing. Indicates support for the Attribute Index Enhancement.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AIE	Meaning
0b0000	The Attribute Index Enhancement is not supported.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

0b0001	The Attribute Index Enhancement at stage 1 is supported.

All other values are reserved.

FEAT_AIE implements the functionality identified by the value 0b0001.

Access to this field is RO.

S2POE, bits [23:20]

Stage 2 Permission Overlay. Indicates support for Permission Overlay at stage 2.

The value of this field is an IMPLEMENTATION DEFINED choice of:

S2POE	Meaning
0b0000	Permission Overlay at stage 2 is not supported.
0b0001	Permission Overlay at stage 2 is supported.

All other values are reserved.

FEAT_S2POE implements the functionality identified by the value 0b0001.

Access to this field is RO.

S1POE, bits [19:16]

Stage 1 Permission Overlay. Indicates support for Permission Overlay at stage 1.

The value of this field is an IMPLEMENTATION DEFINED choice of:

S1POE	Meaning
0b0000	Permission Overlay at stage 1 is not supported.
0b0001	Permission Overlay at stage 1 is supported.

All other values are reserved.

FEAT_S1POE implements the functionality identified by the value 0b0001.

Access to this field is RO.

S2PIE, bits [15:12]

Stage 2 Permission Indirection. Indicates support for Permission Indirection at stage 2.

The value of this field is an IMPLEMENTATION DEFINED choice of:

S2PIE	Meaning
0b0000	Permission Indirection at stage 2 is not supported.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

0b0001	Permission Indirection at stage 2 is supported.

All other values are reserved.

FEAT_S2PIE implements the functionality identified by the value 0b0001.

Access to this field is RO.

S1PIE, bits [11:8]

Stage 1 Permission Indirection. Indicates support for Permission Indirection at stage 1.

The value of this field is an IMPLEMENTATION DEFINED choice of:

S1PIE	Meaning
0b0000	Permission Indirection at stage 1 is not supported.
0b0001	Permission Indirection at stage 1 is supported.

All other values are reserved.

FEAT_S1PIE implements the functionality identified by the value 0b0001.

Access to this field is RO.

SCTLRX, bits [7:4]

SCTLR Extension. Indicates support for extension of SCTLR_ELx.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SCTLRX	Meaning
0b0000	SCTLR2_EL1, SCTLR2_EL2, SCTLR2_EL3 registers, and their associated trap controls are not implemented.
0b0001	SCTLR2_EL1, SCTLR2_EL2, SCTLR2_EL3 resisters, and their associated trap controls are implemented.

All other values are reserved.

From Armv8.9, the value 0b0000 is not permitted.

FEAT_SCTLR2 implements the functionality described by the value 0b0001.

Access to this field is RO.

TCRX, bits [3:0]

TCR Extension. Indicates support for extension of TCR_ELx.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TCRX	Meaning
0b0000	TCR2_EL1, TCR2_EL2, and their associated trap controls are not implemented.
0b0001	TCR2_EL1, TCR2_EL2, and their associated trap controls are implemented.

All other values are reserved.

From Armv8.9, the value 0b0000 is not permitted.

FEAT_TCR2 implements the functionality described by the value 0b0001.

Access to this field is RO.

Accessing ID_AA64MMFR3_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64MMFR3_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0111	0b011

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_AA64MMFR3_EL1) || boolean
     ↪IMPLEMENTATION_DEFINED "ID_AA64MMFR3_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64MMFR3_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64MMFR3_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64MMFR3_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.92 ID_AA64MMFR4_EL1, AArch64 Memory Model Feature Register 4

The ID_AA64MMFR4_EL1 characteristics are:

Purpose

Provides additional information about implemented memory model and memory management support in AArch64.

Configuration

Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64MMFR4_EL1 are UNDEFINED.

Attributes

ID_AA64MMFR4_EL1 is a 64-bit register.

Field descriptions

Bits [63:48]

Reserved, RES0.

SRMASK, bits [47:44]

Indicates support for bitwise write masks for the following registers:

- ACTLR_EL1.
- CPACR_EL1.
- SCTLR_EL1.
- SCTLR2_EL1.
- TCR_EL1.
- TCR2_EL1.
- ACTLR_EL2.
- CPTR_EL2.
- SCTLR_EL2.
- SCTLR2_EL2.
- TCR_EL2.
- TCR2_EL2.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SRMASK	Meaning
0b0000	Bitwise write masks for the specified registers are not supported.
0b0001	Bitwise write masks for the specified registers are supported.

FEAT_SRMASK implements the functionality identified by the value 0b0001.

From Armv9.6, the value 0b0000 is not permitted.

Access to this field is RO.

Bits [43:40]

Reserved, RES0.

E3DSE, bits [39:36]

Delegated SError exceptions from EL3. Describes support for delegated SError injection from EL3.

The value of this field is an IMPLEMENTATION DEFINED choice of:

E3DSE	Meaning
0b0000	FEAT_E3DSE is not implemented.
0b0001	FEAT_E3DSE is implemented. The following are implemented: • Register fields SCR_EL3.DSE and SCR_EL3.EnDSE. • Registers VSESR_EL3 and VDISR_EL3.

All other values are reserved.

FEAT_E3DSE implements the functionality described by the value 0b0001.

Access to this field is RO.

Bits [35:32]

Reserved, RES0.

RMEGDI, bits [31:28]

RME Granular Data Isolation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RMEGDI	Meaning
0b0000	The Granular Data isolation GPI encodings are reserved.
0b0001	The Granular Data Isolation GPI encodings can be configured to be either reserved or No Access from this PE.

All other values are reserved.

FEAT_RME_GDI implements the functionality described by the value 0b0001.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E2H0, bits [27:24]

Indicates support for programming HCR_EL2.E2H.

The value of this field is an IMPLEMENTATION DEFINED choice of:

E2H0	Meaning
0b0000	FEAT_E2H0 is implemented.
0b1110	FEAT_E2H0 is not implemented. HCR_EL2.NV1 is RES0.
0b1111	FEAT_E2H0 is not implemented.

All other values are reserved.

If FEAT_NV is not implemented, then the value 0b1110 is not permitted.

If FEAT_E2H0 is implemented and FEAT_VHE is not implemented, then HCR_EL2.E2H is RES0.

If FEAT_E2H0 is implemented and FEAT_VHE is implemented, then HCR_EL2.E2H can be programmed to 0 or 1.

If FEAT_E2H0 is not implemented then:

- FEAT_VHE is implemented.
- HCR_EL2.E2H is RES1 and behaves as though it is 1.

Access to this field is RO.

NV_frac, bits [23:20]

Indicates support for a subset of FEAT_NV and FEAT_NV2 behaviors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NV_frac	Meaning
0b0000	Support for FEAT_NV and FEAT_NV2 is described in ID_AA64MMFR2_EL1.NV.
0b0001	FEAT_NV and FEAT_NV2 are implemented, but all of the following apply: • ID_AA64MMFR2_EL1.NV is 0b0000. • Programming HCR_EL2.{NV, NV2} to {1, 0} behaves as {1, 1}.
0b0010	As 0b0001 and adds stateful bits in specified _EL1 registers for software usage under nested virtualization.

FEAT_NV2p1 implements the functionality indicated by the value 0b0010.

Access to this field is RO.

FGWTE3, bits [19:16]

Indicates support for Fine Grained Write Trap EL3 feature.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FGWTE3	Meaning
0b0000	Fine Grained Write Trap EL3 is not supported.
0b0001	Fine Grained Write Trap EL3 is supported.

All other values are reserved.

FEAT_FGWTE3 implements the functionality identified by the value 0b0001.

Access to this field is RO.

HACDBS, bits [15:12]

Support for Hardware accelerator for cleaning Dirty state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HACDBS	Meaning
0b0000	Hardware accelerator for cleaning Dirty state is not supported.
0b0001	Hardware accelerator for cleaning Dirty state is supported.

All other values are reserved.

FEAT_HACDBS implements the functionality identified by the value 0b0001.

Access to this field is RO.

ASID2, bits [11:8]

Indicates support for concurrent use of two ASIDs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ASID2	Meaning
0b0000	FEAT_ASID2 is not implemented.
0b0001	FEAT_ASID2 is implemented.

All other values are reserved.

From Armv9.5, the value 0b0000 is not permitted.

Access to this field is RO.

EIESB, bits [7:4]

When FEAT_IESB is implemented:

Early Implicit Error Synchronization event. Indicates whether the implicit Error synchronization event inserted on taking an exception to ELx when SCTLR_ELx.IESB is 1 is inserted before or after the exception is taken.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EIESB	Meaning
0b1111	FEAT_IESB is implemented. When SError exceptions are routed to EL3, and either FEAT_DoubleFault is not implemented or the Effective value of SCR_EL3.NMEA is 1, an implicit Error synchronization event might be inserted after an exception is taken to EL3.
0b0000	FEAT_IESB is not implemented, or behavior is not described.
0b0001	FEAT_IESB is implemented. When SError exceptions are routed to EL3, and either FEAT_DoubleFault is not implemented or the Effective value of SCR_EL3.NMEA is 1, an implicit Error synchronization event is inserted before an exception taken to EL3.
0b0010	As 0b0001, and also: • When SError exceptions are routed to EL1, and either FEAT_DoubleFault2 is not implemented or the Effective value of SCTLR2_EL1.NMEA is 1, an implicit Error synchronization event is inserted before an exception taken to EL1. • When SError exceptions are routed to EL2, and either FEAT_DoubleFault2 is not implemented or the Effective value of SCTLR2_EL2.NMEA is 1, an implicit Error synchronization event is inserted before an exception taken to EL2.

All other values are reserved.

This field describes the PE behavior on taking an exception to ELx when SCTLR_ELx.IESB is 1. This field does not describe the behavior when SCTLR_ELx.IESB is 0.

The behavior described by this field only applies for the conditions described above. For example, if ID_AA64MMFR4_EL1.EIESB reads as 0b0001, then it does not describe the behavior when SError exceptions are not routed to EL3, or when FEAT_DoubleFault is implemented and the Effective value of SCR_EL3.NMEA is 0.

Inserting the event before the exception is taken means that if the Error synchronization event causes an SError exception to become pending, and SError exceptions are not masked and not disabled, then the SError exception is taken in place of the original exception.

When FEAT_IESB is not implemented, the only permitted value of this field is 0b0000.

Access to this field is RO.

Otherwise:

Reserved, RES0.

PoPS, bits [3:0]

Support for the clean and invalidate to the Point of Physical Storage instructions:

- DC CIVAPS.
- If FEAT_MTE2 is implemented, DC CIGDVAPS.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PoPS	Meaning
0b0000	The System instructions to clean and invalidate to the Point of Physical Storage are not implemented.
0b0001	The specified System instructions to clean and invalidate to the Point of Physical Storage are implemented.

FEAT_PoPS implements the functionality described by the value 0b0001.

All other values are reserved.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing ID_AA64MMFR4_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64MMFR4_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0111	0b100

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_AA64MMFR4_EL1) || boolean
     ↪IMPLEMENTATION_DEFINED "ID_AA64MMFR4_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64MMFR4_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64MMFR4_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64MMFR4_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.93 ID_AA64PFR0_EL1, AArch64 Processor Feature Register 0

The ID_AA64PFR0_EL1 characteristics are:

Purpose

Provides additional information about implemented PE features in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

The external register EDPFR gives information from this register.

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64PFR0_EL1 are UNDEFINED.

Attributes

ID_AA64PFR0_EL1 is a 64-bit register.

Field descriptions

CSV3, bits [63:60]

Speculative use of faulting data.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CSV3	Meaning
`0b0000`	This PE does not disclose whether data loaded or read from a register under speculation where the data load or register read would not be permitted architecturally, can be used by instructions newer than the load or register read in a manner that allows the value of the inaccessible data to be recovered by code architecturally executed.
`0b0001`	Data loaded or read from a register under speculation where the data load or register read would not be permitted architecturally, cannot be used by instructions newer than the load or register read in a manner that allows the value of the inaccessible data to be recovered by code architecturally executed.

All other values are reserved.

FEAT_CSV3 implements the functionality identified by the value `0b0001`.

If FEAT_E0PD is implemented, FEAT_CSV3 must be implemented.

From Armv8.5, the value `0b0000` is not permitted.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CSV2, bits [59:56]

Speculative use of out of context prediction resources.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CSV2	Meaning
0b0000	The implementation does not disclose whether FEAT_CSV2 is implemented.
0b0001	FEAT_CSV2 is implemented, but FEAT_CSV2_2 and FEAT_CSV2_3 are not implemented. ID_AA64PFR1_EL1.CSV2_frac determines whether either or both of FEAT_CSV2_1p1 or FEAT_CSV2_1p2 are implemented.
0b0010	FEAT_CSV2_2 is implemented, but FEAT_CSV2_3 is not implemented.
0b0011	FEAT_CSV2_3 is implemented.

All other values are reserved.

FEAT_CSV2 implements the functionality identified by the value 0b0001.

FEAT_CSV2_2 implements the functionality identified by the value 0b0010.

FEAT_CSV2_3 implements the functionality identified by the feature 0b0011.

From Armv8.5, the value 0b0000 is not permitted.

Access to this field is RO.

RME, bits [55:52]

Realm Management Extension (RME).

The value of this field is an IMPLEMENTATION DEFINED choice of:

RME	Meaning
0b0000	Realm Management Extension not implemented.
0b0001	RMEv1 is implemented.
0b0010	As 0b0001, and adds support for the GPC2 Extension.
0b0011	As 0b0010, and adds support for the GPC3 Extension.

All other values are reserved.

FEAT_RME implements the functionality identified by the value 0b0001.

FEAT_RME_GPC2 implements the functionality identified by the value 0b0010.

FEAT_RME_GPC3 implements the functionality identified by the value 0b0011.

Access to this field is RO.

DIT, bits [51:48]

Data Independent Timing.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

DIT	Meaning
0b0000	AArch64 does not guarantee constant execution time of any instructions.
0b0001	AArch64 provides the PSTATE.DIT mechanism to guarantee constant execution time of certain instructions.

All other values are reserved.

FEAT_DIT implements the functionality identified by the value 0b0001.

From Armv8.4, the value 0b0000 is not permitted.

Access to this field is RO.

AMU, bits [47:44]

Indicates support for Activity Monitors Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AMU	Meaning
0b0000	Activity Monitors Extension is not implemented.
0b0001	FEAT_AMUv1 is implemented.
0b0010	FEAT_AMUv1p1 is implemented. As 0b0001 and adds support for virtualization of the activity monitor event counters.

All other values are reserved.

FEAT_AMUv1 implements the functionality identified by the value 0b0001.

FEAT_AMUv1p1 implements the functionality identified by the value 0b0010.

In Armv8.0, the only permitted value is 0b0000.

In Armv8.4, the permitted values are 0b0000 and 0b0001.

From Armv8.6, the permitted values are 0b0000, 0b0001, and 0b0010.

Access to this field is RO.

MPAM, bits [43:40]

Indicates the major version number of support for the MPAM Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MPAM	Meaning
0b0000	The major version number of the MPAM extension is 0.
0b0001	The major version number of the MPAM extension is 1.

All other values are reserved.

When combined with the minor version number from ID_AA64PFR1_EL1.MPAM_frac, the “major.minor” version is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

MPAM Extension version	MPAM	MPAM_frac
Not implemented.	0b0000	0b0000
v0.1 is implemented.	0b0000	0b0001
v1.0 is implemented.	0b0001	0b0000
v1.1 is implemented.	0b0001	0b0001

For more information, see 'The Memory Partitioning and Monitoring (MPAM) Extension'.

Access to this field is RO.

SEL2, bits [39:36]

Secure EL2.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SEL2	Meaning
0b0000	Secure EL2 is not implemented.
0b0001	Secure EL2 is implemented.

All other values are reserved.

FEAT_SEL2 implements the functionality identified by the value 0b0001.

From Armv8.4, if Secure state and EL2 are implemented, the value 0b0000 is not permitted.

From Armv8.4, if Secure state is not implemented, or if EL2 is not implemented, the only permitted value is 0b0000.

Access to this field is RO.

SVE, bits [35:32]

Scalable Vector Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SVE	Meaning
0b0000	SVE architectural state and programmers' model are not implemented.
0b0001	SVE architectural state and programmers' model are implemented.

All other values are reserved.

FEAT_SVE implements the functionality identified by the value 0b0001.

If implemented, refer to ID_AA64ZFR0_EL1 for information about which SVE instructions are available.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

RAS, bits [31:28]

RAS Extension version.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RAS	Meaning
0b0000	No RAS Extension.
0b0001	Support for the Reliability, Availability, and Serviceability Extension is implemented. The ESB instruction and the Error synchronization event are supported.
0b0010	As 0b0001, and adds support for: • If EL3 is implemented, FEAT_DoubleFault. • Additional ERXMISC<m>_EL1 System registers. • Additional System registers ERXPFGCDN_EL1, ERXPFGCTL_EL1, and ERXPFGF_EL1, and the SCR_EL3.FIEN and HCR_EL2.FIEN trap controls, to support the optional RAS Common Fault Injection Model Extension. Error records accessed through System registers conform to RAS System Architecture v1.1, which includes simplifications to ERR<n>STATUS and support for the optional RAS Timestamp and RAS Common Fault Injection Model Extensions.
0b0011	As 0b0010 and adds support for: • The error group status register, ERXGSR_EL1. • The SCR_EL3.TWERR write trap control for error record System registers. • Additional fields in ESR_ELx.ISS for error exceptions. Error records accessed through System registers conform to either RAS System Architecture v1.1 or RAS System Architecture v2.

All other values are reserved.

FEAT_RAS implements the functionality identified by the value 0b0001.

FEAT_RASv1p1 and FEAT_DoubleFault implement the functionality identified by the value 0b0010.

FEAT_RASv2 implements the functionality identified by the value 0b0011.

In Armv8.0 and Armv8.1, the permitted values are 0b0000 and 0b0001.

From Armv8.2, the value 0b0000 is not permitted.

From Armv8.4, if FEAT_DoubleFault is implemented or ERRIDR_EL1.NUM is nonzero, the value 0b0001 is not permitted.

Note

When the value of this field is 0b0001, ID_AA64PFR1_EL1.RAS_frac indicates whether FEAT_RASv1p1 is implemented.

From Armv8.9, if ERRIDR_EL1.NUM is nonzero, the value 0b0010 is not permitted.

Access to this field is RO.

GIC, bits [27:24]

System register GIC CPU interface.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

GIC	Meaning
0b0000	GIC CPU interface system registers not implemented.
0b0001	System register interface to versions 3.0 and 4.0 of the GIC CPU interface is supported.
0b0011	System register interface to version 4.1 of the GIC CPU interface is supported.

All other values are reserved.

Access to this field is RO.

AdvSIMD, bits [23:20]

Advanced SIMD.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AdvSIMD	Meaning
0b0000	Advanced SIMD is implemented, including support for the following SISD and SIMD operations: • Integer byte, halfword, word and doubleword element operations. • Single-precision and double-precision floating-point arithmetic. • Conversions between single-precision and half-precision data types, and double-precision and half-precision data types.
0b0001	As for 0b0000, and also includes support for half-precision floating-point arithmetic.
0b1111	Advanced SIMD is not implemented.

All other values are reserved.

This field must have the same value as the FP field.

The permitted values are:

- 0b0000 in an implementation with Advanced SIMD support that does not include the FEAT_FP16 extension.
- 0b0001 in an implementation with Advanced SIMD support that includes the FEAT_FP16 extension.
- 0b1111 in an implementation without Advanced SIMD support.

Access to this field is RO.

FP, bits [19:16]

Floating-point.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FP	Meaning
0b0000	Floating-point is implemented, and includes support for: • Single-precision and double-precision floating-point types. • Conversions between single-precision and half-precision data types, and double-precision and half-precision data types.
0b0001	As for 0b0000, and also includes support for half-precision floating-point arithmetic.
0b1111	Floating-point is not implemented.

All other values are reserved.

This field must have the same value as the AdvSIMD field.

The permitted values are:

- 0b0000 in an implementation with floating-point support that does not include the FEAT_FP16 extension.
- 0b0001 in an implementation with floating-point support that includes the FEAT_FP16 extension.
- 0b1111 in an implementation without floating-point support.

Access to this field is RO.

EL3, bits [15:12]

EL3 Exception level handling.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EL3	Meaning
0b0000	EL3 is not implemented.
0b0001	EL3 can be executed in AArch64 state only.
0b0010	EL3 can be executed in either AArch64 or AArch32 state.

All other values are reserved.

The value 0b0010 is not permitted in Armv9-A implementations.

Access to this field is RO.

EL2, bits [11:8]

EL2 Exception level handling.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EL2	Meaning
0b0000	EL2 is not implemented.
0b0001	EL2 can be executed in AArch64 state only.
0b0010	EL2 can be executed in either AArch64 or AArch32 state.

All other values are reserved.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The value `0b0010` is not permitted in Armv9-A implementations.

Access to this field is RO.

EL1, bits [7:4]

EL1 Exception level handling.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EL1	Meaning
`0b0001`	EL1 can be executed in AArch64 state only.
`0b0010`	EL1 can be executed in either AArch64 or AArch32 state.

All other values are reserved.

The value `0b0010` is not permitted in Armv9-A implementations.

Access to this field is RO.

EL0, bits [3:0]

EL0 Exception level handling.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EL0	Meaning
`0b0001`	EL0 can be executed in AArch64 state only.
`0b0010`	EL0 can be executed in either AArch64 or AArch32 state.

All other values are reserved.

Access to this field is RO.

Accessing ID_AA64PFR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64PFR0_EL1
```

op0	op1	CRn	CRm	op2
`0b11`	`0b000`	`0b0000`	`0b0100`	`0b000`

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64PFR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64PFR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64PFR0_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.94 ID_AA64PFR1_EL1, AArch64 Processor Feature Register 1

The ID_AA64PFR1_EL1 characteristics are:

Purpose

Provides additional information about implemented PE features in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64PFR1_EL1 are UNDEFINED.

Attributes

ID_AA64PFR1_EL1 is a 64-bit register.

Field descriptions

PFAR, bits [63:60]

Support for physical fault address registers, FEAT_PFAR.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PFAR	Meaning
0b0000	FEAT_PFAR is not implemented.
0b0001	FEAT_PFAR is implemented. Includes support for the PFAR_ELx and, if EL3 is implemented, MFAR_EL3 registers.

All other values are reserved.

FEAT_PFAR implements the functionality identified by the value 0b0001.

Access to this field is RO.

DF2, bits [59:56]

Support for error exception routing extensions, FEAT_DoubleFault2.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

DF2	Meaning
0b0000	FEAT_DoubleFault2 is not implemented. **Note** This does not mean that FEAT_DoubleFault, as identified by ID_AA64PFR0_EL1.RAS >= 0b0010, is not implemented.
0b0001	FEAT_DoubleFault2 is implemented. As ID_AA64PFR0_EL1.RAS == 0b0010, and also includes support for routing error exceptions: • Traps for masked error exceptions, HCRX_EL2.TMEA and SCR_EL3.TMEA. • Additional controls for masking SError exceptions, SCTLR2_EL1.NMEA, and SCTLR2_EL2.NMEA. • Additional controls for taking external aborts to the SError exception vector, SCTLR2_EL1.EASE and SCTLR2_EL2.EASE.

All other values are reserved.

FEAT_DoubleFault2 implements the functionality identified by the value 0b0001.

Access to this field is RO.

MTEX, bits [55:52]

Support for additional MTE tag checking modes.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MTEX	Meaning
0b0000	Canonical Tag checking and Memory tagging with Address tagging disabled are not supported.
0b0001	The following additional tag checking modes for MTE are supported: • Canonical Tag checking. • Memory tagging with Address tagging disabled.

All other values are reserved.

This field is valid only if ID_AA64PFR1_EL1.MTE >= 0b0010.

FEAT_MTE_NO_ADDRESS_TAGS and FEAT_MTE_CANONICAL_TAGS implement the functionality identified by the value 0b0001.

If FEAT_MTE2 is not implemented, the value 0b0001 is not permitted.

From Armv8.9, if FEAT_MTE2 is implemented, the value 0b0000 is not permitted.

Access to this field is RO.

THE, bits [51:48]

Support for Translation Hardening Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

THE	Meaning
0b0000	Translation Hardening Extension is not implemented.
0b0001	The RCW and RCWS instructions, their associated registers and traps are supported. If EL2 is implemented, the AssuredOnly check, TopLevel check, and their associated controls are implemented. If EL2 and FEAT_GCS are implemented, VTCR_EL2.GCSH is implemented.

All other values are reserved.

FEAT_THE implements the functionality identified by the value 0b0001.

Access to this field is RO.

GCS, bits [47:44]

Support for Guarded Control Stack.

The value of this field is an IMPLEMENTATION DEFINED choice of:

GCS	Meaning
0b0000	Guarded Control Stack is not implemented.
0b0001	Guarded Control Stack is implemented.

All other values are reserved.

FEAT_GCS implements the functionality identified by the value 0b0001.

Access to this field is RO.

MTE_frac, bits [43:40]

Support for Asynchronous reporting of a Tag Check Fault.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MTE_frac	Meaning
0b0000	Asynchronous reporting of a Tag Check Fault is supported.
0b1111	Asynchronous reporting of a Tag Check Fault is not supported.

All other values are reserved.

This field is valid only if ID_AA64PFR1_EL1.MTE >= 0b0010.

FEAT_MTE_ASYNC implements the functionality identified by the value 0b0000.

If FEAT_MTE_ASYM_FAULT is implemented this field must be 0b0000.

Access to this field is RO.

NMI, bits [39:36]

Non-maskable Interrupt. Indicates support for Non-maskable interrupts.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

NMI	Meaning
0b0000	SCTLR_ELx.{SPINTMASK, NMI} and PSTATE.ALLINT with its associated instructions are not supported.
0b0001	SCTLR_ELx.{SPINTMASK, NMI} and PSTATE.ALLINT with its associated instructions are supported.

All other values are reserved.

FEAT_NMI implements the functionality identified by the value 0b0001.

From Armv8.8, the value 0b0000 is not permitted.

Access to this field is RO.

CSV2_frac, bits [35:32]

CSV2 fractional field.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CSV2_frac	Meaning
0b0000	Either ID_AA64PFR0_EL1.CSV2 is not 0b0001, or the implementation does not disclose whether FEAT_CSV2_1p1 is implemented. FEAT_CSV2_1p2 is not implemented.
0b0001	FEAT_CSV2_1p1 is implemented, but FEAT_CSV2_1p2 is not implemented.
0b0010	FEAT_CSV2_1p2 is implemented.

All other values are reserved.

FEAT_CSV2_1p1 implements the functionality identified by the value 0b0001.

FEAT_CSV2_1p2 implements the functionality identified by the value 0b0010.

From Armv8.0, the permitted values are 0b0000, 0b0001, and 0b0010.

The values 0b0001 and 0b0010 are permitted only when ID_AA64PFR0_EL1.CSV2 is 0b0001.

Access to this field is RO.

RNDR_trap, bits [31:28]

Random Number trap to EL3 field.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RNDR_trap	Meaning
0b0000	Trapping of RNDR and RNDRRS to EL3 is not supported.
0b0001	Trapping of RNDR and RNDRRS to EL3 is supported. SCR_EL3.TRNDR is present.

All other values are reserved.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FEAT_RNG_TRAP implements the functionality identified by the value `0b0001`.

Access to this field is RO.

SME, bits [27:24]

Scalable Matrix Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SME	Meaning
`0b0000`	SME architectural state and programmers' model are not implemented.
`0b0001`	SME architectural state and programmers' model are implemented.
`0b0010`	As `0b0001`, plus the SME2 ZT0 register.

All other values are reserved.

FEAT_SME implements the functionality identified by the value `0b0001`.

FEAT_SME2 implements the functionality identified by the value `0b0010`.

From Armv9.2, the permitted values are `0b0000`, `0b0001`, and `0b0010`.

If implemented, refer to ID_AA64SMFR0_EL1 and ID_AA64ZFR0_EL1 for information about which SME and SVE instructions are available.

Access to this field is RO.

Bits [23:20]

Reserved, RES0.

MPAM_frac, bits [19:16]

Indicates the minor version number of support for the MPAM Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MPAM_frac	Meaning
`0b0000`	The minor version number of the MPAM extension is 0.
`0b0001`	The minor version number of the MPAM extension is 1.

All other values are reserved.

When combined with the major version number from ID_AA64PFR0_EL1.MPAM, The combined "major.minor" version is:

MPAM Extension version	MPAM	MPAM_frac
Not implemented.	`0b0000`	`0b0000`

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

MPAM Extension version	MPAM	MPAM_frac
v0.1 is implemented.	0b0000	0b0001
v1.0 is implemented.	0b0001	0b0000
v1.1 is implemented.	0b0001	0b0001

For more information, see 'The Memory Partitioning and Monitoring (MPAM) Extension'.

Access to this field is RO.

RAS_frac, bits [15:12]

RAS Extension fractional field.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RAS_frac	Meaning
0b0000	If ID_AA64PFR0_EL1.RAS == 0b0001, support for the Reliability, Availability, and Serviceability Extension is implemented.
0b0001	If ID_AA64PFR0_EL1.RAS == 0b0001, as 0b0000 and adds support for: • Additional ERXMISC<m>_EL1 System registers. • Additional System registers ERXPFGCDN_EL1, ERXPFGCTL_EL1, and ERXPFGF_EL1, and the SCR_EL3.FIEN and HCR_EL2.FIEN trap controls, to support the optional RAS Common Fault Injection Model Extension. Error records accessed through System registers conform to RAS System Architecture v1.1, which includes simplifications to ERR<n>STATUS, and support for the optional RAS Timestamp and RAS Common Fault Injection Model Extensions.

All other values are reserved.

FEAT_RAS implements the functionality identified by the value 0b0000.

FEAT_RASv1p1 implements the functionality identified by the value 0b0001.

This field is valid only if ID_AA64PFR0_EL1.RAS == 0b0001.

Access to this field is RO.

MTE, bits [11:8]

Support for the Memory Tagging Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MTE	Meaning
0b0000	Memory Tagging Extension is not implemented.
0b0001	Instruction-only Memory Tagging Extension is implemented.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

MTE	Meaning
0b0010	As 0b0001, and adds: • Support for in-memory Allocation tags. • Support for synchronous tag checking. • Optional support for Asynchronous reporting of a Tag Check Fault, identified as FEAT_MTE_ASYNC. Support for FEAT_MTE_ASYNC is indicated by ID_AA64PFR1_EL1.MTE_frac.
0b0011	As 0b0010, except that support for FEAT_MTE_ASYNC is mandatory, and adds support for Asymmetric Tag Check Fault handling, identified as FEAT_MTE_ASYM_FAULT.

All other values are reserved.

FEAT_MTE implements the functionality identified by the value 0b0001.

FEAT_MTE2 implements the functionality identified by the value 0b0010.

FEAT_MTE3 implements the functionality identified by the value 0b0011.

From Armv8.7, when the value of this field is >= 0b0010, ID_AA64PFR2_EL1.MTEPERM indicates support for FEAT_MTE_PERM.

From Armv8.7, when the value of this field is >= 0b0010, the following fields indicate support for FEAT_MTE4:

- ID_AA64PFR1_EL1.MTEX
- ID_AA64PFR2_EL1.MTEFAR
- ID_AA64PFR2_EL1.MTESTOREONLY

Access to this field is RO.

SSBS, bits [7:4]

Speculative Store Bypassing controls in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SSBS	Meaning
0b0000	AArch64 provides no mechanism to control the use of Speculative Store Bypassing.
0b0001	AArch64 provides the PSTATE.SSBS mechanism to mark regions that are Speculative Store Bypass Safe.
0b0010	As 0b0001, and adds the MSR and MRS instructions to directly read and write the PSTATE.SSBS field.

All other values are reserved.

FEAT_SSBS implements the functionality identified by the value 0b0001.

FEAT_SSBS2 implements the functionality identified by the value 0b0010.

Access to this field is RO.

BT, bits [3:0]

Branch Target Identification mechanism support in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

BT	Meaning
0b0000	The Branch Target Identification mechanism is not implemented.
0b0001	The Branch Target Identification mechanism is implemented.

All other values are reserved.

FEAT_BTI implements the functionality identified by the value 0b0001.

From Armv8.5, the value 0b0000 is not permitted.

Access to this field is RO.

Accessing ID_AA64PFR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64PFR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0100	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64PFR1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64PFR1_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64PFR1_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.95 ID_AA64PFR2_EL1, AArch64 Processor Feature Register 2

The ID_AA64PFR2_EL1 characteristics are:

Purpose

Provides additional information about implemented PE features in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AA64PFR2_EL1 are UNDEFINED.

Attributes

ID_AA64PFR2_EL1 is a 64-bit register.

Field descriptions

Bits [63:36]

Reserved, RES0.

FPMR, bits [35:32]

Indicates support for FPMR.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPMR	Meaning
0b0000	FPMR is not implemented.
0b0001	FPMR is implemented.

All other values are reserved.

FEAT_FPMR implements the functionality identified by the value 0b0001.

Access to this field is RO.

Bits [31:20]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

UINJ, bits [19:16]

Support for software injection of Undefined Instruction exceptions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UINJ	Meaning
0b0000	Software injection of Undefined Instruction exceptions is not implemented.
0b0001	Software injection of Undefined Instruction exceptions is implemented.

FEAT_UINJ implements the functionality identified by the value 0b0001.

From Armv9.6, the value 0b0000 is not permitted.

Access to this field is RO.

Bits [15:12]

Reserved, RES0.

MTEFAR, bits [11:8]

Indicates whether FAR_ELx[63:60] are UNKNOWN on a synchronous exception due to a Tag Check Fault.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MTEFAR	Meaning
0b0000	On a synchronous exception due to a Tag Check Fault, FAR_ELx[63:60] are UNKNOWN.
0b0001	On a synchronous exception due to a Tag Check Fault, FAR_ELx[63:60] are not UNKNOWN.

All other values are reserved.

FEAT_MTE_TAGGED_FAR implements the functionality identified by the value 0b0001.

If FEAT_MTE2 is not implemented, the value 0b0001 is not permitted.

From Armv8.9, if FEAT_MTE2 is implemented, the value 0b0000 is not permitted.

Access to this field is RO.

MTESTOREONLY, bits [7:4]

Support for Store-only Tag checking.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MTESTOREONLY	Meaning
0b0000	Store-only Tag checking is not supported.
0b0001	Store-only Tag checking is supported.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

All other values are reserved.

FEAT_MTE_STORE_ONLY implements the functionality identified by the value `0b0001`.

If FEAT_MTE2 is not implemented, the value `0b0001` is not permitted.

From Armv8.9, if FEAT_MTE2 is implemented, the value `0b0000` is not permitted.

Access to this field is RO.

MTEPERM, bits [3:0]

Support for Allocation tag access permissions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MTEPERM	Meaning
`0b0000`	Allocation tag access permissions are not supported.
`0b0001`	Allocation tag access permissions are supported. **Note** NoTagAccess is supported at stage 2 of translation only.

All other values are reserved.

FEAT_MTE_PERM implements the functionality identified by the value `0b0001`

If FEAT_MTE2 is not implemented, the value `0b0001` is not permitted.

From Armv8.9, if FEAT_MTE2 is implemented, the value `0b0000` is not permitted.

Access to this field is RO.

Accessing ID_AA64PFR2_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64PFR2_EL1
```

op0	op1	CRn	CRm	op2
`0b11`	`0b000`	`0b0000`	`0b0100`	`0b010`

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
        ↪then
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_AA64PFR2_EL1) || boolean
         ↪IMPLEMENTATION_DEFINED "ID_AA64PFR2_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64PFR2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
         ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64PFR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64PFR2_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.96 ID_AA64SMFR0_EL1, SME Feature ID Register 0

The ID_AA64SMFR0_EL1 characteristics are:

Purpose

Provides information about the implemented features of the AArch64 Scalable Matrix Extension.

The fields in this register do not follow the standard ID scheme. See Alternative ID scheme used for ID_AA64SMFR0_EL1 and ID_AA64FPFR0_EL1.

Configuration

Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

Attributes

ID_AA64SMFR0_EL1 is a 64-bit register.

Field descriptions

FA64, bit [63]

Indicates support at each Exception Level for execution of the full AArch64 Advanced SIMD and SVE instruction sets when the PE is in Streaming SVE mode.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FA64	Meaning
0b0	Only those AArch64 instructions defined as being legal can be executed in streaming SVE mode.
0b1	All implemented AArch64 instructions are legal for execution in Streaming SVE mode, when enabled by SMCR_EL1.FA64, SMCR_EL2.FA64, and SMCR_EL3.FA64.

FEAT_SME_FA64 implements the functionality identified by the value 0b1.

Access to this field is RO.

Bits [62:61]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

LUTv2, bit [60]

Indicates support for the following additional variants of SME2 lookup table `LUTI4` and `MOVT` instructions:

- A `LUTI4` instruction with 8-bit result elements, two consecutively numbered source vectors, and four consecutively numbered destination vectors.
- If FEAT_SME2p1 is implemented, a `LUTI4` instruction with 8-bit result elements, two consecutively numbered source vectors, and four destination vectors with strided register numbering.
- A `MOVT` instruction that copies a single Z source vector to ZT0.

The value of this field is an IMPLEMENTATION DEFINED choice of:

LUTv2	Meaning
`0b0`	The specified instructions are not implemented by this control.
`0b1`	The specified instructions are implemented.

All other values are reserved.

FEAT_SME_LUTv2 implements the functionality identified by the value `0b1`.

Access to this field is RO.

SMEver, bits [59:56]

Indicates support for SME instructions when FEAT_SME is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SMEver	Meaning
`0b0000`	The mandatory SME instructions are implemented.
`0b0001`	As `0b0000`, and adds the mandatory SME2 instructions.
`0b0010`	As `0b0001`, and adds the mandatory SME2.1 instructions.
`0b0011`	As `0b0010`, and adds the mandatory SME2.2 instructions.

All other values are reserved.

FEAT_SME implements the functionality identified by the value `0b0000`.

FEAT_SME2 implements the functionality identified by the value `0b0001`.

FEAT_SME2p1 implements the functionality identified by the value `0b0010`.

From Armv9.4, the value `0b0001` is not permitted.

FEAT_SME2p2 implements the functionality identified by `0b0011`.

From Armv9.6, the values `0b0000` and `0b0010` are not permitted.

Access to this field is RO.

I16I64, bits [55:52]

Indicates support for the following SME instructions that accumulate into 64-bit integer elements in the ZA array:

- The variants of the `ADDHA`, `ADDVA`, `SMOPA`, `SMOPS`, `SUMOPA`, `SUMOPS`, `UMOPA`, `UMOPS`, `USMOPA`, and `USMOPS` instructions that accumulate into 64-bit integer tiles.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- When FEAT_SME2 is implemented, the variants of the ADD, ADDA, SDOT, SMLALL, SMLSLL, SUB, SUBA, SVDOT, UDOT, UMLALL, UMLSLL, and UVDOT instructions that accumulate into 64-bit integer elements in ZA array vectors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

I16I64	Meaning
0b0000	The specified instructions are not implemented by this control.
0b1111	The specified instructions are implemented.

All other values are reserved.

FEAT_SME_I16I64 implements the functionality identified by the value 0b1111.

Access to this field is RO.

Bits [51:49]

Reserved, RES0.

F64F64, bit [48]

Indicates support for the following SME instructions that accumulate into double-precision floating-point elements in the ZA array:

- The variants of the FMOPA and FMOPS instructions that accumulate into double-precision tiles.
- When FEAT_SME2 is implemented, the variants of the FADD, FMLA, FMLS, and FSUB instructions that accumulate into double-precision elements in ZA array vectors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F64F64	Meaning
0b0	The specified instructions are not implemented by this control.
0b1	The specified instructions are implemented .

FEAT_SME_F64F64 implements the functionality identified by the value 0b1.

Access to this field is RO.

I16I32, bits [47:44]

When FEAT_SME2 is implemented:

Indicates support for SME2 SMOPA (2-way), SMOPS (2-way), UMOPA (2-way), and UMOPS (2-way) instructions that accumulate 16-bit outer products into 32-bit integer tiles.

The value of this field is an IMPLEMENTATION DEFINED choice of:

I16I32	Meaning
0b0000	The specified instructions are not implemented by this control.
0b0101	The specified instructions are implemented.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

I16I32	Meaning

All other values are reserved.

If FEAT_SME2 is implemented, the value `0b0000` is not permitted.

Access to this field is RO.

Otherwise:

Reserved, RES0.

B16B16, bit [43]

Indicates support for the SME ZA-targeting non-widening BFloat16 `BFADD`, `BFMLA`, `BFMLS`, `BFMOPA`, `BFMOPS`, and `BFSUB` instructions with BFloat16 operands and results.

The value of this field is an IMPLEMENTATION DEFINED choice of:

B16B16	Meaning
`0b0`	The specified instructions are not implemented by this control.
`0b1`	The specified instructions are implemented.

FEAT_SME_B16B16 implements the functionality identified by the value `0b1`.

Access to this field is RO.

F16F16, bit [42]

Indicates support for the following SME2 half-precision floating-point instructions:

- `FMOPA` and `FMOPS` instructions that accumulate half-precision outer-products into half-precision tiles.
- Multi-vector `FADD`, `FMLA`, `FMLS`, and `FSUB` instructions with half-precision operands and results.
- Multi-vector `FCVT` and `FCVTL` instructions that convert half-precision inputs to single-precision results.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F16F16	Meaning
`0b0`	The specified instructions are not implemented by this control.
`0b1`	The specified instructions are implemented.

FEAT_SME_F16F16 implements the functionality identified by the value `0b1`.

Access to this field is RO.

F8F16, bit [41]

Indicates support for the following SME2 instructions:

- The ZA-targeting FP8 instructions `FDOT` (2-way), `FMLAL`, `FMOPA` (2-way), and `FVDOT` that accumulate into half-precision floating-point elements.
- ZA-targeting non-widening half-precision `FADD` and `FSUB` instructions.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The value of this field is an IMPLEMENTATION DEFINED choice of:

F8F16	Meaning
0b0	The specified instructions are not implemented by this control.
0b1	The specified instructions are implemented.

FEAT_SME_F8F16 implements the functionality identified by the value 0b1.

Access to this field is RO.

F8F32, bit [40]

Indicates support for the SME2 ZA-targeting FP8 FDOT (4-way), FMLALL, FMOPA (4-way), FVDOTB, and FVDOTT instructions that accumulate into single-precision floating-point elements.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F8F32	Meaning
0b0	The specified instructions are not implemented by this control.
0b1	The specified instructions are implemented.

FEAT_SME_F8F32 implements the functionality identified by the value 0b1.

Access to this field is RO.

I8I32, bits [39:36]

When FEAT_SME is implemented:

Indicates support for SME SMOPA, SMOPS, SUMOPA, SUMOPS, UMOPA, UMOPS, USMOPA, and USMOPS instructions that accumulate 8-bit outer products into 32-bit tiles

The value of this field is an IMPLEMENTATION DEFINED choice of:

I8I32	Meaning
0b0000	The specified instructions are not implemented by this control.
0b1111	The specified instructions are implemented.

All other values are reserved.

If FEAT_SME is implemented, the value 0b0000 is not permitted.

Access to this field is RO.

Otherwise:

Reserved, RES0.

F16F32, bit [35]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_SME is implemented:

Indicates support for SME `FMOPA` and `FMOPS` instructions that accumulate half-precision floating-point outer products into single-precision floating-point tiles.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F16F32	Meaning
`0b0`	The specified instructions are not implemented by this control.
`0b1`	The specified instructions are implemented.

If FEAT_SME is implemented, the value `0b0` is not permitted.

Access to this field is RO.

Otherwise:

Reserved, RES0.

B16F32, bit [34]

When FEAT_SME is implemented:

Indicates support for SME `BFMOPA` and `BFMOPS` instructions that accumulate BFloat16 outer products into single-precision floating-point tiles.

The value of this field is an IMPLEMENTATION DEFINED choice of:

B16F32	Meaning
`0b0`	The specified instructions are not implemented by this control.
`0b1`	The specified instructions are implemented.

If FEAT_SME is implemented, the value `0b0` is not permitted.

Access to this field is RO.

Otherwise:

Reserved, RES0.

BI32I32, bit [33]

When FEAT_SME2 is implemented:

Indicates support for SME `BMOPA` and `BMOPS` instructions that accumulate thirty-two 1-bit binary outer products into 32-bit integer tiles.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BI32I32	Meaning
`0b0`	The specified instructions are not implemented by this control.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

0b1	The specified instructions are implemented.

If FEAT_SME2 is implemented, the value 0b0 is not permitted.

Access to this field is RO.

Otherwise:

Reserved, RES0.

F32F32, bit [32]

When FEAT_SME is implemented:

Indicates support for SME FMOPA and FMOPS instructions that accumulate single-precision floating-point outer products into single-precision floating-point tiles.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F32F32	Meaning
0b0	The specified instructions are not implemented by this control.
0b1	The specified instructions are implemented.

If FEAT_SME is implemented, the value 0b0 is not permitted.

Access to this field is RO.

Otherwise:

Reserved, RES0.

Bit [31]

Reserved, RES0.

SF8FMA, bit [30]

Indicates support for the SVE2 FP8 to single-precision and half-precision multiply-accumulate FMLALB, FMLALT, FMLALLBB, FMLALLBT, FMLALLTB, and FMLALLTT instructions when the PE is in Streaming SVE mode.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SF8FMA	Meaning
0b0	This control has no effect on Streaming SVE mode behavior.
0b1	The specified instructions are supported in Streaming SVE mode.

Note

Other features may support some of the specified instructions in Non-streaming SVE mode.

If FEAT_SME2 and FEAT_FP8FMA are implemented, the value 0b0 is not permitted.

FEAT_SSVE_FP8FMA implements the functionality identified by the value 0b1.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SF8DP4, bit [29]

Indicates support for the SVE2 FP8 to single-precision 4-way dot product `FDOT` (4-way) instructions when the PE is in Streaming SVE mode.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SF8DP4	Meaning
`0b0`	This control has no effect on Streaming SVE mode behavior.
`0b1`	The specified instructions are supported in Streaming SVE mode.

Note

Other features may support some of the specified instructions in Non-streaming SVE mode.

If FEAT_SME2 and FEAT_FP8DOT4 are implemented, the value `0b0` is not permitted.

FEAT_SSVE_FP8DOT4 implements the functionality identified by the value `0b1`.

Access to this field is RO.

SF8DP2, bit [28]

Indicates support for the SVE2 FP8 to half-precision 2-way dot product `FDOT` (2-way) instructions when the PE is in Streaming SVE mode.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SF8DP2	Meaning
`0b0`	This control has no effect on Streaming SVE mode behavior.
`0b1`	The specified instructions are supported in Streaming SVE mode.

Note

Other features may support some of the specified instructions in Non-streaming SVE mode.

If FEAT_SME2 and FEAT_FP8DOT2 are implemented, the value `0b0` is not permitted.

FEAT_SSVE_FP8DOT2 implements the functionality identified by the value `0b1`.

Access to this field is RO.

Bits [27:26]

Reserved, RES0.

SBitPerm, bit [25]

Indicates support for the SVE bit permute instructions identified as implemented by ID_AA64ZFR0_EL1.BitPerm, when the PE is in Streaming SVE mode.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SBitPerm	Meaning
0b0	This control has no effect on Streaming SVE mode behavior.
0b1	The specified instructions are implemented when the PE is in Streaming SVE mode.

Note

Other features may support some of the specified instructions in Non-streaming SVE mode.

FEAT_SSVE_BitPerm implements the functionality identified by the value 0b1.

Access to this field is RO.

AES, bit [24]

Indicates support for the SVE AES and 128-bit polynomial multiply long instructions identified as implemented by ID_AA64ZFR0_EL1.AES, when the PE is in Streaming SVE mode.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AES	Meaning
0b0	This control has no effect on Streaming SVE mode behavior.
0b1	The specified instructions are supported when the PE is in Streaming SVE mode.

Note

Other features may support some of the specified instructions in Non-streaming SVE mode.

FEAT_SSVE_AES implements the functionality identified by the value 0b1.

Access to this field is RO.

SFEXPA, bit [23]

Indicates support for the SVE FEXPA instruction when the PE is in Streaming SVE mode.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SFEXPA	Meaning
0b0	This control has no effect on Streaming SVE mode behavior.
0b1	The specified instruction is supported when the PE is in Streaming SVE mode.

Note

Other features may support some of the specified instructions in Non-streaming SVE mode.

FEAT_SSVE_FEXPA implements the functionality identified by the value 0b1.

Access to this field is RO.

Bits [22:17]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

STMOP, bit [16]

Indicates support for the following SME Structured sparsity outer product instructions:

- `BFTMOPA` (non-widening), if FEAT_SME_B16B16 is implemented.
- `BFTMOPA` (widening, BF16 to FP32).
- `FTMOPA` (non-widening, FP16), if FEAT_SME_F16F16 is implemented.
- `FTMOPA` (non-widening, FP32).
- `FTMOPA` (widening, 2-way, FP16 to FP32).
- `FTMOPA` (widening, 2-way, FP8 to FP16), if FEAT_SME_F8F16 is implemented.
- `FTMOPA` (widening, 4-way, FP8 to FP32) if FEAT_SME_F8F32 is implemented.
- `STMOPA`, `SUTMOPA`, `USTMOPA`, `UTMOPA` (4-way, Int8 to Int32). `STMOPA`, `UTMOPA` (2-way, Int16 to Int32).

The value of this field is an IMPLEMENTATION DEFINED choice of:

STMOP	Meaning
`0b0`	The specified instructions are not implemented by this control.
`0b1`	The specified instructions are implemented.

FEAT_SME_TMOP implements the functionality identified by the value `0b1`.

Access to this field is RO.

Bits [15:1]

Reserved, RES0.

SMOP4, bit [0]

Indicates support for the following SME Quarter-tile outer product instructions:

- `BFMOP4A`, `BFMOP4S` (non-widening, BF16), if FEAT_SME_B16B16 is implemented.
- `BFMOP4A`, `BFMOP4S` (widening, 2-way, BF16 to FP32).
- `FMOP4A`, `FMOP4S` (non-widening, FP16), if FEAT_SME_F16F16 is implemented.
- `FMOP4A`, `FMOP4S` (non-widening, FP32).
- `FMOP4A`, `FMOP4S` (non-widening, FP64), if FEAT_SME_F64F64 is implemented.
- `FMOP4A`, `FMOP4S` (widening, 2-way, FP16 to FP32).
- `FMOP4A`(widening, 2-way, FP8 to FP16), if FEAT_SME_F8F16 is implemented.
- `FMOP4A`(widening, 4-way, FP8 to FP32) instruction, if FEAT_SME_F8F32 is implemented.
- `SMOP4A`, `SMOP4S`, `SUMOP4A`, `SUMOP4S`, `UMOP4A`, `UMOP4S`, `USMOP4A`, `USMOP4S` (4-way, Int8 to Int32).
- `SMOP4A`, `SMOP4S`, `SUMOP4A`, `SUMOP4S`, `UMOP4A`, `UMOP4S`, `USMOP4A`, `USMOP4S` (4-way, Int16 to Int64), if FEAT_SME_I16I64 is implemented.
- `SMOP4A`, `SMOP4S`, `UMOP4A`, `UMOP4S` (2-way, Int16 to Int32).

The value of this field is an IMPLEMENTATION DEFINED choice of:

SMOP4	Meaning
`0b0`	The specified instructions are not implemented by this control.
`0b1`	The specified instructions are implemented.

FEAT_SME_MOP4 implements the functionality identified by the value `0b1`.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing ID_AA64SMFR0_EL1

This register is read-only and can be accessed from EL1 and higher.

This register is only accessible from the AArch64 state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64SMFR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0100	0b101

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_AA64SMFR0_EL1) || boolean
     ↪IMPLEMENTATION_DEFINED "ID_AA64SMFR0_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64SMFR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64SMFR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64SMFR0_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.97 ID_AA64ZFR0_EL1, SVE Feature ID Register 0

The ID_AA64ZFR0_EL1 characteristics are:

Purpose

Provides additional information about the implemented features of the AArch64 Scalable Vector Extension instruction set, when FEAT_SVE or FEAT_SME is implemented.

For general information about the interpretation of the ID registers, see Principles of the ID scheme for fields in ID registers.

Configuration

Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.
If FEAT_SME is implemented and FEAT_SVE is not implemented, then SVE instructions can only be executed when the PE is in Streaming SVE mode and the instructions are legal for execution in Streaming SVE mode.

Attributes

ID_AA64ZFR0_EL1 is a 64-bit register.

Field descriptions

Bits [63:60]

Reserved, RES0.

F64MM, bits [59:56]

Indicates support for the SVE double-precision floating-point matrix multiply-accumulate instruction `FMMLA`, the `LD1RO*` instructions, the 128-bit element variants of the SVE `TRN1`, `TRN2`, `UZP1`, `UZP2`, `ZIP1`, and `ZIP2` instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F64MM	Meaning
`0b0000`	This field does not indicate support for the specified instructions.
`0b0001`	The specified instructions are implemented.

All other values are reserved.

FEAT_F64MM implements the functionality identified by `0b0001`.

The instructions described by nonzero values of this field might not be legal when the PE is in Streaming SVE mode.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F32MM, bits [55:52]

Indicates support for the SVE single-precision floating-point matrix multiply-accumulate instruction `FMMLA`.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F32MM	Meaning
`0b0000`	This field does not indicate support for the specified instruction.
`0b0001`	The specified instruction is implemented.

All other values are reserved.

FEAT_F32MM implements the functionality identified by `0b0001`.

The instructions described by nonzero values of this field might not be legal when the PE is in Streaming SVE mode.

Access to this field is RO.

F16MM, bits [51:48]

Indicates support for the SVE half-precision floating-point matrix multiply-accumulate to single-precision instruction `FMMLA` (widening, FP16 to FP32).

The value of this field is an IMPLEMENTATION DEFINED choice of:

F16MM	Meaning
`0b0000`	This field does not indicate support for the specified instruction.
`0b0001`	The specified instruction is implemented.

FEAT_SVE_F16F32MM implements the functionality identified by `0b0001`.

The instructions described by nonzero values of this field might not be legal when the PE is in Streaming SVE mode.

Access to this field is RO.

I8MM, bits [47:44]

Indicates support for the following SVE 8-bit integer sum-of-products and accumulate to 32-bit integer instructions `SMMLA`, `SUDOT`, `UMMLA`, `USMMLA`, and `USDOT`.

The value of this field is an IMPLEMENTATION DEFINED choice of:

I8MM	Meaning
`0b0000`	This field does not indicate support for the specified instructions.
`0b0001`	The specified instructions are implemented.

All other values are reserved.

FEAT_I8MM implements the functionality identified by `0b0001`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When Advanced SIMD and SVE are both implemented, this field must return the same value as ID_AA64ISAR1_EL1.I8MM.

From Armv8.6, if SVE is implemented, the value `0b0000` is not permitted.

SVE `SMMLA`, `UMMLA`, and `USMMLA` instructions might not be legal when the PE is in Streaming SVE mode.

Access to this field is RO.

SM4, bits [43:40]

Indicates support for SVE SM4 instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SM4	Meaning
`0b0000`	This field does not indicate support for the specified instructions.
`0b0001`	SVE `SM4E` and `SM4EKEY` instructions are implemented.

All other values are reserved.

FEAT_SVE_SM4 implements the functionality identified by `0b0001`.

The instructions described by nonzero values of this field might not be legal when the PE is in Streaming SVE mode.

Access to this field is RO.

Bits [39:36]

Reserved, RES0.

SHA3, bits [35:32]

Indicates support for the SVE SHA3 instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SHA3	Meaning
`0b0000`	This field does not indicate support for the specified instructions.
`0b0001`	SVE `RAX1` instruction is implemented.

All other values are reserved.

FEAT_SVE_SHA3 implements the functionality identified by `0b0001`.

If FEAT_SME2p1 is not implemented, then the instructions described by nonzero values of this field might not be legal when the PE is in Streaming SVE mode.

Access to this field is RO.

Bits [31:28]

Reserved, RES0.

B16B16, bits [27:24]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Indicates support for SVE non-widening BFloat16 instructions and SME multi-vector Z-targeting non-widening BFloat16 instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

B16B16	Meaning
0b0000	This field does not indicate support for the specified instructions.
0b0001	The following non-widening BFloat16 instructions are implemented: • SVE instructions: BFADD, BFCLAMP, BFMAX, BFMAXNM, BFMIN, BFMINNM, BFMLA, BFMLS, BFMUL, and BFSUB. • If FEAT_SME2 is implemented, SME multi-vector Z-targeting instructions: BFCLAMP, BFMAX, BFMAXNM, BFMIN, and BFMINNM.
0b0010	As 0b0001, and adds the following non-widening BFloat16 instructions: • SVE instruction BFSCALE. • If FEAT_SME2 is implemented, SME multi-vector Z-targeting instructions BFMUL and BFSCALE.

FEAT_SVE_B16B16 implements the functionality identified by 0b0001.

FEAT_SVE_BFSCALE implements the functionality identified by 0b0010.

Access to this field is RO.

BF16, bits [23:20]

Indicates support for SVE BFloat16 instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BF16	Meaning
0b0000	SVE BFloat16 instructions are not implemented.
0b0001	SVE BFCVT, BFCVTNT, BFDOT, BFMLALB, BFMLALT, and BFMMLA instructions are implemented.
0b0010	As 0b0001, but the FPCR.EBF field is also supported.

All other values are reserved.

FEAT_BF16 implements the functionality identified by 0b0001.

FEAT_EBF16 implements the functionality identified by 0b0010.

This field must return the same value as ID_AA64ISAR1_EL1.BF16.

SVE BFMMLA instructions might not be legal when the PE is in Streaming SVE mode.

From Armv8.6 and Armv9.1, the value 0b0000 is not permitted.

Access to this field is RO.

BitPerm, bits [19:16]

Indicates support for the SVE bit permute instructions BDEP, BEXT, and BGRP.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

BitPerm	Meaning
0b0000	This field does not indicate support for the specified instructions.
0b0001	The specified instructions are implemented.

All other values are reserved.

FEAT_SVE_BitPerm implements the functionality identified by 0b0001 when the PE is not in Streaming SVE mode.

FEAT_SSVE_BitPerm which is identified in ID_AA64SMFR0_EL1.SBitPerm, indicates whether the FEAT_SVE_BitPerm functionality is implemented when the PE is in Streaming SVE mode.

Access to this field is RO.

EltPerm, bits [15:12]

When FEAT_SVE2p2 is implemented or FEAT_SME2p2 is implemented:

If FEAT_SVE2p2 is implemented, the following SVE instructions are implemented when the PE is not in Streaming SVE mode:

- 8-bit and 16-bit element COMPACT.
- 8-bit, 16-bit, 32-bit, and 64-bit element EXPAND.

If FEAT_SME2p2 is implemented, the following SVE instructions are implemented when the PE is in Streaming SVE mode:

- 8-bit, 16-bit, 32-bit, and 64-bit element COMPACT and EXPAND.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EltPerm	Meaning
0b0000	This field does not indicate support for the specified instructions.
0b0001	The specified instructions are implemented.

If FEAT_SVE2p2 or FEAT_SME2p2 is implemented, the value 0b0000 is not permitted.

Access to this field is RO.

Otherwise:

Reserved, RES0.

Bits [11:8]

Reserved, RES0.

AES, bits [7:4]

Indicates support for SVE Advanced Encryption Standard instructions and 128-bit polynomial multiply long instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

AES	Meaning
0b0000	This field does not indicate support for the specified instructions.
0b0001	The following instructions are implemented: • SVE single-vector AESD, AESE, AESIMC, and AESMC instructions. • The 128-bit destination element variant of the SVE single-vector PMULLB and PMULLT instructions.
0b0010	As 0b0001.
0b0011	As 0b0010, and adds the following SVE instructions: • Multi-vector AESD, AESDIMC, AESE and AESEMC instructions. • Multi-vector 128-bit destination element PMULL and PMLAL instructions.

All other values are reserved.

FEAT_SVE_AES implements the functionality identified by the value 0b0001.

FEAT_SVE_PMULL128 implements the functionality identified by the value 0b0010.

FEAT_SVE_AES2 implements the functionality identified by 0b0011.

If FEAT_SSVE_AES is not implemented, then the instructions described by nonzero values of this field might not be legal when the PE is in Streaming SVE mode.

Access to this field is RO.

SVEver, bits [3:0]

Indicates support for SVE instructions when FEAT_SME or FEAT_SVE is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SVEver	Meaning
0b0000	The SVE instructions are implemented.
0b0001	As 0b0000, and adds the mandatory SVE2 instructions.
0b0010	As 0b0001, and adds the mandatory SVE2.1 instructions.
0b0011	As 0b0010, and adds the mandatory SVE2.2 instructions.

All other values are reserved.

FEAT_SVE2 implements the functionality identified by 0b0001 when the PE is not in Streaming SVE mode.

FEAT_SME implements the functionality identified by 0b0001 when the PE is in Streaming SVE mode.

FEAT_SME2p1 implements the functionality identified by 0b0010 when the PE is in Streaming SVE mode.

FEAT_SVE2p1 implements the functionality identified by 0b0010 when the PE is not in Streaming SVE mode.

FEAT_SVE2p2 implements the functionality identified by 0b0011 when the PE is not in Streaming SVE mode.

FEAT_SME2p2 implements the functionality identified by 0b0011 when the PE is in Streaming SVE mode.

From Armv9, if this register is present, the value 0b0000 is not permitted.

From Armv9.4, the value 0b0001 is not permitted.

From Armv9.6, the value 0b0010 is not permitted.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing ID_AA64ZFR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AA64ZFR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0100	0b100

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_AA64ZFR0_EL1) || boolean
     ↪IMPLEMENTATION_DEFINED "ID_AA64ZFR0_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64ZFR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AA64ZFR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AA64ZFR0_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.98 ID_AFR0_EL1, AArch32 Auxiliary Feature Register 0

The ID_AFR0_EL1 characteristics are:

Purpose

Provides information about the IMPLEMENTATION DEFINED features of the PE in AArch32 state.

Must be interpreted with the Main ID Register, MIDR_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_AFR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_AFR0[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_AFR0_EL1 are UNDEFINED.

Attributes

ID_AFR0_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:

Bits [63:16]

Reserved, RES0.

IMPLEMENTATION DEFINED, bits [15:12]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [11:8]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [7:4]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [3:0]

IMPLEMENTATION DEFINED.

Otherwise:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

63	32
UNKNOWN	

31	0
UNKNOWN	

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_AFR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_AFR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0001	0b011

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AFR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_AFR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_AFR0_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.99 ID_DFR0_EL1, AArch32 Debug Feature Register 0

The ID_DFR0_EL1 characteristics are:

Purpose

Provides top-level information about the debug system in AArch32 state.

Must be interpreted with the Main ID Register, MIDR_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_DFR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_DFR0[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_DFR0_EL1 are UNDEFINED.

Attributes

ID_DFR0_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:

Bits [63:32]

Reserved, RES0.

TraceFilt, bits [31:28]

Armv8.4 Self-hosted Trace Extension version.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TraceFilt	Meaning
0b0000	Armv8.4 Self-hosted Trace Extension not implemented.
0b0001	Armv8.4 Self-hosted Trace Extension implemented.

All other values are reserved.

FEAT_TRF implements the functionality identified by the value 0b0001.

From Armv8.4, if FEAT_ETMv4 is implemented, the value 0b0000 is not permitted.

If FEAT_ETE is implemented, the value 0b0000 is not permitted.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

PerfMon, bits [27:24]

Performance Monitors Extension version.

This field does not follow the standard ID scheme, but uses the alternative ID scheme described in 'Alternative ID scheme used for the Performance Monitors Extension version'

The value of this field is an IMPLEMENTATION DEFINED choice of:

PerfMon	Meaning
0b0000	Performance Monitors Extension not implemented.
0b0001	Performance Monitors Extension, PMUv1 implemented.
0b0010	Performance Monitors Extension, PMUv2 implemented.
0b0011	Performance Monitors Extension, PMUv3 implemented.
0b0100	PMUv3 for Armv8.1. As 0b0011, and adds support for: • Extended 16-bit PMEVTYPER<n>.evtCount field. • If EL2 is implemented, the HDCR.HPMD control.
0b0101	PMUv3 for Armv8.4. As 0b0100, and adds support for the PMMIR register.
0b0110	PMUv3 for Armv8.5. As 0b0101, and adds support for: • 64-bit event counters. • If EL2 is implemented, the HDCR.HCCD control. • If EL3 is implemented, the MDCR_EL3.SCCD control.
0b0111	PMUv3 for Armv8.7. As 0b0110, and adds support for: • The PMCR.FZO and, if EL2 is implemented, HDCR.HPMFZO controls. • If EL3 is implemented, the MDCR_EL3.{MPMX,MCCD} controls.
0b1000	PMUv3 for Armv8.8. As 0b0111, and: • Extends the Common event number space to include 0x0040 to 0x00BF and 0x4040 to 0x40BF. • Removes the CONSTRAINED UNPREDICTABLE behaviors if a reserved or unimplemented PMU event number is selected.
0b1001	PMUv3 for Armv8.9. As 0b1000, and: • Updates the definitions of existing PMU events. • Adds support for the EDECR.PME control.
0b1111	IMPLEMENTATION DEFINED form of performance monitors supported, PMUv3 not supported. Arm does not recommend this value for new implementations.

All other values are reserved.

FEAT_PMUv3 implements the functionality identified by the value 0b0011.

FEAT_PMUv3p1 implements the functionality identified by the value 0b0100.

FEAT_PMUv3p4 implements the functionality identified by the value 0b0101.

FEAT_PMUv3p5 implements the functionality identified by the value 0b0110.

FEAT_PMUv3p7 implements the functionality identified by the value 0b0111.

FEAT_PMUv3p8 implements the functionality identified by the value 0b1000.

FEAT_PMUv3p9 implements the functionality identified by the value 0b1001.

In any Armv8 implementation, the values 0b0001 and 0b0010 are not permitted.

From Armv8.1, if FEAT_PMUv3 is implemented, the value 0b0011 is not permitted.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

From Armv8.4, if FEAT_PMUv3 is implemented, the value `0b0100` is not permitted.

From Armv8.5, if FEAT_PMUv3 is implemented, the value `0b0101` is not permitted.

From Armv8.7, if FEAT_PMUv3 is implemented, the value `0b0110` is not permitted.

From Armv8.8, if FEAT_PMUv3 is implemented, the value `0b0111` is not permitted.

From Armv8.9, if FEAT_PMUv3 is implemented, the value `0b1000` is not permitted.

Access to this field is RO.

MProfDbg, bits [23:20]

M-profile Debug. Support for memory-mapped debug model for M-profile processors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MProfDbg	Meaning
`0b0000`	Not supported.
`0b0001`	Support for M-profile Debug architecture, with memory-mapped access.

All other values are reserved.

In Armv8-A, the only permitted value is `0b0000`.

Access to this field is RO.

MMapTrc, bits [19:16]

Memory-mapped Trace. Support for memory-mapped trace model.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MMapTrc	Meaning
`0b0000`	Not supported.
`0b0001`	Support for Arm trace architecture, with memory-mapped access.

All other values are reserved.

In Armv8-A, the permitted values are `0b0000` and `0b0001`.

For more information, see the Arm® Embedded Trace Macrocell Architecture Specification, ETMv4 (ARM IHI 0064).

Access to this field is RO.

CopTrc, bits [15:12]

Support for System registers-based trace model, using registers in the coproc == `0b1110` encoding space.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CopTrc	Meaning
0b0000	Not supported.
0b0001	Support for Arm trace architecture, with System registers access.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

For more information, see the Arm® Embedded Trace Macrocell Architecture Specification, ETMv4 (ARM IHI 0064).

Access to this field is RO.

MMapDbg, bits [11:8]

Memory-mapped Debug. Support for Armv7 memory-mapped debug model for A and R-profile processors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MMapDbg	Meaning
0b0000	Not supported.
0b0100	Support for Armv7, v7 Debug architecture, with memory-mapped access.
0b0101	Support for Armv7, v7.1 Debug architecture, with memory-mapped access.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

The optional memory map defined by Armv8 is not compatible with Armv7.

Access to this field is RO.

CopSDbg, bits [7:4]

Support for a System registers-based Secure debug model, using registers in the coproc = 0b1110 encoding space, for an A-profile processor that includes EL3.

If EL3 is not implemented and the implemented Security state is Non-secure state, this field is RES0. Otherwise, this field reads the same as bits [3:0].

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

CopDbg, bits [3:0]

Debug architecture version. Indicates presence of Armv8 debug architecture.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CopDbg	Meaning
0b0000	Not supported.
0b0010	Armv6, v6 Debug architecture, with System registers access.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CopDbg	Meaning
0b0011	Armv6, v6.1 Debug architecture, with System registers access.
0b0100	Armv7, v7 Debug architecture, with System registers access.
0b0101	Armv7, v7.1 Debug architecture, with System registers access.
0b0110	Armv8 debug architecture.
0b0111	Armv8.1 debug architecture, FEAT_Debugv8p1.
0b1000	Armv8.2 debug architecture, FEAT_Debugv8p2.
0b1001	Armv8.4 debug architecture, FEAT_Debugv8p4.
0b1010	Armv8.8 debug architecture, FEAT_Debugv8p8.
0b1011	Armv8.9 debug architecture, FEAT_Debugv8p9.

All other values are reserved.

The values 0b0000, 0b0010, 0b0011, 0b0100, and 0b0101 are not permitted in Armv8.

FEAT_Debugv8p1 implements the functionality identified by the value 0b0111.

FEAT_Debugv8p2 implements the functionality identified by the value 0b1000.

FEAT_Debugv8p4 implements the functionality identified by the value 0b1001.

FEAT_Debugv8p8 implements the functionality identified by the value 0b1010.

FEAT_Debugv8p9 implements the functionality identified by the value 0b1011.

From Armv8.1, when FEAT_Debugv8p1 is implemented the value 0b0110 is not permitted.

From Armv8.2, the values 0b0110 and 0b0111 are not permitted.

From Armv8.4, the value 0b1000 is not permitted.

From Armv8.8, the value 0b1001 is not permitted.

From Armv8.9, the value 0b1010 is not permitted.

Access to this field is RO.

Otherwise:

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_DFR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_DFR0_EL1
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0001	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_DFR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_DFR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_DFR0_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.100 ID_DFR1_EL1, AArch32 Debug Feature Register 1

The ID_DFR1_EL1 characteristics are:

Purpose

Provides top-level information about the debug system in AArch32.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

AArch64 System register ID_DFR1_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_DFR1[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_DFR1_EL1 are UNDEFINED.

Attributes

ID_DFR1_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:

Bits [63:8]

Reserved, RES0.

HPMN0, bits [7:4]

Zero PMU event counters for a Guest operating system.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HPMN0	Meaning
0b0000	Setting HDCR.HPMN to zero has CONSTRAINED UNPREDICTABLE behavior.
0b0001	Setting HDCR.HPMN to zero has defined behavior.

All other values are reserved.

If FEAT_PMUv3 is not implemented, FEAT_FGT is not implemented, or EL2 is not implemented, the only permitted value is 0b0000.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FEAT_HPMN0 implements the functionality identified by the value `0b0001`.

From Armv8.8, in an implementation that includes FEAT_PMUv3, FEAT_FGT, and EL2, the value `0b0000` is not permitted.

Access to this field is RO.

MTPMU, bits [3:0]

Multi-threaded PMU extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MTPMU	Meaning
`0b0000`	FEAT_MTPMU not implemented. If FEAT_PMUv3 is implemented, it is IMPLEMENTATION DEFINED whether PMEVTYPER<n>.MT are read/write or RES0.
`0b0001`	FEAT_MTPMU and FEAT_PMUv3 implemented. PMEVTYPER<n>.MT are read/write. When FEAT_MTPMU is disabled, the Effective values of PMEVTYPER<n>.MT are 0.
`0b1111`	FEAT_MTPMU not implemented. If FEAT_PMUv3 is implemented, PMEVTYPER<n>.MT are RES0.

All other values are reserved.

FEAT_MTPMU implements the functionality identified by the value `0b0001`.

From Armv8.6, in an implementation that includes FEAT_PMUv3, the value `0b0000` is not permitted.

In an implementation that does not include FEAT_PMUv3, the value `0b0001` is not permitted.

Access to this field is RO.

Otherwise:

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_DFR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_DFR1_EL1
```

op0	op1	CRn	CRm	op2
`0b11`	`0b000`	`0b0000`	`0b0011`	`0b101`

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
        ↪then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_DFR1_EL1) || boolean
        ↪IMPLEMENTATION_DEFINED "ID_DFR1_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_DFR1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
        ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_DFR1_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_DFR1_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.101 ID_ISAR0_EL1, AArch32 Instruction Set Attribute Register 0

The ID_ISAR0_EL1 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with ID_ISAR1_EL1, ID_ISAR2_EL1, ID_ISAR3_EL1, ID_ISAR4_EL1, and ID_ISAR5_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_ISAR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_ISAR0[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_ISAR0_EL1 are UNDEFINED.

Attributes

ID_ISAR0_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:

Bits [63:28]

Reserved, RES0.

Divide, bits [27:24]

Indicates the implemented Divide instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Divide	Meaning
0b0000	None implemented.
0b0001	Adds SDIV and UDIV in the T32 instruction set.
0b0010	As for 0b0001, and adds SDIV and UDIVin the A32 instruction set.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Debug, bits [23:20]

Indicates the implemented Debug instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Debug	Meaning
0b0000	None implemented.
0b0001	Adds BKPT.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is RO.

Coproc, bits [19:16]

Indicates the implemented System register access instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Coproc	Meaning
0b0000	None implemented, except for instructions separately attributed by the architecture to provide access to AArch32 System registers and System instructions.
0b0001	Adds generic CDP, LDC, MCR, MRC, and STC.
0b0010	As for 0b0001, and adds generic CDP2, LDC2, MCR2, MRC2, and STC2.
0b0011	As for 0b0010, and adds generic MCRR and MRRC.
0b0100	As for 0b0011, and adds generic MCRR2 and MRRC2.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is RO.

CmpBranch, bits [15:12]

Indicates the implemented combined Compare and Branch instructions in the T32 instruction set.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CmpBranch	Meaning
0b0000	None implemented.
0b0001	Adds CBNZ and CBZ.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

BitField, bits [11:8]

Indicates support for the following BitField instructions `BFC`, `BFI`, `SBFX`, and `UBFX`.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BitField	Meaning
`0b0000`	The specified instructions are not implemented.
`0b0001`	The specified instructions are implemented.

All other values are reserved.

In Armv8-A, the only permitted value is `0b0001`.

Access to this field is RO.

BitCount, bits [7:4]

Indicates the implemented Bit Counting instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BitCount	Meaning
`0b0000`	None implemented.
`0b0001`	Adds `CLZ`.

All other values are reserved.

In Armv8-A, the only permitted value is `0b0001`.

Access to this field is RO.

Swap, bits [3:0]

Indicates the implemented Swap instructions in the A32 instruction set.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Swap	Meaning
`0b0000`	None implemented.
`0b0001`	Adds `SWP` and `SWPB`.

All other values are reserved.

In Armv8-A, the only permitted value is `0b0000`.

Access to this field is RO.

Otherwise:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

63	32
UNKNOWN	

31	0
UNKNOWN	

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_ISAR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_ISAR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_ISAR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_ISAR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_ISAR0_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.102 ID_ISAR1_EL1, AArch32 Instruction Set Attribute Register 1

The ID_ISAR1_EL1 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with ID_ISAR0_EL1, ID_ISAR2_EL1, ID_ISAR3_EL1, ID_ISAR4_EL1, and ID_ISAR5_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_ISAR1_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_ISAR1[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_ISAR1_EL1 are UNDEFINED.

Attributes

ID_ISAR1_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:

Bits [63:32]

Reserved, RES0.

Jazelle, bits [31:28]

Indicates the implemented Jazelle extension instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Jazelle	Meaning
0b0000	No support for Jazelle.
0b0001	Adds the BXJ instruction and the J bit in the PSR. This setting might indicate a trivial implementation of the Jazelle extension.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Interwork, bits [27:24]

Indicates the implemented Interworking instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Interwork	Meaning
0b0000	None implemented.
0b0001	Adds the BX instruction, and the T bit in the PSR.
0b0010	As for 0b0001, and adds the BLX instruction. PC loads have BX-like behavior.
0b0011	As for 0b0010, and guarantees that data-processing instructions in the A32 instruction set with the PC as the destination and the S bit clear have BX-like behavior.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0011.

Access to this field is RO.

Immediate, bits [23:20]

Indicates support for the following data-processing instructions with long immediates:

- The MOVT instruction.
- The MOV instruction encodings with zero-extended 16-bit immediates.
- The T32 ADD and SUB instruction encodings with zero-extended 12-bit immediates, and the other ADD, ADR, and SUB encodings cross-referenced by the pseudocode for those encodings.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Immediate	Meaning
0b0000	The specified instructions are not implemented.
0b0001	The specified instructions are implemented.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is RO.

IfThen, bits [19:16]

Indicates the implemented If-Then instructions in the T32 instruction set.

The value of this field is an IMPLEMENTATION DEFINED choice of:

IfThen	Meaning
0b0000	None implemented.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

0b0001	Adds the IT instructions, and the IT bits in the PSRs.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is RO.

Extend, bits [15:12]

Indicates the implemented Extend instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Extend	Meaning
0b0000	No scalar sign-extend or zero-extend instructions are implemented, where scalar instructions means non-Advanced SIMD instructions.
0b0001	Adds the SXTB, SXTH, UXTB, and UXTH instructions.
0b0010	As for 0b0001, and adds the SXTB16, SXTAB, SXTAB16, SXTAH, UXTB16, UXTAB, UXTAB16, and UXTAH instructions.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Access to this field is RO.

Except_AR, bits [11:8]

Indicates the implemented A and R-profile exception-handling instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Except_AR	Meaning
0b0000	None implemented.
0b0001	Adds the SRS and RFE instructions, and the A and R-profile forms of the CPS instruction.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is RO.

Except, bits [7:4]

Indicates the implemented exception-handling instructions in the A32 instruction set.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Except	Meaning
0b0000	Not implemented. This indicates that the User bank and Exception return forms of the LDM and STM instructions are not implemented.
0b0001	Adds the LDM (exception return), LDM (user registers), and STM (user registers) instruction versions.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is RO.

Endian, bits [3:0]

Indicates the implemented Endian instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Endian	Meaning
0b0000	None implemented.
0b0001	Adds the SETEND instruction, and the E bit in the PSRs.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

Access to this field is RO.

Otherwise:

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_ISAR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_ISAR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0010	0b001

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
        ↪then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_ISAR1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
        ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_ISAR1_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_ISAR1_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.103 ID_ISAR2_EL1, AArch32 Instruction Set Attribute Register 2

The ID_ISAR2_EL1 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with ID_ISAR0_EL1, ID_ISAR1_EL1, ID_ISAR3_EL1, ID_ISAR4_EL1, and ID_ISAR5_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_ISAR2_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_ISAR2[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_ISAR2_EL1 are UNDEFINED.

Attributes

ID_ISAR2_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:

Bits [63:32]

Reserved, RES0.

Reversal, bits [31:28]

Indicates the implemented Reversal instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Reversal	Meaning
0b0000	None implemented.
0b0001	Adds the REV, REV16, and REVSH instructions.
0b0010	As for 0b0001, and adds the RBIT instruction.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

PSR_AR, bits [27:24]

Indicates the implemented A and R-profile instructions to manipulate the PSR.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PSR_AR	Meaning
0b0000	None implemented.
0b0001	Adds the MRS and MSR instructions, and the exception return forms of data-processing instructions.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

The exception return forms of the data-processing instructions are:

- In the A32 instruction set, data-processing instructions with the PC as the destination and the S bit set. These instructions might be affected by the WithShifts attribute.
- In the T32 instruction set, the SUBS PC,LR,#N instruction.

Access to this field is RO.

MultU, bits [23:20]

Indicates the implemented advanced unsigned Multiply instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MultU	Meaning
0b0000	None implemented.
0b0001	Adds the UMULL and UMLAL instructions.
0b0010	As for 0b0001, and adds the UMAAL instruction.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Access to this field is RO.

MultS, bits [19:16]

Indicates the implemented advanced signed Multiply instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MultS	Meaning
0b0000	None implemented.
0b0001	Adds the SMULL and SMLAL instructions.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

MultS	Meaning
0b0010	As for 0b0001, and adds the SMLABB, SMLABT, SMLALBB, SMLALBT, SMLALTB, SMLALTT, SMLATB, SMLATT, SMLAWB, SMLAWT, SMULBB, SMULBT, SMULTB, SMULTT, SMULWB, and SMULWT instructions. Also adds the Q bit in the PSRs.
0b0011	As for 0b0010, and adds the SMLAD, SMLADX, SMLALD, SMLALDX, SMLSD, SMLSDX, SMLSLD, SMLSLDX, SMMLA, SMMLAR, SMMLS, SMMLSR, SMMUL, SMMULR, SMUAD, SMUADX, SMUSD, and SMUSDX instructions.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0011.

Access to this field is RO.

Mult, bits [15:12]

Indicates the implemented additional Multiply instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Mult	Meaning
0b0000	No additional instructions implemented. This means only MUL is implemented.
0b0001	Adds the MLA instruction.
0b0010	As for 0b0001, and adds the MLS instruction.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Access to this field is RO.

MultiAccessInt, bits [11:8]

Indicates the support for interruptible multi-access instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MultiAccessInt	Meaning
0b0000	No support. This means the LDM and STM instructions are not interruptible.
0b0001	LDM and STM instructions are restartable.
0b0010	LDM and STM instructions are continuable.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is RO.

MemHint, bits [7:4]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Indicates the implemented Memory Hint instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MemHint	Meaning
`0b0000`	None implemented.
`0b0001`	Adds the PLD instruction.
`0b0010`	Adds the PLD instruction. (`0b0001` and `0b0010` have identical effects.)
`0b0011`	As for `0b0001` (or `0b0010`), and adds the PLI instruction.
`0b0100`	As for `0b0011`, and adds the PLDW instruction.

All other values are reserved.

In Armv8-A, the only permitted value is `0b0100`.

Access to this field is RO.

LoadStore, bits [3:0]

Indicates the implemented additional load/store instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

LoadStore	Meaning
`0b0000`	No additional load/store instructions implemented.
`0b0001`	Adds the LDRD and STRD instructions.
`0b0010`	As for `0b0001`, and adds the Load Acquire (LDAB, LDAH, LDA, LDAEXB, LDAEXH, LDAEX, LDAEXD) and Store Release (STLB, STLH, STL, STLEXB, STLEXH, STLEX, STLEXD) instructions.

All other values are reserved.

In Armv8-A, the only permitted value is `0b0010`.

Access to this field is RO.

Otherwise:

Bits [63:0]

Reserved, UNKNOWN.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing ID_ISAR2_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_ISAR2_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0010	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_ISAR2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_ISAR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_ISAR2_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.104 ID_ISAR3_EL1, AArch32 Instruction Set Attribute Register 3

The ID_ISAR3_EL1 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with ID_ISAR0_EL1, ID_ISAR1_EL1, ID_ISAR2_EL1, ID_ISAR4_EL1, and ID_ISAR5_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_ISAR3_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_ISAR3[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_ISAR3_EL1 are UNDEFINED.

Attributes

ID_ISAR3_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:

Bits [63:32]

Reserved, RES0.

T32EE, bits [31:28]

Indicates the implemented T32EE instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

T32EE	Meaning
0b0000	None implemented.
0b0001	Adds the ENTERX and LEAVEX instructions, and modifies the load behavior to include null checking.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TrueNOP, bits [27:24]

Indicates the implemented true NOP instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TrueNOP	Meaning
0b0000	None implemented. This means there are no NOP instructions that do not have any register dependencies.
0b0001	Adds true NOP instructions in both the T32 and A32 instruction sets. This also permits additional NOP-compatible hints.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is RO.

T32Copy, bits [23:20]

Indicates the support for T32 non flag-setting MOV instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

T32Copy	Meaning
0b0000	Not supported. This means that in the T32 instruction set, encoding T1 of the MOV (register) instruction does not support a copy from a low register to a low register.
0b0001	Adds support for T32 instruction set encoding T1 of the MOV (register) instruction, copying from a low register to a low register.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is RO.

TabBranch, bits [19:16]

Indicates the implemented Table Branch instructions in the T32 instruction set.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TabBranch	Meaning
0b0000	None implemented.
0b0001	Adds the TBB and TBH instructions.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SynchPrim, bits [15:12]

Used in conjunction with ID_ISAR4.SynchPrim_frac to indicate the implemented Synchronization Primitive instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SynchPrim	Meaning
`0b0000`	If SynchPrim_frac == `0b0000`, no Synchronization Primitives implemented.
`0b0001`	If SynchPrim_frac == `0b0000`, adds the LDREX and STREX instructions. If SynchPrim_frac == `0b0011`, also adds the CLREX, LDREXB, STREXB, and STREXH instructions.
`0b0010`	If SynchPrim_frac == `0b0000`, as for [`0b0001`, `0b0011`] and also adds the LDREXD and STREXD instructions.

All other combinations of SynchPrim and SynchPrim_frac are reserved.

In Armv8-A, the only permitted value is `0b0010`.

Access to this field is RO.

SVC, bits [11:8]

Indicates the implemented SVC instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SVC	Meaning
`0b0000`	Not implemented.
`0b0001`	Adds the SVC instruction.

All other values are reserved.

In Armv8-A, the only permitted value is `0b0001`.

Access to this field is RO.

SIMD, bits [7:4]

Indicates the implemented SIMD instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SIMD	Meaning
`0b0000`	None implemented.
`0b0001`	Adds the SSAT and USAT instructions, and the Q bit in the PSRs.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SIMD	Meaning
0b0011	As for 0b0001, and adds the PKHBT, PKHTB, QADD16, QADD8, QASX, QSUB16, QSUB8, QSAX, SADD16, SADD8, SASX, SEL, SHADD16, SHADD8, SHASX, SHSUB16, SHSUB8, SHSAX, SSAT16, SSUB16, SSUB8, SSAX, SXTAB16, SXTB16, UADD16, UADD8, UASX, UHADD16, UHADD8, UHASX, UHSUB16, UHSUB8, UHSAX, UQADD16, UQADD8, UQASX, UQSUB16, UQSUB8, UQSAX, USAD8, USADA8, USAT16, USUB16, USUB8, USAX, UXTAB16, and UXTB16 instructions. Also adds support for the GE[3:0] bits in the PSRs.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0011.

The SIMD field relates only to implemented instructions that perform SIMD operations on the general-purpose registers. In an implementation that supports Advanced SIMD and floating-point instructions, MVFR0, MVFR1, and MVFR2 give information about the implemented Advanced SIMD instructions.

Access to this field is RO.

Saturate, bits [3:0]

Indicates the implemented Saturate instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Saturate	Meaning
0b0000	None implemented. This means no non-Advanced SIMD saturate instructions are implemented.
0b0001	Adds the QADD, QDADD, QDSUB, and QSUB instructions, and the Q bit in the PSRs.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is RO.

Otherwise:

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_ISAR3_EL1

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, ID_ISAR3_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0010	0b011

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_ISAR3_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_ISAR3_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_ISAR3_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.105 ID_ISAR4_EL1, AArch32 Instruction Set Attribute Register 4

The ID_ISAR4_EL1 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with ID_ISAR0_EL1, ID_ISAR1_EL1, ID_ISAR2_EL1, ID_ISAR3_EL1, and ID_ISAR5_EL1.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_ISAR4_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_ISAR4[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_ISAR4_EL1 are UNDEFINED.

Attributes

ID_ISAR4_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:

Bits [63:32]

Reserved, RES0.

SWP_frac, bits [31:28]

Indicates support for the memory system locking the bus for SWP or SWPB instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SWP_frac	Meaning
0b0000	SWP or SWPB instructions not implemented.
0b0001	SWP or SWPB implemented but only in a uniprocessor context. SWP and SWPB do not guarantee whether memory accesses from other Requesters can come between the load memory access and the store memory access of the SWP or SWPB.

All other values are reserved. This field is valid only if ID_ISAR0.Swap is 0b0000.

In Armv8-A, the only permitted value is 0b0000.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Access to this field is RO.

PSR_M, bits [27:24]

Indicates the implemented M-profile instructions to modify the PSRs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PSR_M	Meaning
0b0000	None implemented.
0b0001	Adds the M-profile forms of the CPS, MRS, and MSR instructions.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is RO.

SynchPrim_frac, bits [23:20]

Used in conjunction with ID_ISAR3.SynchPrim to indicate the implemented Synchronization Primitive instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SynchPrim_frac	Meaning
0b0000	If SynchPrim == 0b0000, no Synchronization Primitives implemented. If SynchPrim == 0b0001, adds the LDREX and STREX instructions. If SynchPrim == 0b0010, also adds the CLREX, LDREXB, LDREXH, STREXB, STREXH, LDREXD, and STREXD instructions.
0b0011	If SynchPrim == 0b0001, adds the LDREX, STREX, CLREX, LDREXB, LDREXH, STREXB, and STREXH instructions.

All other combinations of SynchPrim and SynchPrim_frac are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is RO.

Barrier, bits [19:16]

Indicates the implemented Barrier instructions in the T32 and A32 instruction sets.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Barrier	Meaning
0b0000	None implemented. Barrier operations are provided only as System instructions in the (coproc==0b1111) encoding space.
0b0001	Adds the DMB, DSB, and ISB barrier instructions.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

All other values are reserved.

In Armv8-A, the only permitted value is `0b0001`.

Access to this field is RO.

SMC, bits [15:12]

Indicates the implemented SMC instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SMC	Meaning
`0b0000`	None implemented.
`0b0001`	Adds the SMC instruction.

All other values are reserved.

In Armv8-A, the permitted values are:

- If EL3 is implemented and EL1 can use AArch32, the only permitted value is `0b0001`.
- If neither EL3 nor EL2 is implemented, the only permitted value is `0b0000`.

If EL1 cannot use AArch32, this field has the value `0b0000`.

Access to this field is RO.

Writeback, bits [11:8]

Indicates the support for Writeback addressing modes.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Writeback	Meaning
`0b0000`	Basic support. Only the LDM, STM, PUSH, POP, SRS, and RFE instructions support writeback addressing modes. These instructions support all of their writeback addressing modes.
`0b0001`	Adds support for all of the writeback addressing modes.

All other values are reserved.

In Armv8-A, the only permitted value is `0b0001`.

Access to this field is RO.

WithShifts, bits [7:4]

Indicates the support for instructions with shifts.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

WithShifts	Meaning
`0b0000`	Nonzero shifts supported only in MOV and shift instructions.
`0b0001`	Adds support for shifts of loads and stores over the range LSL 0-3.
`0b0011`	As for `0b0001`, and adds support for other constant shift options, both on load/store and other instructions.
`0b0100`	As for `0b0011`, and adds support for register-controlled shift options.

All other values are reserved.

In Armv8-A, the only permitted value is `0b0100`.

Access to this field is RO.

Unpriv, bits [3:0]

Indicates the implemented unprivileged instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Unpriv	Meaning
`0b0000`	None implemented. No T variant instructions are implemented.
`0b0001`	Adds the LDRBT, LDRT, STRBT, and STRT instructions.
`0b0010`	As for `0b0001`, and adds the LDRHT, LDRSBT, LDRSHT, and STRHT instructions.

All other values are reserved.

In Armv8-A, the only permitted value is `0b0010`.

Access to this field is RO.

Otherwise:

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_ISAR4_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_ISAR4_EL1
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0010	0b100

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_ISAR4_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_ISAR4_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_ISAR4_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.106 ID_ISAR5_EL1, AArch32 Instruction Set Attribute Register 5

The ID_ISAR5_EL1 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with ID_ISAR0_EL1, ID_ISAR1_EL1, ID_ISAR2_EL1, ID_ISAR3_EL1, and ID_ISAR4_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_ISAR5_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_ISAR5[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_ISAR5_EL1 are UNDEFINED.

Attributes

ID_ISAR5_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:

Bits [63:32]

Reserved, RES0.

VCMA, bits [31:28]

Indicates AArch32 support for complex number addition and multiplication where numbers are stored in vectors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

VCMA	Meaning
0b0000	The VCMLA and VCADD instructions are not implemented in AArch32.
0b0001	The VCMLA and VCADD instructions are implemented in AArch32.

All other values are reserved.

FEAT_FCMA implements the functionality identified by 0b0001.

From Armv8.3, the value 0b0000 is not permitted.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

RDM, bits [27:24]

Indicates whether the `VQRDMLAH` and `VQRDMLSH` instructions are implemented in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RDM	Meaning
`0b0000`	No `VQRDMLAH` and `VQRDMLSH` instructions implemented.
`0b0001`	`VQRDMLAH` and `VQRDMLSH` instructions implemented.

All other values are reserved.

FEAT_RDM implements the functionality identified by the value `0b0001`.

From Armv8.1, the value `0b0000` is not permitted.

Access to this field is RO.

Bits [23:20]

Reserved, RES0.

CRC32, bits [19:16]

Indicates whether the CRC32 instructions `CRC32B`, `CRC32H`, `CRC32W`, `CRC32CB`, `CRC32CH`, and `CRC32CW` are implemented in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CRC32	Meaning
`0b0000`	CRC32 instructions are not implemented.
`0b0001`	The specified instructions are implemented.

All other values are reserved.

FEAT_CRC32 implements the functionality identified by the value `0b0001`.

In Armv8.0, the permitted values are `0b0000` and `0b0001`.

From Armv8.1, the value `0b0000` is not permitted.

Access to this field is RO.

SHA2, bits [15:12]

Indicates whether the SHA2 instructions `SHA256H`, `SHA256H2`, `SHA256SU0`, and `SHA256SU1` are implemented in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SHA2	Meaning
`0b0000`	No SHA2 instructions implemented.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SHA2	Meaning
0b0001	The specified instructions are implemented.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

Access to this field is RO.

SHA1, bits [11:8]

Indicates whether the SHA1 instructions SHA1C, SHA1P, SHA1M, SHA1H, SHA1SU0, and SHA1SU1 are implemented in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SHA1	Meaning
0b0000	No SHA1 instructions implemented.
0b0001	The specified instructions are implemented.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

Access to this field is RO.

AES, bits [7:4]

Indicates whether the AES instructions are implemented in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AES	Meaning
0b0000	No AES instructions implemented.
0b0001	AESE, AESD, AESMC, and AESIMC implemented.
0b0010	As for 0b0001, plus VMULL (polynomial) instructions operating on 64-bit data quantities.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0010.

Access to this field is RO.

SEVL, bits [3:0]

Indicates whether the SEVL instruction is implemented in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SEVL	Meaning
0b0000	SEVL is implemented as a NOP.
0b0001	SEVL is implemented as Send Event Local.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is RO.

Otherwise:

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_ISAR5_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_ISAR5_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0010	0b101

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        X[t, 64] = ID_ISAR5_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_ISAR5_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_ISAR5_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.107 ID_ISAR6_EL1, AArch32 Instruction Set Attribute Register 6

The ID_ISAR6_EL1 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with ID_ISAR0_EL1, ID_ISAR1_EL1, ID_ISAR2_EL1, ID_ISAR3_EL1, ID_ISAR4_EL1 and ID_ISAR5_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

AArch64 System register ID_ISAR6_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_ISAR6[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_ISAR6_EL1 are UNDEFINED.

Attributes

ID_ISAR6_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:

Bits [63:32]

Reserved, RES0.

CLRBHB, bits [31:28]

Indicates support for the CLRBHB instruction in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CLRBHB	Meaning
0b0000	CLRBHB instruction is not implemented.
0b0001	CLRBHB instruction is implemented.

All other values are reserved.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FEAT_CLRBHB implements the functionality identified by `0b0001`.

From Armv8.9, the value `0b0000` is not permitted.

Access to this field is RO.

I8MM, bits [27:24]

Indicates support for the following Advanced SIMD Int8 matrix multiplication instructions `VSMMLA`, `VSUDOT`, `VUMMLA`, `VUSMMLA`, and `VUSDOT` in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

I8MM	Meaning
`0b0000`	Int8 matrix multiplication instructions are not implemented.
`0b0001`	The specified instructions are implemented.

All other values are reserved.

FEAT_AA32I8MM implements the functionality identified by `0b0001`.

Access to this field is RO.

BF16, bits [23:20]

Indicates support for the following Advanced SIMD and floating-point BFloat16 instructions `VCVT`, `VCVTB`, `VCVTT`, `VDOT`, `VFMAB`, `VFMAT`, and `VMMLA` instructions with BF16 operand or result types in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BF16	Meaning
`0b0000`	BFloat16 instructions are not implemented.
`0b0001`	The specified instructions are implemented.

All other values are reserved.

FEAT_AA32BF16 implements the functionality identified by `0b0001`.

Access to this field is RO.

SPECRES, bits [19:16]

Indicates support for prediction invalidation instructions in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SPECRES	Meaning
`0b0000`	Prediction invalidation instructions are not implemented.
`0b0001`	CFPRCTX, DVPRCTX, and CPPRCTX instructions are implemented.
`0b0010`	As `0b0001`, and COSPRCTX instruction is implemented.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

All other values are reserved.

FEAT_SPECRES implements the functionality identified by `0b0001`.

FEAT_SPECRES2 implements the functionality identified by `0b0010`.

From Armv8.5, the value `0b0000` is not permitted.

From Armv8.9, the value `0b0001` is not permitted.

Access to this field is RO.

SB, bits [15:12]

Indicates support for the `SB` instruction in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SB	Meaning
`0b0000`	`SB` instruction is not implemented.
`0b0001`	`SB` instruction is implemented.

All other values are reserved.

FEAT_SB implements the functionality identified by `0b0001`.

From Armv8.5, value `0b0000` is not permitted.

Access to this field is RO.

FHM, bits [11:8]

Indicates support for the following Advanced SIMD and floating-point `VFMAL` and `VFMSL` instructions in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FHM	Meaning
`0b0000`	The specified instructions are not implemented.
`0b0001`	The specified instructions are implemented.

All other values are reserved.

FEAT_FHM implements the functionality identified by `0b0001`.

From Armv8.2, the permitted values are `0b0000` and `0b0001`.

Access to this field is RO.

DP, bits [7:4]

Indicates support for dot product instructions in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

DP	Meaning
0b0000	Dot product instructions are not implemented.
0b0001	VUDOT and VSDOT instructions are implemented.

All other values are reserved.

FEAT_DotProd implements the functionality identified by 0b0001.

In Armv8.2, the permitted values are 0b0000 and 0b0001.

From Armv8.4, the only permitted value is 0b0001.

Access to this field is RO.

JSCVT, bits [3:0]

Indicates support for the VJCVT instruction in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

JSCVT	Meaning
0b0000	The VJCVT instruction is not implemented.
0b0001	The VJCVT instruction is implemented.

All other values are reserved.

FEAT_JSCVT implements the functionality identified by 0b0001.

In Armv8.0, Armv8.1, and Armv8.2, the only permitted value is 0b0000.

From Armv8.3, if Advanced SIMD or Floating-point is implemented, the value 0b0000 is not implemented.

From Armv8.3, if Advanced SIMD or Floating-point is not implemented, the only permitted value is 0b0000.

Access to this field is RO.

Otherwise:

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_ISAR6_EL1

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, ID_ISAR6_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0010	0b111

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_ISAR6_EL1) || boolean
     ↪IMPLEMENTATION_DEFINED "ID_ISAR6_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_ISAR6_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_ISAR6_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_ISAR6_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.108 ID_MMFR0_EL1, AArch32 Memory Model Feature Register 0

The ID_MMFR0_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_MMFR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_MMFR0[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_MMFR0_EL1 are UNDEFINED.

Attributes

ID_MMFR0_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:

Bits [63:32]

Reserved, RES0.

InnerShr, bits [31:28]

Innermost Shareability. Indicates the innermost shareability domain implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

InnerShr	Meaning
0b0000	Implemented as Non-cacheable.
0b0001	Implemented with hardware coherency support.
0b1111	Shareability ignored.

All other values are reserved.

From Armv8 the permitted values are 0b0000, 0b0001, and 0b1111.

This field is valid only if the implementation supports two levels of shareability, as indicated by ID_MMFR0_EL1.ShareLvl having the value 0b0001.

When ID_MMFR0_EL1.ShareLvl is zero, this field is UNKNOWN.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FCSE, bits [27:24]

Indicates whether the implementation includes the FCSE.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FCSE	Meaning
0b0000	Not supported.
0b0001	Support for FCSE.

All other values are reserved.

From Armv8 the only permitted value is 0b0000.

Access to this field is RO.

AuxReg, bits [23:20]

Auxiliary Registers. Indicates support for Auxiliary registers.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AuxReg	Meaning
0b0000	None supported.
0b0001	Support for Auxiliary Control Register only.
0b0010	Support for Auxiliary Fault Status Registers (AIFSR and ADFSR) and Auxiliary Control Register.

All other values are reserved.

From Armv8 the only permitted value is 0b0010.

Note

Accesses to unimplemented Auxiliary registers are UNDEFINED.

Access to this field is RO.

TCM, bits [19:16]

Indicates support for TCMs and associated DMAs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TCM	Meaning
0b0000	Not supported.
0b0001	Support is IMPLEMENTATION DEFINED.
0b0010	Support for TCM only, Armv6 implementation.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

0b0011	Support for TCM and DMA, Armv6 implementation.

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

Access to this field is RO.

ShareLvl, bits [15:12]

Shareability Levels. Indicates the number of shareability levels implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ShareLvl	Meaning
0b0000	One level of shareability implemented.
0b0001	Two levels of shareability implemented.

All other values are reserved.

From Armv8 the only permitted value is 0b0001.

Access to this field is RO.

OuterShr, bits [11:8]

Outermost Shareability. Indicates the outermost shareability domain implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

OuterShr	Meaning
0b0000	Implemented as Non-cacheable.
0b0001	Implemented with hardware coherency support.
0b1111	Shareability ignored.

All other values are reserved.

From Armv8 the permitted values are 0b0000, 0b0001, and 0b1111.

Access to this field is RO.

PMSA, bits [7:4]

Indicates support for a PMSA.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

PMSA	Meaning
0b0000	Not supported.
0b0001	Support for IMPLEMENTATION DEFINED PMSA.
0b0010	Support for PMSAv6, with a Cache Type Register implemented.
0b0011	Support for PMSAv7, with support for memory subsections. Armv7-R profile.

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

Access to this field is RO.

VMSA, bits [3:0]

Indicates support for a VMSA.

The value of this field is an IMPLEMENTATION DEFINED choice of:

VMSA	Meaning
0b0000	Not supported.
0b0001	Support for IMPLEMENTATION DEFINED VMSA.
0b0010	Support for VMSAv6, with Cache and TLB Type Registers implemented.
0b0011	Support for VMSAv7, with support for remapping and the Access flag. Armv7-A profile.
0b0100	As for 0b0011, and adds support for the PXN bit in the Short-descriptor translation table format descriptors.
0b0101	As for 0b0100, and adds support for the Long-descriptor translation table format.

All other values are reserved.

In Armv8-A the only permitted value is 0b0101.

Access to this field is RO.

Otherwise:

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_MMFR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, ID_MMFR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0001	0b100

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_MMFR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_MMFR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_MMFR0_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.109 ID_MMFR1_EL1, AArch32 Memory Model Feature Register 1

The ID_MMFR1_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_MMFR1_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_MMFR1[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_MMFR1_EL1 are UNDEFINED.

Attributes

ID_MMFR1_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:

Bits [63:32]

Reserved, RES0.

BPred, bits [31:28]

Branch Predictor. Indicates branch predictor management requirements.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BPred	Meaning
0b0000	No branch predictor, or no MMU present. Implies a fixed MPU configuration.
0b0001	Branch predictor requires flushing on: • Enabling or disabling a stage of address translation. • Writing new data to instruction locations. • Writing new mappings to the translation tables. • Changes to the TTBR0, TTBR1, or TTBCR registers. • Changes to the ContextID or ASID, or to the FCSE ProcessID if this is supported.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

BPred	Meaning
0b0010	Branch predictor requires flushing on: • Enabling or disabling a stage of address translation. • Writing new data to instruction locations. • Writing new mappings to the translation tables. • Any change to the TTBR0, TTBR1, or TTBCR registers without a change to the corresponding ContextID or ASID, or FCSE ProcessID if this is supported.
0b0011	Branch predictor requires flushing only on writing new data to instruction locations.
0b0100	For execution correctness, branch predictor requires no flushing at any time.

All other values are reserved.

In Armv8-A, the permitted values are 0b0010, 0b0011, and 0b0100. For values other than 0b0000 and 0b0100 the Arm Architecture Reference Manual, or the product documentation, might give more information about the required maintenance.

Access to this field is RO.

L1TstCln, bits [27:24]

Level 1 cache Test and Clean. Indicates the supported Level 1 data cache test and clean operations, for Harvard or unified cache implementations.

The value of this field is an IMPLEMENTATION DEFINED choice of:

L1TstCln	Meaning
0b0000	None supported.
0b0001	Supported Level 1 data cache test and clean operations are: • Test and clean data cache.
0b0010	As for 0b0001, and adds: • Test, clean, and invalidate data cache.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is RO.

L1Uni, bits [23:20]

Level 1 Unified cache. Indicates the supported entire Level 1 cache maintenance operations for a unified cache implementation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

L1Uni	Meaning
0b0000	None supported.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

L1Uni	Meaning
0b0001	Supported entire Level 1 cache operations are: • Invalidate cache, including branch predictor if appropriate. • Invalidate branch predictor, if appropriate.
0b0010	As for 0b0001, and adds: • Clean cache, using a recursive model that uses the cache dirty status bit. • Clean and invalidate cache, using a recursive model that uses the cache dirty status bit.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is RO.

L1Hvd, bits [19:16]

Level 1 Harvard cache. Indicates the supported entire Level 1 cache maintenance operations for a Harvard cache implementation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

L1Hvd	Meaning
0b0000	None supported.
0b0001	Supported entire Level 1 cache operations are: • Invalidate instruction cache, including branch predictor if appropriate. • Invalidate branch predictor, if appropriate.
0b0010	As for 0b0001, and adds: • Invalidate data cache. • Invalidate data cache and instruction cache, including branch predictor if appropriate.
0b0011	As for 0b0010, and adds: • Clean data cache, using a recursive model that uses the cache dirty status bit. • Clean and invalidate data cache, using a recursive model that uses the cache dirty status bit.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is RO.

L1UniSW, bits [15:12]

Level 1 Unified cache by Set/Way. Indicates the supported Level 1 cache line maintenance operations by set/way, for a unified cache implementation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

L1UniSW	Meaning
0b0000	None supported.
0b0001	Supported Level 1 unified cache line maintenance operations by set/way are: • Clean cache line by set/way.
0b0010	As for 0b0001, and adds: • Clean and invalidate cache line by set/way.
0b0011	As for 0b0010, and adds: • Invalidate cache line by set/way.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is RO.

L1HvdSW, bits [11:8]

Level 1 Harvard cache by Set/Way. Indicates the supported Level 1 cache line maintenance operations by set/way, for a Harvard cache implementation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

L1HvdSW	Meaning
0b0000	None supported.
0b0001	Supported Level 1 Harvard cache line maintenance operations by set/way are: • Clean data cache line by set/way. • Clean and invalidate data cache line by set/way.
0b0010	As for 0b0001, and adds: • Invalidate data cache line by set/way.
0b0011	As for 0b0010, and adds: • Invalidate instruction cache line by set/way.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is RO.

L1UniVA, bits [7:4]

Level 1 Unified cache by Virtual Address. Indicates the supported Level 1 cache line maintenance operations by VA, for a unified cache implementation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

L1UniVA	Meaning
0b0000	None supported.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

L1UniVA	Meaning
0b0001	Supported Level 1 unified cache line maintenance operations by VA are: • Clean cache line by VA. • Invalidate cache line by VA. • Clean and invalidate cache line by VA.
0b0010	As for 0b0001, and adds: • Invalidate branch predictor by VA, if branch predictor is implemented.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is RO.

L1HvdVA, bits [3:0]

Level 1 Harvard cache by Virtual Address. Indicates the supported Level 1 cache line maintenance operations by VA, for a Harvard cache implementation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

L1HvdVA	Meaning
0b0000	None supported.
0b0001	Supported Level 1 Harvard cache line maintenance operations by VA are: • Clean data cache line by VA. • Invalidate data cache line by VA. • Clean and invalidate data cache line by VA. • Clean instruction cache line by VA.
0b0010	As for 0b0001, and adds: • Invalidate branch predictor by VA, if branch predictor is implemented.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Access to this field is RO.

Otherwise:

Bits [63:0]

Reserved, UNKNOWN.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing ID_MMFR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_MMFR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0001	0b101

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_MMFR1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_MMFR1_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_MMFR1_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.110 ID_MMFR2_EL1, AArch32 Memory Model Feature Register 2

The ID_MMFR2_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_MMFR2_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_MMFR2[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_MMFR2_EL1 are UNDEFINED.

Attributes

ID_MMFR2_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:

Bits [63:32]

Reserved, RES0.

HWAccFlg, bits [31:28]

Hardware Access Flag. In earlier versions of the Arm Architecture, this field indicates support for a Hardware Access flag, as part of the VMSAv7 implementation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HWAccFlg	Meaning
0b0000	Not supported.
0b0001	Support for VMSAv7 Access flag, updated in hardware.

All other values are reserved.

From Armv8.0, 0b0001 is not permitted.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

WFIStall, bits [27:24]

Wait For Interrupt Stall. Indicates the support for Wait For Interrupt (WFI) stalling.

The value of this field is an IMPLEMENTATION DEFINED choice of:

WFIStall	Meaning
0b0000	Not supported.
0b0001	Support for WFI stalling.

All other values are reserved.

Access to this field is RO.

MemBarr, bits [23:20]

Memory Barrier. Indicates the supported memory barrier System instructions in the (coproc==0b1111) encoding space:

The value of this field is an IMPLEMENTATION DEFINED choice of:

MemBarr	Meaning
0b0000	None supported.
0b0001	Supported memory barrier System instructions are: • Data Synchronization Barrier (DSB).
0b0010	As for 0b0001, and adds: • Instruction Synchronization Barrier (ISB). • Data Memory Barrier (DMB).

All other values are reserved.

From Armv8.0, the values 0b000 and 0b0001 are not permitted.

Arm deprecates the use of these operations. ID_ISAR4.Barrier_instrs indicates the level of support for the preferred barrier instructions.

Access to this field is RO.

UniTLB, bits [19:16]

Unified TLB. Indicates the supported TLB maintenance operations, for a unified TLB implementation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

UniTLB	Meaning
0b0000	Not supported.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

UniTLB	Meaning
0b0001	Supported unified TLB maintenance operations are: • Invalidate all entries in the TLB. • Invalidate TLB entry by VA.
0b0010	As for 0b0001, and adds: • Invalidate TLB entries by ASID match.
0b0011	As for 0b0010, and adds: • Invalidate instruction TLB and data TLB entries by VA All ASID. This is a shared unified TLB operation.
0b0100	As for 0b0011, and adds: • Invalidate Hyp mode unified TLB entry by VA. • Invalidate entire Non-secure PL1&0 unified TLB. • Invalidate entire Hyp mode unified TLB.
0b0101	As for 0b0100, and adds the following operations: TLBIMVALIS, TLBIMVAALIS, TLBIMVALHIS, TLBIMVAL, TLBIMVAAL, TLBIMVALH.
0b0110	As for 0b0101, and adds the following operations: TLBIIPAS2IS, TLBIIPAS2LIS, TLBIIPAS2, TLBIIPAS2L.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0110.

Access to this field is RO.

HvdTLB, bits [15:12]

If the Unified TLB field (UniTLB, bits [19:16]) is not 0000, then the meaning of this field is IMPLEMENTATION DEFINED. Arm deprecates the use of this field by software.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

L1HvdRng, bits [11:8]

Level 1 Harvard cache Range. Indicates the supported Level 1 cache maintenance range operations, for a Harvard cache implementation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

L1HvdRng	Meaning
0b0000	Not supported.
0b0001	Supported Level 1 Harvard cache maintenance range operations are: • Invalidate data cache range by VA. • Invalidate instruction cache range by VA. • Clean data cache range by VA. • Clean and invalidate data cache range by VA.

All other values are reserved.

From Armv8.0, the value 0b0001 is not permitted.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

L1HvdBG, bits [7:4]

Level 1 Harvard cache Background fetch. Indicates the supported Level 1 cache background fetch operations, for a Harvard cache implementation. When supported, background fetch operations are non-blocking operations.

The value of this field is an IMPLEMENTATION DEFINED choice of:

L1HvdBG	Meaning
`0b0000`	Not supported.
`0b0001`	Supported Level 1 Harvard cache background fetch operations are: • Fetch instruction cache range by VA. • Fetch data cache range by VA.

All other values are reserved.

From Armv8.0, the value `0b0001` is not permitted.

Access to this field is RO.

L1HvdFG, bits [3:0]

Level 1 Harvard cache Foreground fetch. Indicates the supported Level 1 cache foreground fetch operations, for a Harvard cache implementation. When supported, foreground fetch operations are blocking operations.

The value of this field is an IMPLEMENTATION DEFINED choice of:

L1HvdFG	Meaning
`0b0000`	Not supported.
`0b0001`	Supported Level 1 Harvard cache foreground fetch operations are: • Fetch instruction cache range by VA. • Fetch data cache range by VA.

All other values are reserved.

From Armv8.0, the value `0b0001` is not permitted.

Access to this field is RO.

Otherwise:

Bits [63:0]

Reserved, UNKNOWN.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing ID_MMFR2_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_MMFR2_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0001	0b110

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_MMFR2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_MMFR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_MMFR2_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.111 ID_MMFR3_EL1, AArch32 Memory Model Feature Register 3

The ID_MMFR3_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_MMFR3_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_MMFR3[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_MMFR3_EL1 are UNDEFINED.

Attributes

ID_MMFR3_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:

Bits [63:32]

Reserved, RES0.

Supersec, bits [31:28]

Supersections. On a VMSA implementation, indicates whether Supersections are supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Supersec	Meaning
0b0000	Supersections supported.
0b1111	Supersections not supported.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b1111.

Access to this field is RO.

CMemSz, bits [27:24]

Cached Memory Size. Indicates the physical memory size supported by the caches.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The value of this field is an IMPLEMENTATION DEFINED choice of:

CMemSz	Meaning
0b0000	4GB, corresponding to a 32-bit physical address range.
0b0001	64GB, corresponding to a 36-bit physical address range.
0b0010	1TB or more, corresponding to a 40-bit or larger physical address range.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000, 0b0001, and 0b0010.

Access to this field is RO.

CohWalk, bits [23:20]

Coherent Walk. Indicates whether Translation table updates require a clean to the Point of Unification.:

The value of this field is an IMPLEMENTATION DEFINED choice of:

CohWalk	Meaning
0b0000	Updates to the translation tables require a clean to the Point of Unification to ensure visibility by subsequent translation table walks.
0b0001	Updates to the translation tables do not require a clean to the Point of Unification to ensure visibility by subsequent translation table walks.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is RO.

PAN, bits [19:16]

Privileged Access Never. Indicates support for the PAN bit in CPSR, SPSR, and DSPSR in AArch32 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PAN	Meaning
0b0000	PAN not supported.
0b0001	PAN supported.
0b0010	PAN supported and ATS1CPRP and ATS1CPWP instructions supported.

All other values are reserved.

FEAT_PAN implements the functionality identified by the value 0b0001.

FEAT_PAN2 implements the functionality added by the value 0b0010.

From Armv8.1, the value 0b0000 is not permitted.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

From Armv8.2, the value `0b0001` is not permitted.

Access to this field is RO.

MaintBcst, bits [15:12]

Maintenance Broadcast. Indicates whether Cache, TLB, and branch predictor operations are broadcast.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MaintBcst	Meaning
`0b0000`	Cache, TLB, and branch predictor operations only affect local structures.
`0b0001`	Cache and branch predictor operations affect structures according to shareability and defined behavior of instructions. TLB operations only affect local structures.
`0b0010`	Cache, TLB, and branch predictor operations affect structures according to shareability and defined behavior of instructions.

All other values are reserved.

In Armv8-A, the only permitted value is `0b0010`.

Access to this field is RO.

BPMaint, bits [11:8]

Branch Predictor Maintenance. Indicates the supported branch predictor maintenance operations in an implementation with hierarchical cache maintenance operations.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BPMaint	Meaning
`0b0000`	None supported.
`0b0001`	Supported branch predictor maintenance operations are: • Invalidate all branch predictors.
`0b0010`	As for `0b0001`, and adds: • Invalidate branch predictors by VA.

All other values are reserved.

In Armv8-A, the only permitted value is `0b0010`.

Access to this field is RO.

CMaintSW, bits [7:4]

Cache Maintenance by Set/Way. Indicates the supported cache maintenance operations by set/way, in an implementation with hierarchical caches.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CMaintSW	Meaning
`0b0000`	None supported.
`0b0001`	Supported hierarchical cache maintenance instructions by set/way are: • Invalidate data cache by set/way. • Clean data cache by set/way. • Clean and invalidate data cache by set/way.

All other values are reserved.

In Armv8-A, the only permitted value is `0b0001`.

In a unified cache implementation, the data cache maintenance operations apply to the unified caches.

Access to this field is RO.

CMaintVA, bits [3:0]

Cache Maintenance by Virtual Address. Indicates the supported cache maintenance operations by VA, in an implementation with hierarchical caches.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CMaintVA	Meaning
`0b0000`	None supported.
`0b0001`	Supported hierarchical cache maintenance operations by VA are: • Invalidate data cache by VA. • Clean data cache by VA. • Clean and invalidate data cache by VA. • Invalidate instruction cache by VA. • Invalidate all instruction cache entries.

All other values are reserved.

In Armv8-A, the only permitted value is `0b0001`.

In a unified cache implementation, data cache maintenance operations apply to the unified caches, and the instruction cache maintenance instructions are not implemented.

Access to this field is RO.

Otherwise:

Bits [63:0]

Reserved, UNKNOWN.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing ID_MMFR3_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_MMFR3_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0001	0b111

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_MMFR3_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_MMFR3_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_MMFR3_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.112 ID_MMFR4_EL1, AArch32 Memory Model Feature Register 4

The ID_MMFR4_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

AArch64 System register ID_MMFR4_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_MMFR4[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_MMFR4_EL1 are UNDEFINED.

Attributes

ID_MMFR4_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:

Bits [63:32]

Reserved, RES0.

EVT, bits [31:28]

Enhanced Virtualization Traps. If EL2 is implemented, indicates support for the HCR2.{TTLBIS, TOCU, TICAB, TID4} traps.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EVT	Meaning
0b0000	HCR2.{TTLBIS, TOCU, TICAB, TID4} traps are not supported.
0b0001	HCR2.{TOCU, TICAB, TID4} traps are supported. HCR2.TTLBIS trap is not supported.
0b0010	HCR2.{TTLBIS, TOCU, TICAB, TID4} traps are supported.

All other values are reserved.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FEAT_EVT implements the functionality identified by the values `0b0001` and `0b0010`.

If EL2 is not implemented or does not support AArch32, the only permitted value is `0b0000`.

From Armv8.5, if EL2 is supported and supports AArch32, the value `0b0001` is not permitted.

Access to this field is RO.

CCIDX, bits [27:24]

Support for use of the revised CCSIDR format and the presence of the CCSIDR2 is indicated.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CCIDX	Meaning
`0b0000`	32-bit format implemented for all levels of the CCSIDR, and the CCSIDR2 register is not implemented.
`0b0001`	64-bit format implemented for all levels of the CCSIDR, and the CCSIDR2 register is implemented.

All other values are reserved.

FEAT_CCIDX implements the functionality identified by `0b0001`.

From Armv8.3, the permitted values are `0b0000` and `0b0001`.

Access to this field is RO.

LSM, bits [23:20]

Indicates support for LSMAOE and nTLSMD bits in HSCTLR and SCTLR.

The value of this field is an IMPLEMENTATION DEFINED choice of:

LSM	Meaning
`0b0000`	LSMAOE and nTLSMD bits not supported.
`0b0001`	LSMAOE and nTLSMD bits supported.

All other values are reserved.

FEAT_LSMAOC implements the functionality identified by the value `0b0001`.

From Armv8.2, the permitted values are `0b0000` and `0b0001`.

Access to this field is RO.

HPDS, bits [19:16]

Hierarchical permission disables bits in translation tables.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

HPDS	Meaning
0b0000	Disabling of hierarchical controls not supported.
0b0001	Supports disabling of hierarchical controls using the TTBCR2.HPD0, TTBCR2.HPD1, and HTCR.HPD bits.
0b0010	As for value 0b0001, and adds possible hardware allocation of bits[62:59] of the Translation table descriptors from the final lookup level for IMPLEMENTATION DEFINED use.

All other values are reserved.

FEAT_AA32HPD implements the functionality identified by the value 0b0001.

FEAT_HPDS2 implements the functionality added by the value 0b0010.

Note

The value 0b0000 implies that the encoding for TTBCR2 is UNDEFINED.

Access to this field is RO.

CnP, bits [15:12]

Common not Private translations.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CnP	Meaning
0b0000	Common not Private translations not supported.
0b0001	Common not Private translations supported.

All other values are reserved.

FEAT_TTCNP implements the functionality identified by the value 0b0001.

From Armv8.2, the value 0b0000 is not permitted.

Access to this field is RO.

XNX, bits [11:8]

Support for execute-never control distinction by Exception level at stage 2.

The value of this field is an IMPLEMENTATION DEFINED choice of:

XNX	Meaning
0b0000	Distinction between EL0 and EL1 execute-never control at stage 2 not supported.
0b0001	Distinction between EL0 and EL1 execute-never control at stage 2 supported.

All other values are reserved.

FEAT_XNX implements the functionality identified by the value 0b0001.

When FEAT_XNX is implemented:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- If all of the following conditions are true, it is IMPLEMENTATION DEFINED whether the value of ID_MMFR4_EL1.XNX is `0b0000` or `0b0001`:
  - ID_AA64MMFR1_EL1.XNX ==1.
  - EL2 cannot use AArch32.
  - EL1 can use AArch32.
- If EL2 can use AArch32 then the value `0b0000` is not permitted.

Access to this field is RO.

AC2, bits [7:4]

Indicates the extension of the ACTLR and HACTLR registers using ACTLR2 and HACTLR2.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AC2	Meaning
`0b0000`	ACTLR2 and HACTLR2 are not implemented.
`0b0001`	ACTLR2 and HACTLR2 are implemented.

All other values are reserved.

In Armv8.0 and Armv8.1 the permitted values are `0b0000` and `0b0001`.

From Armv8.2, the only permitted value is `0b0001`.

Access to this field is RO.

SpecSEI, bits [3:0]

When FEAT_RAS is implemented:

Describes whether the PE can generate SError exceptions from speculative reads of memory, including speculative instruction fetches.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SpecSEI	Meaning
`0b0000`	The PE never generates an SError exception due to an External abort on a speculative read.
`0b0001`	The PE might generate an SError exception due to an External abort on a speculative read.

All other values are reserved.

FEAT_SpecSEI implements the functionality identified by the value `0b0001`.

Access to this field is RO.

Otherwise:

Reserved, RES0.

Otherwise:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

63 … 32	31 … 0
UNKNOWN	UNKNOWN

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_MMFR4_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_MMFR4_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0010	0b110

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
         ↪then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_MMFR4_EL1) || boolean
         ↪IMPLEMENTATION_DEFINED "ID_MMFR4_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_MMFR4_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
         ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_MMFR4_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_MMFR4_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.113 ID_MMFR5_EL1, AArch32 Memory Model Feature Register 5

The ID_MMFR5_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

AArch64 System register ID_MMFR5_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_MMFR5[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_MMFR5_EL1 are UNDEFINED.

Attributes

ID_MMFR5_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:

Bits [63:8]

Reserved, RES0.

nTLBPA, bits [7:4]

Indicates support for intermediate caching of translation table walks.

The value of this field is an IMPLEMENTATION DEFINED choice of:

nTLBPA	Meaning
0b0000	The intermediate caching of translation table walks might include non-coherent physical translation caches.
0b0001	The intermediate caching of translation table walks does not include non-coherent physical translation caches.

Non-coherent physical translation caches are non-coherent caches of previous valid translation table entries since the last completed relevant TLBI applicable to the PE, where either:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- The caching is indexed by the physical address of the location holding the translation table entry.
- The caching is used for stage 1 translations and is indexed by the intermediate physical address of the location holding the translation table entry.

All other values are reserved.

FEAT_nTLBPA implements the functionality identified by the value `0b0001`.

From Armv8.0, the permitted values are `0b0000` and `0b0001`.

Access to this field is RO.

ETS, bits [3:0]

Indicates support for Enhanced Translation Synchronization.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ETS	Meaning
`0b0000`	Enhanced Translation Synchronization is not supported.
`0b0001`	Enhanced Translation Synchronization is not supported.
`0b0010`	FEAT_ETS2 is implemented.
`0b0011`	FEAT_ETS3 is implemented.

All other values are reserved.

FEAT_ETS2 implements the functionality identified by the value `0b0010`.

FEAT_ETS3 implements the functionality identified by the value `0b0011`.

From Armv8.8, the values `0b0000` and `0b0001` are not permitted.

From Armv9.5, the value `0b0010` is not permitted.

Access to this field is RO.

Otherwise:

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_MMFR5_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_MMFR5_EL1
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0011	0b110

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_MMFR5_EL1) || boolean
     ↪IMPLEMENTATION_DEFINED "ID_MMFR5_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_MMFR5_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_MMFR5_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_MMFR5_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.114 ID_PFR0_EL1, AArch32 Processor Feature Register 0

The ID_PFR0_EL1 characteristics are:

Purpose

Gives top-level information about the instruction sets supported by the PE in AArch32 state.

Must be interpreted with ID_PFR1_EL1.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_PFR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_PFR0[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_PFR0_EL1 are UNDEFINED.

Attributes

ID_PFR0_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:

Bits [63:32]

Reserved, RES0.

RAS, bits [31:28]

RAS Extension version.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RAS	Meaning
0b0000	No RAS Extension.
0b0001	Support for the Reliability, Availability, and Serviceability Extension is implemented. The ESB instruction and the Error synchronization event are supported.
0b0010	As 0b0001, and adds support for additional ERXMISC<m> System registers. Error records accessed through System registers conform to RAS System Architecture v1.1, which includes simplifications to ERR<n>STATUS and support for the optional RAS Timestamp Extension.
0b0011	As 0b0010, and requires that error records accessed through System registers conform to either RAS System Architecture v1.1 or RAS System Architecture v2.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

All other values are reserved.

FEAT_RAS implements the functionality identified by the value `0b0001`.

FEAT_RASv1p1 implements the functionality identified by the value `0b0010`.

FEAT_RASv2 implements the functionality identified by the value `0b0011`.

In Armv8.0 and Armv8.1, the permitted values are `0b0000` and `0b0001`.

From Armv8.2, the value `0b0000` is not permitted.

From Armv8.4, if FEAT_DoubleFault is implemented or ERRIDR_EL1.NUM is nonzero, the value `0b0001` is not permitted.

Note

When the value of this field is `0b0001`, ID_PFR2_EL1.RAS_frac indicates whether FEAT_RASv1p1 is implemented.

From Armv8.9, if ERRIDR_EL1.NUM is nonzero, the value `0b0010` is not permitted.

Access to this field is RO.

DIT, bits [27:24]

Data Independent Timing.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DIT	Meaning
`0b0000`	AArch32 does not guarantee constant execution time of any instructions.
`0b0001`	AArch32 provides the PSTATE.DIT mechanism to guarantee constant execution time of certain instructions.

All other values are reserved.

FEAT_DIT implements the functionality identified by the value `0b0001`.

From Armv8.4, the only permitted value is `0b0001`.

Access to this field is RO.

AMU, bits [23:20]

Indicates support for Activity Monitors Extension.

The value of this field is an IMPLEMENTATION DEFINED choice of:

AMU	Meaning
`0b0000`	Activity Monitors Extension is not implemented.
`0b0001`	FEAT_AMUv1 is implemented.
`0b0010`	FEAT_AMUv1p1 is implemented. As `0b0001` and adds support for virtualization of the activity monitor event counters.

All other values are reserved.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FEAT_AMUv1 implements the functionality identified by the value `0b0001`.

FEAT_AMUv1p1 implements the functionality identified by the value `0b0010`.

In Armv8.0, the only permitted value is `0b0000`.

In Armv8.4, the permitted values are `0b0000` and `0b0001`.

From Armv8.6, the permitted values are `0b0000`, `0b0001`, and `0b0010`.

Access to this field is RO.

CSV2, bits [19:16]

Speculative use of out of context branch targets.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CSV2	Meaning
`0b0000`	The implementation does not disclose whether FEAT_CSV2 is implemented.
`0b0001`	FEAT_CSV2 is implemented, but FEAT_CSV2_1p1 is not implemented.
`0b0010`	FEAT_CSV2_1p1 is implemented.

All other values are reserved.

FEAT_CSV2 implements the functionality identified by the value `0b0001`.

FEAT_CSV2_1p1 implements the functionality identified by the value `0b0010`.

From Armv8.5, the permitted values are `0b0001` and `0b0010`.

Access to this field is RO.

State3, bits [15:12]

T32EE instruction set support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

State3	Meaning
`0b0000`	Not implemented.
`0b0001`	T32EE instruction set implemented.

All other values are reserved.

In Armv8-A, the only permitted value is `0b0000`.

Access to this field is RO.

State2, bits [11:8]

Jazelle extension support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

State2	Meaning
0b0000	Not implemented.
0b0001	Jazelle extension implemented, without clearing of JOSCR.CV on exception entry.
0b0010	Jazelle extension implemented, with clearing of JOSCR.CV on exception entry.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is RO.

State1, bits [7:4]

T32 instruction set support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

State1	Meaning
0b0000	T32 instruction set not implemented.
0b0001	T32 encodings before the introduction of Thumb-2 technology implemented: • All instructions are 16-bit. • A BL or BLX is a pair of 16-bit instructions. • 32-bit instructions other than BL and BLX cannot be encoded.
0b0011	T32 encodings after the introduction of Thumb-2 technology implemented, for all 16-bit and 32-bit T32 basic instructions.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0011.

Access to this field is RO.

State0, bits [3:0]

A32 instruction set support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

State0	Meaning
0b0000	A32 instruction set not implemented.
0b0001	A32 instruction set implemented.

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Access to this field is RO.

Otherwise:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

63 … 32	31 … 0
UNKNOWN	UNKNOWN

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_PFR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_PFR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_PFR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_PFR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_PFR0_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.115 ID_PFR1_EL1, AArch32 Processor Feature Register 1

The ID_PFR1_EL1 characteristics are:

Purpose

Gives information about the AArch32 programmers' model.

Must be interpreted with ID_PFR0_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_PFR1_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_PFR1[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_PFR1_EL1 are UNDEFINED.

Attributes

ID_PFR1_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:

Bits [63:32]

Reserved, RES0.

GIC, bits [31:28]

System register GIC CPU interface.

The value of this field is an IMPLEMENTATION DEFINED choice of:

GIC	Meaning
0b0000	GIC CPU interface system registers not implemented.
0b0001	System register interface to versions 3.0 and 4.0 of the GIC CPU interface is supported.
0b0011	System register interface to version 4.1 of the GIC CPU interface is supported.

All other values are reserved.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Virt_frac, bits [27:24]

Virtualization fractional field. When the Virtualization field is `0b0000`, determines the support for Virtualization Extensions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Virt_frac	Meaning
`0b0000`	No Virtualization Extensions are implemented.
`0b0001`	The following Virtualization Extensions are implemented: • The SCR.SIF bit, if EL3 is implemented. • The modifications to the SCR.AW and SCR.FW bits described in the Virtualization Extensions, if EL3 is implemented. • The MSR (banked register) and MRS (banked register) instructions. • The ERET instruction.

All other values are reserved.

In Armv8-A, the permitted values are:

- `0b0000` when EL2 is implemented.
- `0b0001` when EL2 is not implemented.

This field is valid only when the value of ID_PFR1_EL1.Virtualization is 0, otherwise it holds the value `0b0000`.

Note

The ID_ISAR registers do not identify whether the instructions added by the Virtualization Extensions are implemented.

Access to this field is RO.

Sec_frac, bits [23:20]

Security fractional field. When the Security field is `0b0000`, determines the support for Security Extensions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Sec_frac	Meaning
`0b0000`	No Security Extensions are implemented.
`0b0001`	The following Security Extensions are implemented: • The VBAR register. • The TTBCR.PD0 and TTBCR.PD1 bits.
`0b0010`	As for `0b0001`, plus the ability to access Secure or Non-secure physical memory is supported.

All other values are reserved.

In Armv8-A, the permitted values are:

- `0b0000` when EL3 is implemented.
- `0b0001` or `0b0010` when EL3 is not implemented.

This field is valid only when the value of ID_PFR1_EL1.Security is 0, otherwise it holds the value `0b0000`.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

GenTimer, bits [19:16]

Generic Timer support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

GenTimer	Meaning
0b0000	Generic Timer is not implemented.
0b0001	Generic Timer is implemented.
0b0010	Generic Timer is implemented, and also includes support for CNTHCTL.EVNTIS and CNTKCTL.EVNTIS fields, and CNTPCTSS and CNTVCTSS counter views.

All other values are reserved.

FEAT_ECV implements the functionality identified by the value 0b0010.

In Armv8.0, the only permitted value is 0b0001.

From Armv8.6, the only permitted value is 0b0010.

Access to this field is RO.

Virtualization, bits [15:12]

Virtualization support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Virtualization	Meaning
0b0000	EL2, Hyp mode, and the HVC instruction not implemented.
0b0001	EL2, Hyp mode, the HVC instruction, and all the features described by Virt_frac == 0b0001 implemented.

All other values are reserved.

In Armv8-A, the permitted values are:

- 0b0000 when EL2 is not implemented.
- 0b0001 when EL2 is implemented.

In an implementation that includes EL2, if EL2 cannot use AArch32 but EL1 can use AArch32, then this field has the value 0b0001.

If EL1 cannot use AArch32, then this field has the value 0b0000.

Note

The ID_ISARs do not identify whether the HVC instruction is implemented.

Access to this field is RO.

MProgMod, bits [11:8]

M-profile programmers' model support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

MProgMod	Meaning
0b0000	Not supported.
0b0010	Support for two-stack programmers' model.

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

Access to this field is RO.

Security, bits [7:4]

Security support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Security	Meaning
0b0000	EL3, Monitor mode, and the SMC instruction not implemented.
0b0001	EL3, Monitor mode, the SMC instruction, and all the features described by Sec_frac == 0b0001 implemented.
0b0010	As for 0b0001, and adds the ability to set the NSACR.RFR bit. Not permitted in Armv8 as the NSACR.RFR bit is RES0.

All other values are reserved.

In Armv8-A, the permitted values are:

- 0b0000 when EL3 is not implemented.
- 0b0001 when EL3 is implemented.

In an implementation that includes EL3, if EL3 cannot use AArch32 but EL1 can use AArch32, then this field has the value 0b0001.

If EL1 cannot use AArch32, then this field has the value 0b0000.

Access to this field is RO.

ProgMod, bits [3:0]

Support for the standard programmers' model for Armv4 and later. Model must support User, FIQ, IRQ, Supervisor, Abort, Undefined, and System modes.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ProgMod	Meaning
0b0000	Not supported.
0b0001	Supported.

All other values are reserved.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

In Armv8-A, the permitted values are `0b0001` and `0b0000`.

If EL1 cannot use AArch32, then this field has the value `0b0000`.

Access to this field is RO.

Otherwise:

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_PFR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_PFR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_PFR1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_PFR1_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_PFR1_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.116 ID_PFR2_EL1, AArch32 Processor Feature Register 2

The ID_PFR2_EL1 characteristics are:

Purpose

Gives information about the AArch32 programmers' model.

Must be interpreted with ID_PFR0_EL1 and ID_PFR1_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

AArch64 System register ID_PFR2_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_PFR2[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ID_PFR2_EL1 are UNDEFINED.

Attributes

ID_PFR2_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:

Bits [63:12]

Reserved, RES0.

RAS_frac, bits [11:8]

RAS Extension fractional field.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RAS_frac	Meaning
0b0000	If ID_PFR0_EL1.RAS == 0b0001, Support for the Reliability, Availability, and Serviceability Extension is implemented.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

RAS_frac	Meaning
0b0001	If ID_PFR0_EL1.RAS == 0b0001, as 0b0000 and adds support for additional ERXMISC<m> System registers. Error records accessed through System registers conform to RAS System Architecture v1.1, which includes simplifications to ERR<n>STATUS and support for the optional RAS Timestamp Extension.

All other values are reserved.

FEAT_RAS implements the functionality identified by the value 0b0000.

FEAT_RASv1p1 implements the functionality identified by the value 0b0001.

This field is valid only if ID_PFR0_EL1.RAS == 0b0001.

Access to this field is RO.

SSBS, bits [7:4]

Speculative Store Bypassing controls in AArch64 state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SSBS	Meaning
0b0000	AArch32 provides no mechanism to control the use of Speculative Store Bypassing.
0b0001	AArch32 provides the PSTATE.SSBS mechanism to mark regions that are Speculative Store Bypass Safe.

In Armv8.0, the permitted values are 0b0000 and 0b0001.

From Armv8.5, the only permitted value is 0b0001.

All other values are reserved.

Access to this field is RO.

CSV3, bits [3:0]

Speculative use of faulting data.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CSV3	Meaning
0b0000	This PE does not disclose whether data loaded under speculation with a permission or domain fault can be used to form an address or generate condition codes or SVE predicate values to be used by other instructions in the speculative sequence.
0b0001	Data loaded under speculation with a permission or domain fault cannot be used to form an address, generate condition codes, or generate SVE predicate values to be used by other instructions in the speculative sequence. The execution timing of any other instructions in the speculative sequence is not a function of the data loaded under speculation.

All other values are reserved.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FEAT_CSV3 implements the functionality identified by the value `0b0001`.

In Armv8.0, the permitted values are `0b0000` and `0b0001`.

From Armv8.5, the only permitted value is `0b0001`.

If FEAT_E0PD is implemented, FEAT_CSV3 must be implemented.

Access to this field is RO.

Otherwise:

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_PFR2_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ID_PFR2_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0011	0b100

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && (IsFeatureImplemented(FEAT_FGT) || !IsZero(ID_PFR2_EL1) || boolean
     ↪IMPLEMENTATION_DEFINED "ID_PFR2_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_PFR2_EL1;
elsif PSTATE.EL == EL2 then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
         ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ID_PFR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ID_PFR2_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.117 IFSR32_EL2, Instruction Fault Status Register (EL2)

The IFSR32_EL2 characteristics are:

Purpose

Allows access to the AArch32 IFSR register from AArch64 state only. Its value has no effect on execution in AArch64 state.

Configuration

If EL2 is not implemented but EL3 is implemented, and EL1 is capable of using AArch32, then this register is not RES0.

AArch64 System register IFSR32_EL2 bits [31:0] are architecturally mapped to AArch32 System register IFSR[31:0].

This register is present only when FEAT_AA32EL1 is implemented. Otherwise, direct accesses to IFSR32_EL2 are UNDEFINED.

Attributes

IFSR32_EL2 is a 64-bit register.

Field descriptions

When TTBCR.EAE == '0':

Bits [63:17]

Reserved, RES0.

FnV, bit [16]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

FnV	Meaning
0b0	IFAR is valid.
0b1	IFAR is not valid, and holds an UNKNOWN value.

This field is valid only for a synchronous External abort other than a synchronous External abort on a translation table walk. It is RES0 for all other Prefetch Abort exceptions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bits [15:13]

Reserved, RES0.

ExT, bit [12]

External abort type. This bit can be used to provide an IMPLEMENTATION DEFINED classification of External aborts.

In an implementation that does not provide any classification of External aborts, this bit is RES0.

For aborts other than External aborts this bit always returns 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [11]

Reserved, RES0.

FS, bits [10, 3:0]

Fault Status bits. Bits [10] and [3:0] are interpreted together.

FS	Meaning	Applies when
0b00001	PC alignment fault.	
0b00010	Debug exception.	
0b00011	Access flag fault, level 1.	
0b00101	Translation fault, level 1.	
0b00110	Access flag fault, level 2.	
0b00111	Translation fault, level 2.	
0b01000	Synchronous External abort, not on translation table walk.	
0b01001	Domain fault, level 1.	
0b01011	Domain fault, level 2.	
0b01100	Synchronous External abort, on translation table walk, level 1.	
0b01101	Permission fault, level 1.	
0b01110	Synchronous External abort, on translation table walk, level 2.	
0b01111	Permission fault, level 2.	
0b10000	TLB conflict abort.	
0b10100	IMPLEMENTATION DEFINED fault (Lockdown fault).	
0b11001	Synchronous parity or ECC error on memory access, not on translation table walk.	FEAT_RAS is not implemented
0b11100	Synchronous parity or ECC error on translation table walk, level 1.	FEAT_RAS is not implemented
0b11110	Synchronous parity or ECC error on translation table walk, level 2.	FEAT_RAS is not implemented

All other values are reserved.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults on a Short-descriptor translation table lookup'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

LPAE, bit [9]

On taking a Data Abort exception, this bit is set as follows:

LPAE	Meaning
0b0	Using the Short-descriptor translation table formats.
0b1	Using the Long-descriptor translation table formats.

Hardware does not interpret this bit to determine the behavior of the memory system, and therefore software can set this bit to 0 or 1 without affecting operation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:4]

Reserved, RES0.

When TTBCR.EAE == '1':

Bits [63:17]

Reserved, RES0.

FnV, bit [16]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

FnV	Meaning
0b0	IFAR is valid.
0b1	IFAR is not valid, and holds an UNKNOWN value.

This field is valid only for a synchronous External abort other than a synchronous External abort on a translation table walk. It is RES0 for all other Prefetch Abort exceptions.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [15:13]

Reserved, RES0.

ExT, bit [12]

External abort type. This bit can be used to provide an IMPLEMENTATION DEFINED classification of External aborts.

In an implementation that does not provide any classification of External aborts, this bit is RES0.

For aborts other than External aborts this bit always returns 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:10]

Reserved, RES0.

LPAE, bit [9]

On taking a Data Abort exception, this bit is set as follows:

LPAE	Meaning
0b0	Using the Short-descriptor translation table formats.
0b1	Using the Long-descriptor translation table formats.

Hardware does not interpret this bit to determine the behavior of the memory system, and therefore software can set this bit to 0 or 1 without affecting operation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:6]

Reserved, RES0.

STATUS, bits [5:0]

Fault status bits. Possible values of this field are:

STATUS	Meaning	Applies when
0b000000	Address size fault in translation table base register.	
0b000001	Address size fault, level 1.	
0b000010	Address size fault, level 2.	
0b000011	Address size fault, level 3.	
0b000101	Translation fault, level 1.	
0b000110	Translation fault, level 2.	

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

STATUS	Meaning	Applies when
0b000111	Translation fault, level 3.	
0b001001	Access flag fault, level 1.	
0b001010	Access flag fault, level 2.	
0b001011	Access flag fault, level 3.	
0b001101	Permission fault, level 1.	
0b001110	Permission fault, level 2.	
0b001111	Permission fault, level 3.	
0b010000	Synchronous External abort, not on translation table walk.	
0b010101	Synchronous External abort on translation table walk, level 1.	
0b010110	Synchronous External abort on translation table walk, level 2.	
0b010111	Synchronous External abort on translation table walk, level 3.	
0b011000	Synchronous parity or ECC error on memory access, not on translation table walk.	FEAT_RAS is not implemented
0b011101	Synchronous parity or ECC error on memory access on translation table walk, level 1.	FEAT_RAS is not implemented
0b011110	Synchronous parity or ECC error on memory access on translation table walk, level 2.	FEAT_RAS is not implemented
0b011111	Synchronous parity or ECC error on memory access on translation table walk, level 3.	FEAT_RAS is not implemented
0b100001	PC alignment fault.	
0b100010	Debug exception.	
0b110000	TLB conflict abort.	

All other values are reserved.

When FEAT_RAS is implemented, 0b011000, 0b011101, 0b011110, and 0b011111 are reserved.

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults on a Long-descriptor translation table lookup'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing IFSR32_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, IFSR32_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0101	0b0000	0b001

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = IFSR32_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = IFSR32_EL2;
```

```
MSR IFSR32_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0101	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    IFSR32_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    IFSR32_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.118 ISR_EL1, Interrupt Status Register

The ISR_EL1 characteristics are:

Purpose

Shows the pending status of IRQ and FIQ interrupts and SError exceptions.

When FEAT_NMI is implemented, also shows whether a pending IRQ or FIQ interrupt has Superpriority.

Configuration

AArch64 System register ISR_EL1 bits [31:0] are architecturally mapped to AArch32 System register ISR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to ISR_EL1 are UNDEFINED.

Attributes

ISR_EL1 is a 64-bit register.

Field descriptions

Bits [63:11]

Reserved, RES0.

IS, bit [10]

When FEAT_NMI is implemented:

IRQ with Superpriority pending bit. Indicates whether an IRQ interrupt with Superpriority is pending.

IS	Meaning
0b0	No pending IRQ with Superpriority.
0b1	An IRQ interrupt with Superpriority is pending.

If all of the following apply, then this field shows the pending status of virtual IRQ interrupts with Superpriority:

- EL2 is implemented and enabled in the current Security state.
- HCR_EL2.IMO is 1.
- The PE is executing at EL1.

Otherwise, this field shows the pending status of physical IRQ interrupts with Superpriority.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FS, bit [9]

When FEAT_NMI is implemented:

FIQ with Superpriority pending bit. Indicates whether an FIQ interrupt with Superpriority is pending.

FS	Meaning
0b0	No pending FIQ with Superpriority.
0b1	An FIQ interrupt with Superpriority is pending.

If all of the following apply, then this field shows the pending status of virtual FIQ interrupts with Superpriority:

- EL2 is implemented and enabled in the current Security state.
- HCR_EL2.FMO is 1.
- The PE is executing at EL1.

Otherwise, this field shows the pending status of physical FIQ interrupts with Superpriority.

Otherwise:

Reserved, RES0.

A, bit [8]

SError exception pending bit. Indicates whether an SError exception is pending.

A	Meaning
0b0	No pending SError.
0b1	An SError exception is pending.

If all of the following apply, then this field shows the pending status of virtual SError exceptions:

- EL2 is implemented and enabled in the current Security state.
- Any of the following apply:
  - HCR_EL2.AMO is 1.
  - FEAT_DoubleFault2 is implemented and the Effective value of HCRX_EL2.TMEA is 1.
- The PE is executing at EL1.

Otherwise, if all of the following apply, then this field shows the pending status of delegated SError exceptions:

- FEAT_E3DSE is implemented.
- SCR_EL3.EnDSE is 1.
- The PE is executing at EL2 or EL1.

Otherwise, this field shows the pending status of physical SError exceptions.

If the physical SError exception is edge-triggered, this field is cleared to zero when the physical SError exception is taken.

I, bit [7]

IRQ pending bit. Indicates whether an IRQ interrupt is pending.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

I	Meaning
0b0	No pending IRQ.
0b1	An IRQ interrupt is pending.

If all of the following apply, then this field shows the pending status of virtual IRQ interrupts:

- EL2 is implemented and enabled in the current Security state.
- HCR_EL2.IMO is 1.
- The PE is executing at EL1.

Otherwise, this field shows the pending status of physical IRQ interrupts.

Note

This bit indicates the presence of a pending IRQ interrupt regardless of whether the interrupt has Superpriority.

F, bit [6]

FIQ pending bit. Indicates whether an FIQ interrupt is pending.

F	Meaning
0b0	No pending FIQ.
0b1	An FIQ interrupt is pending.

If all of the following apply, then this field shows the pending status of virtual FIQ interrupts:

- EL2 is implemented and enabled in the current Security state.
- HCR_EL2.FMO is 1.
- The PE is executing at EL1.

Otherwise, this field shows the pending status of physical FIQ interrupts.

Note

This bit indicates the presence of a pending FIQ interrupt regardless of whether the interrupt has Superpriority.

Bits [5:0]

Reserved, RES0.

Accessing ISR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ISR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b0001	0b000

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        ↪HFGRTR_EL2.ISR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = ISR_EL1;
elsif PSTATE.EL == EL2 then
    X[t, 64] = ISR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ISR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.119 LORC_EL1, LORegion Control (EL1)

The LORC_EL1 characteristics are:

Purpose

Enables and disables LORegions, and selects the current LORegion descriptor.

Configuration

If no LORegion descriptors are supported by the PE, then this register is RES0.

This register is present only when FEAT_LOR is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to LORC_EL1 are UNDEFINED.

Attributes

LORC_EL1 is a 64-bit register.

Field descriptions

Bits [63:10]

Reserved, RES0.

DS, bits [9:2]

Descriptor Select. Selects the current LORegion descriptor accessed by LORSA_EL1, LOREA_EL1, and LORN_EL1.

If this field points to an LORegion descriptor that is not supported by an implementation, then the registers LORN_EL1, LOREA_EL1, and LORSA_EL1 are RES0.

The number of LORegion descriptors in IMPLEMENTATION DEFINED. The maximum number of LORegion descriptors supported is 256. If the number is less than 256, then bits[63:M+2] are RES0, where M is Log_2(Number of LORegion descriptors supported by the implementation).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [1]

Reserved, RES0.

EN, bit [0]

Enable. Indicates whether LORegions are enabled.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EN	Meaning
0b0	Disabled. Memory accesses do not match any LORegions.
0b1	Enabled. Memory accesses may match a LORegion.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing LORC_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, LORC_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0100	0b011

```
if !(IsFeatureImplemented(FEAT_LOR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TLOR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.LORC_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = LORC_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = LORC_EL1;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        UNDEFINED;
    else
        X[t, 64] = LORC_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MSR LORC_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0100	0b011

```
if !(IsFeatureImplemented(FEAT_LOR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TLOR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.LORC_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        LORC_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        LORC_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        UNDEFINED;
    else
        LORC_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.120 LOREA_EL1, LORegion End Address (EL1)

The LOREA_EL1 characteristics are:

Purpose

Holds the physical address of the end of the LORegion described in the current LORegion descriptor selected by LORC_EL1.DS.

Configuration

This register is RES0 if any of the following apply:

- No LORegion descriptors are supported by the PE.
- LORC_EL1.DS points to a LORegion that is not supported by the PE.

This register is present only when FEAT_LOR is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to LOREA_EL1 are UNDEFINED.

Attributes

LOREA_EL1 is a 64-bit register.

Field descriptions

Any of the fields in this register are permitted to be cached in a TLB.

Bits [63:56]

Reserved, RES0.

EA[55:52], bits [55:52]

When FEAT_D128 is implemented:

Extension to EA[47:16]. For more information, see EA[47:16].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EA[51:48], bits [51:48]

When FEAT_LPA is implemented:

Extension to EA[47:16]. For more information, see EA[47:16].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

EA[47:16], bits [47:16]

Bits [47:16] of the end physical address of an LORegion described in the current LORegion descriptor selected by LORC_EL1.DS. Bits[15:0] of this address are `0xFFFF`. For implementations with fewer than 48 bits, the upper bits of this field are RES0.

When FEAT_LPA is implemented and 52-bit addresses are in use, EA[51:48] form bits [51:48] of the end physical address of the LORegion. Otherwise, when 52-bit addresses are not in use, EA[51:48] is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [15:0]

Reserved, RES0.

Accessing LOREA_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, LOREA_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0100	0b001

```
if !(IsFeatureImplemented(FEAT_LOR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TLOR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.LOREA_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = LOREA_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        X[t, 64] = LOREA_EL1;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        UNDEFINED;
    else
        X[t, 64] = LOREA_EL1;
```

MSR LOREA_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0100	0b001

```
if !(IsFeatureImplemented(FEAT_LOR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TLOR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
     ↪HFGWTR_EL2.LOREA_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        LOREA_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        LOREA_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        UNDEFINED;
    else
        LOREA_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.121 LORID_EL1, LORegionID (EL1)

The LORID_EL1 characteristics are:

Purpose

Indicates the number of LORegions and LORegion descriptors supported by the PE.

Configuration

If no LORegion descriptors are implemented, then the registers LORC_EL1, LORN_EL1, LOREA_EL1, and LORSA_EL1 are RES0.

This register is present only when FEAT_LOR is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to LORID_EL1 are UNDEFINED.

Attributes

LORID_EL1 is a 64-bit register.

Field descriptions

Bits [63:24]

Reserved, RES0.

LD, bits [23:16]

Number of LORegion descriptors supported by the PE. This is an 8-bit binary number.

The value of this field is an IMPLEMENTATION DEFINED choice of:

LD	Meaning
0x00..0xFF	The number of LORegions descriptors supported.

Access to this field is RO.

Bits [15:8]

Reserved, RES0.

LR, bits [7:0]

Number of LORegions supported by the PE. This is an 8-bit binary number.

Note

If LORID_EL1 indicates that no LORegions are implemented, then LoadLOAcquire and StoreLORelease will behave as LoadAcquire and StoreRelease.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The value of this field is an IMPLEMENTATION DEFINED choice of:

LR	Meaning
0x00..0xFF	The number of LORegions supported.

Access to this field is RO.

Accessing LORID_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, LORID_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0100	0b111

```
if !(IsFeatureImplemented(FEAT_LOR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TLOR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.LORID_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = LORID_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = LORID_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = LORID_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.122 LORN_EL1, LORegion Number (EL1)

The LORN_EL1 characteristics are:

Purpose

Holds the number of the LORegion described in the current LORegion descriptor selected by LORC_EL1.DS.

Configuration

This register is RES0 if any of the following apply:

- No LORegion descriptors are supported by the PE.
- LORC_EL1.DS points to a LORegion that is not supported by the PE.

This register is present only when FEAT_LOR is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to LORN_EL1 are UNDEFINED.

Attributes

LORN_EL1 is a 64-bit register.

Field descriptions

Any of the fields in this register are permitted to be cached in a TLB.

Bits [63:8]

Reserved, RES0.

Num, bits [7:0]

Number of the LORegion described in the current LORegion descriptor selected by LORC_EL1.DS.

The maximum number of LORegions supported by the PE is 256. If the maximum number is less than 256, then bits[8:N] are RES0, where N is (Log_2(Number of LORegions supported by the PE)).

If this field points to a LORegion that is not supported by the PE, then the current LORegion descriptor does not match any LORegion.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing LORN_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, LORN_EL1
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0100	0b010

```
if !(IsFeatureImplemented(FEAT_LOR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TLOR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.LORN_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = LORN_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = LORN_EL1;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        UNDEFINED;
    else
        X[t, 64] = LORN_EL1;
```

```
MSR LORN_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0100	0b010

```
if !(IsFeatureImplemented(FEAT_LOR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TLOR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.LORN_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        LORN_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        LORN_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        UNDEFINED;
    else
        LORN_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.123 LORSA_EL1, LORegion Start Address (EL1)

The LORSA_EL1 characteristics are:

Purpose

Indicates whether the current LORegion descriptor selected by LORC_EL1.DS is enabled, and holds the physical address of the start of the LORegion.

Configuration

This register is RES0 if any of the following apply:

- No LORegion descriptors are supported by the PE.
- LORC_EL1.DS points to a LORegion that is not supported by the PE.

This register is present only when FEAT_LOR is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to LORSA_EL1 are UNDEFINED.

Attributes

LORSA_EL1 is a 64-bit register.

Field descriptions

Any of the fields in this register are permitted to be cached in a TLB.

Bits [63:56]

Reserved, RES0.

SA, bits [55:16]

When FEAT_D128 is implemented:

SA, bits [39:0]

Bits [55:16] of the start physical address of the LORegion described in the current LORegion descriptor selected by LORC_EL1.DS.

Bits[15:0] of this address are `0x0000`.

For implementations with fewer than 56 physical address bits, the corresponding upper bits of this field are RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_LPA is implemented and FEAT_D128 is not implemented:

Bits [39:36]

Reserved, RES0.

SA, bits [35:0]

Bits [51:16] of the start physical address of the LORegion described in the current LORegion descriptor selected by LORC_EL1.DS.

Bits[15:0] of this address are `0x0000`.

For implementations with fewer than 52 physical address bits, the corresponding upper bits of this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_LPA is not implemented:

Bits [39:32]

Reserved, RES0.

SA, bits [31:0]

Bits [47:16] of the start physical address of the LORegion described in the current LORegion descriptor selected by LORC_EL1.DS.

Bits[15:0] of this address are `0x0000`.

For implementations with fewer than 48 physical address bits, the corresponding upper bits of this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [15:1]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Valid, bit [0]

Indicates whether the current LORegion descriptor is enabled.

Valid	Meaning
0b0	LORegion descriptor is disabled.
0b1	LORegion descriptor is enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing LORSA_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, LORSA_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_LOR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TLOR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.LORSA_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = LORSA_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = LORSA_EL1;
elsif PSTATE.EL == EL3 then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    if SCR_EL3.NS == '0' then
        UNDEFINED;
    else
        X[t, 64] = LORSA_EL1;
```

MSR LORSA_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_LOR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TLOR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
     ↪HFGWTR_EL2.LORSA_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        LORSA_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        LORSA_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        UNDEFINED;
    else
        LORSA_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.124 MAIR2_EL1, Extended Memory Attribute Indirection Register (EL1)

The MAIR2_EL1 characteristics are:

Purpose

Provides the memory attribute encodings corresponding to the possible AttrIndx values in a VMSAv8-64 or VMSAv9-128 translation table entry for stage 1 translations at EL1.

Configuration

This register is present only when FEAT_AIE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to MAIR2_EL1 are UNDEFINED.

Attributes

MAIR2_EL1 is a 64-bit register.

Field descriptions

Attr<n> , bits [8n+7:8n], for n = 7 to 0

Memory Attribute encoding.

When stage 1 Attributes Index Extension is enabled and AttrIndx[3] in a VMSAv8-64 or VMSAv9-128 translation table entry is 1, AttrIndx[2:0] gives the value of <n> in Attr<n>.

When stage 1 Attributes Index Extension is enabled and AttrIndx[3] in a VMSAv8-64 or VMSAv9-128 translation table entry is 0, see MAIR_EL1.Attr

Attr is encoded as follows:

Attr		Meaning
0b0000dd00		Device memory. See encoding of ‘dd’ for the type of Device memory.
0b0000dd01		If FEAT_XS is implemented: Device memory with the XS attribute set to 0. See encoding of ‘dd’ for the type of Device memory. Otherwise, UNPREDICTABLE.
0b0000dd1x		UNPREDICTABLE.
0booooiiii	where oooo != 0000 and iiii != 0000	Normal memory. See encoding of ‘oooo’ and ‘iiii’ for the type of Normal memory.
0b01000000		If FEAT_XS is implemented: Normal Inner Non-cacheable, Outer Non-cacheable memory with the XS attribute set to 0. Otherwise, UNPREDICTABLE.
0b10100000		If FEAT_XS is implemented: Normal Inner Write-through Cacheable, Outer Write-through Cacheable, Read-Allocate, No-Write Allocate, Non-transient memory with the XS attribute set to 0. Otherwise, UNPREDICTABLE.
0b11110000		If FEAT_MTE2 is implemented: Tagged Normal Inner Write-Back, Outer Write-Back, Read-Allocate, Write-Allocate Non-transient memory. Otherwise, UNPREDICTABLE.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Attr		Meaning
0bxxxx0000	where xxxx != 0000 and xxxx != 0100 and xxxx != 1010 and xxxx != 1111	UNPREDICTABLE.

dd is encoded as follows:

'dd'	Meaning
0b00	Device-nGnRnE memory.
0b01	Device-nGnRE memory.
0b10	Device-nGRE memory.
0b11	Device-GRE memory.

oooo is encoded as follows:

'oooo'		Meaning
0b0000		See encoding of Attr.
0b00RW	where RW != 00	Normal memory, Outer Write-Through Transient.
0b0100		Normal memory, Outer Non-cacheable.
0b01RW	where RW != 00	Normal memory, Outer Write-Back Transient.
0b10RW		Normal memory, Outer Write-Through Non-transient.
0b11RW		Normal memory, Outer Write-Back Non-transient.

R encodes the Outer Read-Allocate policy and W encodes the Outer Write-Allocate policy.

iiii is encoded as follows:

'iiii'		Meaning
0b0000		See encoding of Attr.
0b00RW	where RW != 00	Normal memory, Inner Write-Through Transient.
0b0100		Normal memory, Inner Non-cacheable.
0b01RW	where RW != 00	Normal memory, Inner Write-Back Transient.
0b10RW		Normal memory, Inner Write-Through Non-transient.
0b11RW		Normal memory, Inner Write-Back Non-transient.

R encodes the Inner Read-Allocate policy and W encodes the Inner Write-Allocate policy.

In oooo and iiii, R and W are encoded as follows:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

'R' or 'W'	Meaning
0b0	No Allocate.
0b1	Allocate.

When FEAT_XS is implemented, stage 1 Inner Write-Back Cacheable, Outer Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MAIR2_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name MAIR2_EL1 or MAIR2_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MAIR2_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_AIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.nMAIR2_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x280];
    else
        X[t, 64] = MAIR2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = MAIR2_EL2;
    else
        X[t, 64] = MAIR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MAIR2_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MSR MAIR2_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_AIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.nMAIR2_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x280] = X[t, 64];
    else
        MAIR2_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        MAIR2_EL2 = X[t, 64];
    else
        MAIR2_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MAIR2_EL1 = X[t, 64];
```

```
When FEAT_VHE is implemented MRS <Xt>, MAIR2_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_AIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x280];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if ELIsInHost(EL2) then
            if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
                UNDEFINED;
            elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
                if EL3SDDUndef() then
                    UNDEFINED;
                else
                    AArch64.SystemAccessTrap(EL3, 0x18);
            else
                X[t, 64] = MAIR2_EL1;
        else
            UNDEFINED;
    elsif PSTATE.EL == EL3 then
        if ELIsInHost(EL2) then
            X[t, 64] = MAIR2_EL1;
        else
            UNDEFINED;
```

When FEAT_VHE is implemented MSR MAIR2_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_AIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x280] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            MAIR2_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        MAIR2_EL1 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.125 MAIR2_EL2, Extended Memory Attribute Indirection Register (EL2)

The MAIR2_EL2 characteristics are:

Purpose

Provides the memory attribute encodings corresponding to the possible AttrIndx values in a VMSAv8-64 or VMSAv9-128 translation table entry for stage 1 translations at EL1.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_AIE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to MAIR2_EL2 are UNDEFINED.

Attributes

MAIR2_EL2 is a 64-bit register.

Field descriptions

Attr<n> , bits [8n+7:8n], for n = 7 to 0

Memory Attribute encoding.

When stage 1 Attributes Index Extension is enabled and AttrIndx[3] in a VMSAv8-64 or VMSAv9-128 translation table entry is 1, AttrIndx[2:0] gives the value of <n> in Attr<n>.

When stage 1 Attributes Index Extension is enabled and AttrIndx[3] in a VMSAv8-64 or VMSAv9-128 translation table entry is 0, see MAIR_EL2.Attr

Attr is encoded as follows:

Attr		Meaning
`0b0000`dd00		Device memory. See encoding of ‘dd’ for the type of Device memory.
`0b0000`dd01		If FEAT_XS is implemented: Device memory with the XS attribute set to 0. See encoding of ‘dd’ for the type of Device memory. Otherwise, UNPREDICTABLE.
`0b0000`dd1x		UNPREDICTABLE.
0boooiiii	where oooo != 0000 and iiii != 0000	Normal memory. See encoding of ‘oooo’ and ‘iiii’ for the type of Normal memory.
`0b01000000`		If FEAT_XS is implemented: Normal Inner Non-cacheable, Outer Non-cacheable memory with the XS attribute set to 0. Otherwise, UNPREDICTABLE.
`0b10100000`		If FEAT_XS is implemented: Normal Inner Write-through Cacheable, Outer Write-through Cacheable, Read-Allocate, No-Write Allocate, Non-transient memory with the XS attribute set to 0. Otherwise, UNPREDICTABLE.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Attr		Meaning
0b11110000		If FEAT_MTE2 is implemented: Tagged Normal Inner Write-Back, Outer Write-Back, Read-Allocate, Write-Allocate Non-transient memory. Otherwise, UNPREDICTABLE.
0bxxxx0000	where xxxx != 0000 and xxxx != 0100 and xxxx != 1010 and xxxx != 1111	UNPREDICTABLE.

dd is encoded as follows:

'dd'	Meaning
0b00	Device-nGnRnE memory.
0b01	Device-nGnRE memory.
0b10	Device-nGRE memory.
0b11	Device-GRE memory.

oooo is encoded as follows:

'oooo'		Meaning
0b0000		See encoding of Attr.
0b00RW	where RW != 00	Normal memory, Outer Write-Through Transient.
0b0100		Normal memory, Outer Non-cacheable.
0b01RW	where RW != 00	Normal memory, Outer Write-Back Transient.
0b10RW		Normal memory, Outer Write-Through Non-transient.
0b11RW		Normal memory, Outer Write-Back Non-transient.

R encodes the Outer Read-Allocate policy and W encodes the Outer Write-Allocate policy.

iiii is encoded as follows:

'iiii'		Meaning
0b0000		See encoding of Attr.
0b00RW	where RW != 00	Normal memory, Inner Write-Through Transient.
0b0100		Normal memory, Inner Non-cacheable.
0b01RW	where RW != 00	Normal memory, Inner Write-Back Transient.
0b10RW		Normal memory, Inner Write-Through Non-transient.
0b11RW		Normal memory, Inner Write-Back Non-transient.

R encodes the Inner Read-Allocate policy and W encodes the Inner Write-Allocate policy.

In oooo and iiii, R and W are encoded as follows:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

'R' or 'W'	Meaning
0b0	No Allocate.
0b1	Allocate.

When FEAT_XS is implemented, stage 1 Inner Write-Back Cacheable, Outer Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MAIR2_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name MAIR2_EL2 or MAIR2_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MAIR2_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_AIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MAIR2_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MAIR2_EL2;
```

```
MSR MAIR2_EL2, <Xt>
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_AIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MAIR2_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MAIR2_EL2 = X[t, 64];
```

MRS <Xt>, MAIR2_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_AIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.nMAIR2_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x280];
    else
        X[t, 64] = MAIR2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = MAIR2_EL2;
    else
        X[t, 64] = MAIR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MAIR2_EL1;
```

MSR MAIR2_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_AIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.nMAIR2_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x280] = X[t, 64];
    else
        MAIR2_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.AIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        MAIR2_EL2 = X[t, 64];
    else
        MAIR2_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MAIR2_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.126 MAIR2_EL3, Extended Memory Attribute Indirection Register (EL3)

The MAIR2_EL3 characteristics are:

Purpose

Provides the memory attribute encodings corresponding to the possible AttrIndx values in a VMSAv8-64 or VMSAv9-128 translation table entry for stage 1 translations at EL1.

Configuration

This register is present only when FEAT_AIE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to MAIR2_EL3 are UNDEFINED.

Attributes

MAIR2_EL3 is a 64-bit register.

Field descriptions

Attr<n> , bits [8n+7:8n], for n = 7 to 0

Memory Attribute encoding.

When stage 1 Attributes Index Extension is enabled and AttrIndx[3] in a VMSAv8-64 or VMSAv9-128 translation table entry is 1, AttrIndx[2:0] gives the value of <n> in Attr<n>.

When stage 1 Attributes Index Extension is enabled and AttrIndx[3] in a VMSAv8-64 or VMSAv9-128 translation table entry is 0, see MAIR_EL3.Attr

Attr is encoded as follows:

Attr		Meaning
0b0000dd00		Device memory. See encoding of ‘dd’ for the type of Device memory.
0b0000dd01		If FEAT_XS is implemented: Device memory with the XS attribute set to 0. See encoding of ‘dd’ for the type of Device memory. Otherwise, UNPREDICTABLE.
0b0000dd1x		UNPREDICTABLE.
0booooiiii	where oooo != 0000 and iiii != 0000	Normal memory. See encoding of ‘oooo’ and ‘iiii’ for the type of Normal memory.
0b01000000		If FEAT_XS is implemented: Normal Inner Non-cacheable, Outer Non-cacheable memory with the XS attribute set to 0. Otherwise, UNPREDICTABLE.
0b10100000		If FEAT_XS is implemented: Normal Inner Write-through Cacheable, Outer Write-through Cacheable, Read-Allocate, No-Write Allocate, Non-transient memory with the XS attribute set to 0. Otherwise, UNPREDICTABLE.
0b11110000		If FEAT_MTE2 is implemented: Tagged Normal Inner Write-Back, Outer Write-Back, Read-Allocate, Write-Allocate Non-transient memory. Otherwise, UNPREDICTABLE.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Attr		Meaning
0bxxxx0000	where xxxx != 0000 and xxxx != 0100 and xxxx != 1010 and xxxx != 1111	UNPREDICTABLE.

dd is encoded as follows:

'dd'	Meaning
0b00	Device-nGnRnE memory.
0b01	Device-nGnRE memory.
0b10	Device-nGRE memory.
0b11	Device-GRE memory.

oooo is encoded as follows:

'oooo'		Meaning
0b0000		See encoding of Attr.
0b00RW	where RW != 00	Normal memory, Outer Write-Through Transient.
0b0100		Normal memory, Outer Non-cacheable.
0b01RW	where RW != 00	Normal memory, Outer Write-Back Transient.
0b10RW		Normal memory, Outer Write-Through Non-transient.
0b11RW		Normal memory, Outer Write-Back Non-transient.

R encodes the Outer Read-Allocate policy and W encodes the Outer Write-Allocate policy.

iiii is encoded as follows:

'iiii'		Meaning
0b0000		See encoding of Attr.
0b00RW	where RW != 00	Normal memory, Inner Write-Through Transient.
0b0100		Normal memory, Inner Non-cacheable.
0b01RW	where RW != 00	Normal memory, Inner Write-Back Transient.
0b10RW		Normal memory, Inner Write-Through Non-transient.
0b11RW		Normal memory, Inner Write-Back Non-transient.

R encodes the Inner Read-Allocate policy and W encodes the Inner Write-Allocate policy.

In oooo and iiii, R and W are encoded as follows:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

'R' or 'W'	Meaning
0b0	No Allocate.
0b1	Allocate.

When FEAT_XS is implemented, stage 1 Inner Write-Back Cacheable, Outer Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MAIR2_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MAIR2_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_AIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MAIR2_EL3;
```

```
MSR MAIR2_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_AIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.MAIR2_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MAIR2_EL3 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.127 MAIR_EL1, Memory Attribute Indirection Register (EL1)

The MAIR_EL1 characteristics are:

Purpose

Provides the memory attribute encodings corresponding to the possible AttrIndx values in a Long-descriptor format translation table entry for stage 1 translations at EL1.

Configuration

When TTBCR.EAE == ‘0’, AArch64 System register MAIR_EL1 bits [31:0] are architecturally mapped to AArch32 System register PRRR[31:0].

When TTBCR.EAE == ‘1’, AArch64 System register MAIR_EL1 bits [31:0] are architecturally mapped to AArch32 System register MAIR0[31:0].

When TTBCR.EAE == ‘0’, AArch64 System register MAIR_EL1 bits [63:32] are architecturally mapped to AArch32 System register NMRR[31:0].

When TTBCR.EAE == ‘1’, AArch64 System register MAIR_EL1 bits [63:32] are architecturally mapped to AArch32 System register MAIR1[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to MAIR_EL1 are UNDEFINED.

Attributes

MAIR_EL1 is a 64-bit register.

Field descriptions

MAIR_EL1 is permitted to be cached in a TLB.

Attr<n> , bits [8n+7:8n], for n = 7 to 0

Memory Attribute encoding.

When FEAT_AIE is implemented and stage 1 Attributes Index Extension is enabled and AttrIndx[3] in a Long descriptor format translation table entry is 0, or when FEAT_AIE is not implemented, AttrIndx[2:0] gives the value of <n> in Attr<n>.

When FEAT_AIE is implemented and stage 1 Attributes Index Extension is enabled and AttrIndx[3] in a Long descriptor format translation table entry is 1, see MAIR2_EL1.Attr

Attr is encoded as follows:

Attr	Meaning
0b0000dd00	Device memory. See encoding of ‘dd’ for the type of Device memory.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Attr		Meaning
0b0000dd01		If FEAT_XS is implemented: Device memory with the XS attribute set to 0. See encoding of ‘dd’ for the type of Device memory. Otherwise, UNPREDICTABLE.
0b0000dd1x		UNPREDICTABLE.
0booooiiii	where oooo != 0000 and iiii != 0000	Normal memory. See encoding of ‘oooo’ and ‘iiii’ for the type of Normal memory.
0b01000000		If FEAT_XS is implemented: Normal Inner Non-cacheable, Outer Non-cacheable memory with the XS attribute set to 0. Otherwise, UNPREDICTABLE.
0b10100000		If FEAT_XS is implemented: Normal Inner Write-through Cacheable, Outer Write-through Cacheable, Read-Allocate, No-Write Allocate, Non-transient memory with the XS attribute set to 0. Otherwise, UNPREDICTABLE.
0b11110000		If FEAT_MTE2 is implemented: Tagged Normal Inner Write-Back, Outer Write-Back, Read-Allocate, Write-Allocate Non-transient memory. Otherwise, UNPREDICTABLE.
0bxxxx0000	where xxxx != 0000 and xxxx != 0100 and xxxx != 1010 and xxxx != 1111	UNPREDICTABLE.

dd is encoded as follows:

‘dd’	Meaning
0b00	Device-nGnRnE memory.
0b01	Device-nGnRE memory.
0b10	Device-nGRE memory.
0b11	Device-GRE memory.

oooo is encoded as follows:

‘oooo’		Meaning
0b0000		See encoding of Attr.
0b00RW	where RW != 00	Normal memory, Outer Write-Through Transient.
0b0100		Normal memory, Outer Non-cacheable.
0b01RW	where RW != 00	Normal memory, Outer Write-Back Transient.
0b10RW		Normal memory, Outer Write-Through Non-transient.
0b11RW		Normal memory, Outer Write-Back Non-transient.

R encodes the Outer Read-Allocate policy and W encodes the Outer Write-Allocate policy.

iiii is encoded as follows:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

'iiii'		Meaning
0b0000		See encoding of Attr.
0b00RW	where RW != 00	Normal memory, Inner Write-Through Transient.
0b0100		Normal memory, Inner Non-cacheable.
0b01RW	where RW != 00	Normal memory, Inner Write-Back Transient.
0b10RW		Normal memory, Inner Write-Through Non-transient.
0b11RW		Normal memory, Inner Write-Back Non-transient.

R encodes the Inner Read-Allocate policy and W encodes the Inner Write-Allocate policy.

In oooo and iiii, R and W are encoded as follows:

'R' or 'W'	Meaning
0b0	No Allocate.
0b1	Allocate.

When FEAT_XS is implemented, stage 1 Inner Write-Back Cacheable, Outer Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MAIR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name MAIR_EL1 or MAIR_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MAIR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.MAIR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x140];
    else
        X[t, 64] = MAIR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = MAIR_EL2;
    else
        X[t, 64] = MAIR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MAIR_EL1;
```

MSR MAIR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.MAIR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x140] = X[t, 64];
    else
        MAIR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        MAIR_EL2 = X[t, 64];
    else
        MAIR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MAIR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, MAIR_EL12

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x140];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = MAIR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = MAIR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented MSR MAIR_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x140] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        MAIR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        MAIR_EL1 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.128 MAIR_EL2, Memory Attribute Indirection Register (EL2)

The MAIR_EL2 characteristics are:

Purpose

Provides the memory attribute encodings corresponding to the possible AttrIndx values in a Long-descriptor format translation table entry for stage 1 translations at EL2.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

AArch64 System register MAIR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HMAIR0[31:0].

AArch64 System register MAIR_EL2 bits [63:32] are architecturally mapped to AArch32 System register HMAIR1[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to MAIR_EL2 are UNDEFINED.

Attributes

MAIR_EL2 is a 64-bit register.

Field descriptions

MAIR_EL2 is permitted to be cached in a TLB.

Attr<n> , bits [8n+7:8n], for n = 7 to 0

Memory Attribute encoding.

When FEAT_AIE is implemented and stage 1 Attributes Index Extension is enabled and AttrIndx[3] in a Long descriptor format translation table entry is 0, or when FEAT_AIE is not implemented, AttrIndx[2:0] gives the value of <n> in Attr<n>.

When FEAT_AIE is implemented and stage 1 Attributes Index Extension is enabled and AttrIndx[3] in a Long descriptor format translation table entry is 1, see MAIR2_EL2.Attr

Attr is encoded as follows:

Attr	Meaning
`0b0000`dd00	Device memory. See encoding of ‘dd’ for the type of Device memory.
`0b0000`dd01	If FEAT_XS is implemented: Device memory with the XS attribute set to 0. See encoding of ‘dd’ for the type of Device memory. Otherwise, UNPREDICTABLE.
`0b0000`dd1x	UNPREDICTABLE.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Attr		Meaning
0booooiiii	where oooo != 0000 and iiii != 0000	Normal memory. See encoding of ‘oooo’ and ‘iiii’ for the type of Normal memory.
`0b01000000`		If FEAT_XS is implemented: Normal Inner Non-cacheable, Outer Non-cacheable memory with the XS attribute set to 0. Otherwise, UNPREDICTABLE.
`0b10100000`		If FEAT_XS is implemented: Normal Inner Write-through Cacheable, Outer Write-through Cacheable, Read-Allocate, No-Write Allocate, Non-transient memory with the XS attribute set to 0. Otherwise, UNPREDICTABLE.
`0b11110000`		If FEAT_MTE2 is implemented: Tagged Normal Inner Write-Back, Outer Write-Back, Read-Allocate, Write-Allocate Non-transient memory. Otherwise, UNPREDICTABLE.
`0bxxxx0000`	where xxxx != 0000 and xxxx != 0100 and xxxx != 1010 and xxxx != 1111	UNPREDICTABLE.

`dd` is encoded as follows:

‘dd’	Meaning
`0b00`	Device-nGnRnE memory.
`0b01`	Device-nGnRE memory.
`0b10`	Device-nGRE memory.
`0b11`	Device-GRE memory.

`oooo` is encoded as follows:

‘oooo’		Meaning
`0b0000`		See encoding of Attr.
`0b00`RW	where RW != 00	Normal memory, Outer Write-Through Transient.
`0b0100`		Normal memory, Outer Non-cacheable.
`0b01`RW	where RW != 00	Normal memory, Outer Write-Back Transient.
`0b10`RW		Normal memory, Outer Write-Through Non-transient.
`0b11`RW		Normal memory, Outer Write-Back Non-transient.

`R` encodes the Outer Read-Allocate policy and `W` encodes the Outer Write-Allocate policy.

`iiii` is encoded as follows:

‘iiii’		Meaning
`0b0000`		See encoding of Attr.
`0b00`RW	where RW != 00	Normal memory, Inner Write-Through Transient.
`0b0100`		Normal memory, Inner Non-cacheable.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

'iiii'		Meaning
0b01RW	where RW != 00	Normal memory, Inner Write-Back Transient.
0b10RW		Normal memory, Inner Write-Through Non-transient.
0b11RW		Normal memory, Inner Write-Back Non-transient.

R encodes the Inner Read-Allocate policy and W encodes the Inner Write-Allocate policy.

In oooo and iiii, R and W are encoded as follows:

'R' or 'W'	Meaning
0b0	No Allocate.
0b1	Allocate.

When FEAT_XS is implemented, stage 1 Inner Write-Back Cacheable, Outer Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MAIR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name MAIR_EL2 or MAIR_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MAIR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = MAIR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MAIR_EL2;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MSR MAIR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    MAIR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MAIR_EL2 = X[t, 64];
```

```
MRS <Xt>, MAIR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.MAIR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x140];
    else
        X[t, 64] = MAIR_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = MAIR_EL2;
    else
        X[t, 64] = MAIR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MAIR_EL1;
```

```
MSR MAIR_EL1, <Xt>
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.MAIR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x140] = X[t, 64];
    else
        MAIR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        MAIR_EL2 = X[t, 64];
    else
        MAIR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MAIR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.129 MAIR_EL3, Memory Attribute Indirection Register (EL3)

The MAIR_EL3 characteristics are:

Purpose

Provides the memory attribute encodings corresponding to the possible AttrIndx values in a Long-descriptor format translation table entry for stage 1 translations at EL3.

Configuration

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to MAIR_EL3 are UNDEFINED.

Attributes

MAIR_EL3 is a 64-bit register.

Field descriptions

MAIR_EL3 is permitted to be cached in a TLB.

Attr<n> , bits [8n+7:8n], for n = 7 to 0

Memory Attribute encoding.

When FEAT_AIE is implemented and stage 1 Attributes Index Extension is enabled and AttrIndx[3] in a Long descriptor format translation table entry is 0, or when FEAT_AIE is not implemented, AttrIndx[2:0] gives the value of <n> in Attr<n>.

When FEAT_AIE is implemented and stage 1 Attributes Index Extension is enabled and AttrIndx[3] in a Long descriptor format translation table entry is 1, see MAIR2_EL3.Attr

Attr is encoded as follows:

Attr		Meaning
`0b0000`dd00		Device memory. See encoding of ‘dd’ for the type of Device memory.
`0b0000`dd01		If FEAT_XS is implemented: Device memory with the XS attribute set to 0. See encoding of ‘dd’ for the type of Device memory. Otherwise, UNPREDICTABLE.
`0b0000`dd1x		UNPREDICTABLE.
0boooo iiii	where oooo != 0000 and iiii != 0000	Normal memory. See encoding of ‘oooo’ and ‘iiii’ for the type of Normal memory.
`0b01000000`		If FEAT_XS is implemented: Normal Inner Non-cacheable, Outer Non-cacheable memory with the XS attribute set to 0. Otherwise, UNPREDICTABLE.
`0b10100000`		If FEAT_XS is implemented: Normal Inner Write-through Cacheable, Outer Write-through Cacheable, Read-Allocate, No-Write Allocate, Non-transient memory with the XS attribute set to 0. Otherwise, UNPREDICTABLE.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Attr		Meaning
0b11110000		If FEAT_MTE2 is implemented: Tagged Normal Inner Write-Back, Outer Write-Back, Read-Allocate, Write-Allocate Non-transient memory. Otherwise, UNPREDICTABLE.
0bxxxx0000	where xxxx != 0000 and xxxx != 0100 and xxxx != 1010 and xxxx != 1111	UNPREDICTABLE.

dd is encoded as follows:

'dd'	Meaning
0b00	Device-nGnRnE memory.
0b01	Device-nGnRE memory.
0b10	Device-nGRE memory.
0b11	Device-GRE memory.

oooo is encoded as follows:

'oooo'		Meaning
0b0000		See encoding of Attr.
0b00RW	where RW != 00	Normal memory, Outer Write-Through Transient.
0b0100		Normal memory, Outer Non-cacheable.
0b01RW	where RW != 00	Normal memory, Outer Write-Back Transient.
0b10RW		Normal memory, Outer Write-Through Non-transient.
0b11RW		Normal memory, Outer Write-Back Non-transient.

R encodes the Outer Read-Allocate policy and W encodes the Outer Write-Allocate policy.

iiii is encoded as follows:

'iiii'		Meaning
0b0000		See encoding of Attr.
0b00RW	where RW != 00	Normal memory, Inner Write-Through Transient.
0b0100		Normal memory, Inner Non-cacheable.
0b01RW	where RW != 00	Normal memory, Inner Write-Back Transient.
0b10RW		Normal memory, Inner Write-Through Non-transient.
0b11RW		Normal memory, Inner Write-Back Non-transient.

R encodes the Inner Read-Allocate policy and W encodes the Inner Write-Allocate policy.

In oooo and iiii, R and W are encoded as follows:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

'R' or 'W'	Meaning
0b0	No Allocate.
0b1	Allocate.

When FEAT_XS is implemented, stage 1 Inner Write-Back Cacheable, Outer Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MAIR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MAIR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b0010	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MAIR_EL3;
```

```
MSR MAIR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b0010	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.MAIR_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MAIR_EL3 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.130 MECID_A0_EL2, Alternate MECID for EL2 and EL2&0 translation regimes

The MECID_A0_EL2 characteristics are:

Purpose

Alternate MECID for EL2 and EL2&0 accesses translated by TTBR0_EL2.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_MEC is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to MECID_A0_EL2 are UNDEFINED.

Attributes

MECID_A0_EL2 is a 64-bit register.

Field descriptions

Bits [63:16]

Reserved, RES0.

MECID, bits [15:0]

If MECIDWidth is less than 16 bits, bits[15:MECIDWidth] are RES0.

Note

MECIDWidth is defined in MECIDR_EL2.MECIDWidthm1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MECID_A0_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MECID_A0_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b1000	0b001

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_MEC) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Realm) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.MECEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.MECEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MECID_A0_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MECID_A0_EL2;
```

MSR MECID_A0_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b1000	0b001

```
if !(IsFeatureImplemented(FEAT_MEC) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Realm) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.MECEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.MECEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MECID_A0_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MECID_A0_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.131 MECID_A1_EL2, Alternate MECID for EL2&0 translation regimes.

The MECID_A1_EL2 characteristics are:

Purpose

Alternate MECID for EL2&0 accesses translated by TTBR1_EL2.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_MEC is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to MECID_A1_EL2 are UNDEFINED.

Attributes

MECID_A1_EL2 is a 64-bit register.

Field descriptions

Bits [63:16]

Reserved, RES0.

MECID, bits [15:0]

If MECIDWidth is less than 16 bits, bits[15:MECIDWidth] are RES0.

Note

MECIDWidth is defined in MECIDR_EL2.MECIDWidthm1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MECID_A1_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MECID_A1_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b1000	0b011

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_MEC) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Realm) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.MECEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.MECEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MECID_A1_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MECID_A1_EL2;
```

MSR MECID_A1_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b1000	0b011

```
if !(IsFeatureImplemented(FEAT_MEC) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Realm) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.MECEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.MECEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MECID_A1_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MECID_A1_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.132 MECID_P0_EL2, Primary MECID for EL2 and EL2&0 translation regimes

The MECID_P0_EL2 characteristics are:

Purpose

Primary MECID for EL2 and EL2&0 accesses translated by TTBR0_EL2.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_MEC is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to MECID_P0_EL2 are UNDEFINED.

Attributes

MECID_P0_EL2 is a 64-bit register.

Field descriptions

Bits [63:16]

Reserved, RES0.

MECID, bits [15:0]

If MECIDWidth is less than 16 bits, bits[15:MECIDWidth] are RES0.

Note

MECIDWidth is defined in MECIDR_EL2.MECIDWidthm1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MECID_P0_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MECID_P0_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b1000	0b000

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_MEC) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Realm) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.MECEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.MECEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MECID_P0_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MECID_P0_EL2;
```

MSR MECID_P0_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b1000	0b000

```
if !(IsFeatureImplemented(FEAT_MEC) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Realm) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.MECEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.MECEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MECID_P0_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MECID_P0_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.133 MECID_P1_EL2, Primary MECID for EL2&0 translation regimes

The MECID_P1_EL2 characteristics are:

Purpose

Primary MECID for EL2&0 accesses translated by TTBR1_EL2.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_MEC is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to MECID_P1_EL2 are UNDEFINED.

Attributes

MECID_P1_EL2 is a 64-bit register.

Field descriptions

Bits [63:16]

Reserved, RES0.

MECID, bits [15:0]

If MECIDWidth is less than 16 bits, bits[15:MECIDWidth] are RES0.

Note

MECIDWidth is defined in MECIDR_EL2.MECIDWidthm1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MECID_P1_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MECID_P1_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b1000	0b010

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_MEC) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Realm) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.MECEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.MECEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MECID_P1_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MECID_P1_EL2;
```

MSR MECID_P1_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b1000	0b010

```
if !(IsFeatureImplemented(FEAT_MEC) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Realm) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.MECEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.MECEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MECID_P1_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MECID_P1_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.134 MECID_RL_A_EL3, Realm PA space Alternate MECID for EL3 stage 1 translation regime

The MECID_RL_A_EL3 characteristics are:

Purpose

Alternate MECID for EL3 accesses to the Realm physical address space, translated by TTBR0_EL3.

Configuration

This register is present only when FEAT_MEC is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to MECID_RL_A_EL3 are UNDEFINED.

Attributes

MECID_RL_A_EL3 is a 64-bit register.

Field descriptions

Bits [63:16]

Reserved, RES0.

MECID, bits [15:0]

If MECIDWidth is less than 16 bits, bits[15:MECIDWidth] are RES0.

Note

MECIDWidth is defined in MECIDR_EL2.MECIDWidthm1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MECID_RL_A_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MECID_RL_A_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b1010	0b001

```
if !(IsFeatureImplemented(FEAT_MEC) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MECID_RL_A_EL3;
```

MSR MECID_RL_A_EL3, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b1010	0b001

```
if !(IsFeatureImplemented(FEAT_MEC) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.MECID_RL_A_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MECID_RL_A_EL3 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.135 MECIDR_EL2, MEC Identification Register

The MECIDR_EL2 characteristics are:

Purpose

MEC identification register.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_MEC is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to MECIDR_EL2 are UNDEFINED.

Attributes

MECIDR_EL2 is a 64-bit register.

Field descriptions

Bits [63:4]

Reserved, RES0.

MECIDWidthm1, bits [3:0]

MECID width minus 1.

The value of this field plus 1 is the MECID width in bits, that this PE supports.

This field has an IMPLEMENTATION DEFINED value.

For example, the value `0b1111` indicates that this PE supports a MECID width of 16 bits, and provides 2^{16} possible MECID values.

MECIDWidth is defined as MECIDR_EL2.MECIDWidthm1 + 1.

Access to this field is RO.

Accessing MECIDR_EL2

For accesses from EL2 and EL3, this register is RO.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MECIDR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b1000	0b111

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_MEC) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = MECIDR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MECIDR_EL2;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.136 MFAR_EL3, Physical Fault Address Register (EL3)

The MFAR_EL3 characteristics are:

Purpose

Records the faulting physical address for a Granule Protection Check, synchronous External abort, or SError exception taken to EL3.

Configuration

This register is present only when (FEAT_PFAR is implemented or FEAT_RME is implemented) and FEAT_AA64 is implemented. Otherwise, direct accesses to MFAR_EL3 are UNDEFINED.

Attributes

MFAR_EL3 is a 64-bit register.

Field descriptions

When FEAT_RME is implemented and the exception is a GPC exception:

NS, bit [63]

Together with MFAR_EL3.NSE, reports the physical address space of the access that triggered the exception.

NSE	NS	Meaning
0b0	0b0	When Secure state is implemented, Secure. Otherwise reserved.
0b0	0b1	Non-secure.
0b1	0b0	Root.
0b1	0b1	Realm.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSE, bit [62]

Together with MFAR_EL3.NS, reports the physical address space of the access that triggered the exception.

For a description of the values derived by evaluating NS and NSE together, see MFAR_EL3.NS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [61:56]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FPA[55:52], bits [55:52]

When FEAT_D128 is implemented:

When FEAT_D128 is implemented, extension to MFAR_EL3.FPA[47:12].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FPA[51:48], bits [51:48]

When FEAT_LPA is implemented:

When FEAT_LPA is implemented, extension to MFAR_EL3.FPA[47:12].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FPA, bits [47:12]

Bits [47:12] of the Faulting Physical Address.

For implementations with fewer than 48 physical address bits, the corresponding upper bits in this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:0]

Reserved, RES0.

When FEAT_PFAR is implemented and the exception is a synchronous External abort or SError exception:

NS, bit [63]

When FEAT_RME_GDI is implemented:

Together with MFAR_EL3.NSE and MFAR_EL3.NSE2, reports the physical address space of the access that triggered the exception.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

NSE2	NSE	NS	Meaning
0b0	0b0	0b0	When Secure state is implemented, Secure. Otherwise reserved.
0b0	0b0	0b1	Non-secure.
0b0	0b1	0b0	Root.
0b0	0b1	0b1	Realm.
0b1	0b0	0b0	System Agent.
0b1	0b0	0b1	NS Protected.
0b1	0b1	0b0	Reserved.
0b1	0b1	0b1	Reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_RME is implemented:

Together with MFAR_EL3.NSE, reports the physical address space of the access that triggered the exception.

NSE	NS	Meaning
0b0	0b0	When Secure state is implemented, Secure. Otherwise reserved.
0b0	0b1	Non-secure.
0b1	0b0	Root.
0b1	0b1	Realm.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Non-secure. Reports the physical address space of the access that triggered the exception.

NS	Meaning
0b0	Secure physical address space.
0b1	Non-secure physical address space.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSE, bit [62]

When FEAT_RME is implemented:

Together with MFAR_EL3.NS, reports the physical address space of the access that triggered the exception.

For a description of the values derived by evaluating NS and NSE together, see MFAR_EL3.NS.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSE2, bit [61]

When FEAT_RME_GDI is implemented:

Together with MFAR_EL3.NS and MFAR_EL3.NSE, reports the physical address space of the access that triggered the exception.

For a description of the values derived by evaluating NS, NSE, and NSE2 together, see MFAR_EL3.NS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [60:56]

Reserved, RES0.

PA[55:52], bits [55:52]

When FEAT_D128 is implemented:

When FEAT_D128 is implemented, extension to MFAR_EL3.PA[47:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PA[51:48], bits [51:48]

When FEAT_LPA is implemented:

When FEAT_LPA is implemented, extension to MFAR_EL3.PA[47:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PA, bits [47:0]

Physical Address. Bits [47:0] of the aborting physical address.

For implementations with fewer than 48 physical address bits, the corresponding upper bits in this field are RES0.

The recorded address can be any address within the same naturally-aligned fault granule as the faulting physical address, where the size of the fault granule is IMPLEMENTATION DEFINED and no larger than the larger than:

- The size of the range of values permitted to be recorded in FAR_EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing MFAR_EL3

MFAR_EL3 is not valid and reads UNKNOWN if ESR_EL3.EC is recorded indicating an Abort or SError exception and ESR_EL3.PFV is recorded as 0.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MFAR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0110	0b0000	0b101

```
if !((IsFeatureImplemented(FEAT_PFAR) || IsFeatureImplemented(FEAT_RME)) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MFAR_EL3;
```

```
MSR MFAR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0110	0b0000	0b101

```
if !((IsFeatureImplemented(FEAT_PFAR) || IsFeatureImplemented(FEAT_RME)) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    MFAR_EL3 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.137 MIDR_EL1, Main ID Register

The MIDR_EL1 characteristics are:

Purpose

Provides identification information for the PE, including an implementer code for the device and a device ID number.

Configuration

AArch64 System register MIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register MIDR[31:0].

AArch64 System register MIDR_EL1 bits [31:0] are architecturally mapped to External register MIDR_EL1[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to MIDR_EL1 are UNDEFINED.

Attributes

MIDR_EL1 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

Implementer, bits [31:24]

The Implementer code. This field must hold an implementer code that has been assigned by Arm.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Implementer	Meaning
0x00	Reserved for software use.
0x41	Arm Limited.
0x42	Broadcom Corporation.
0x43	Cavium Inc.
0x44	Digital Equipment Corporation.
0x46	Fujitsu Ltd.
0x49	Infineon Technologies AG.
0x4D	Motorola or Freescale Semiconductor Inc.
0x4E	NVIDIA Corporation.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Implementer	Meaning
0x50	Applied Micro Circuits Corporation.
0x51	Qualcomm Inc.
0x56	Marvell International Ltd.
0x69	Intel Corporation.
0xC0	Ampere Computing.

Arm can assign codes that are not published in this manual. All values not assigned by Arm are reserved and must not be used.

Access to this field is RO.

Variant, bits [23:20]

Variant number. Typically, this field is used to distinguish between different product variants, or major revisions of a product.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Architecture, bits [19:16]

Architecture version.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Architecture	Meaning
0b0001	Armv4.
0b0010	Armv4T.
0b0011	Armv5 (obsolete).
0b0100	Armv5T.
0b0101	Armv5TE.
0b0110	Armv5TEJ.
0b0111	Armv6.
0b1111	Architectural features are individually identified in the ID_* registers.

All other values are reserved.

Access to this field is RO.

PartNum, bits [15:4]

Primary Part Number for the device.

On processors implemented by Arm, if the top four bits of the primary part number are 0x0 or 0x7, the variant and architecture are encoded differently.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Revision, bits [3:0]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Revision number for the device.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Accessing MIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MIDR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        ↪HFGRTR_EL2.MIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() then
        X[t, 64] = VPIDR_EL2;
    else
        X[t, 64] = MIDR_EL1;
elsif PSTATE.EL == EL2 then
    X[t, 64] = MIDR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MIDR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.138 MPIDR_EL1, Multiprocessor Affinity Register

The MPIDR_EL1 characteristics are:

Purpose

In a multiprocessor system, provides an additional PE identification mechanism.

Configuration

In a uniprocessor system, Arm recommends that each Aff<n> field of this register returns a value of 0.

AArch64 System register MPIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register MPIDR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to MPIDR_EL1 are UNDEFINED.

Attributes

MPIDR_EL1 is a 64-bit register.

Field descriptions

Bits [63:40]

Reserved, RES0.

Aff3, bits [39:32]

Affinity level 3. See the description of Aff0 for more information.

Aff3 is not supported in AArch32 state.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Bit [31]

Reserved, RES1.

U, bit [30]

Indicates a Uniprocessor system, as distinct from PE 0 in a multiprocessor system.

The value of this field is an IMPLEMENTATION DEFINED choice of:

U	Meaning
0b0	Processor is part of a multiprocessor system.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

U	Meaning
0b1	Processor is part of a uniprocessor system.

Access to this field is RO.

Bits [29:25]

Reserved, RES0.

MT, bit [24]

Indicates whether the lowest level of affinity consists of logical PEs that are implemented using an interdependent approach, such as multithreading. See the description of Aff0 for more information about affinity levels.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MT	Meaning
0b0	Performance of PEs with different affinity level 0 values, and the same values for affinity level 1 and higher, is largely independent.
0b1	Performance of PEs with different affinity level 0 values, and the same values for affinity level 1 and higher, is very interdependent.

Note

This field does not indicate that multithreading is implemented and does not indicate that PEs with different affinity level 0 values, and the same values for affinity level 1 and higher are implemented.

Access to this field is RO.

Aff2, bits [23:16]

Affinity level 2. See the description of Aff0 for more information.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Aff1, bits [15:8]

Affinity level 1. See the description of Aff0 for more information.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Aff0, bits [7:0]

Affinity level 0. The value of the MPIDR.{Aff2, Aff1, Aff0} or MPIDR_EL1.{Aff3, Aff2, Aff1, Aff0} set of fields of each PE must be unique within the system as a whole.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Accessing MPIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, MPIDR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0000	0b101

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
     ↪HFGRTR_EL2.MPIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() then
        X[t, 64] = VMPIDR_EL2;
    else
        X[t, 64] = MPIDR_EL1;
elsif PSTATE.EL == EL2 then
    X[t, 64] = MPIDR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPIDR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.139 MVFR0_EL1, AArch32 Media and VFP Feature Register 0

The MVFR0_EL1 characteristics are:

Purpose

Describes the features provided by the AArch32 Advanced SIMD and floating-point implementation.

Must be interpreted with MVFR1_EL1 and MVFR2_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

In an implementation where at least one Exception level supports execution in AArch32 state, but there is no support for Advanced SIMD and floating-point operation, this register is RAZ.

AArch64 System register MVFR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register MVFR0[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to MVFR0_EL1 are UNDEFINED.

Attributes

MVFR0_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:

Bits [63:32]

Reserved, RES0.

FPRound, bits [31:28]

Floating-Point Rounding modes. Indicates whether the floating-point implementation provides support for rounding modes.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPRound	Meaning
0b0000	Not implemented, or only Round to Nearest mode supported, except that Round towards Zero mode is supported for VCVT instructions that always use that rounding mode regardless of the FPSCR setting.
0b0001	All rounding modes supported.

All other values are reserved.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

In Armv8-A the permitted values are `0b0000` and `0b0001`.

Access to this field is RO.

FPShVec, bits [27:24]

Short Vectors. Indicates whether the floating-point implementation provides support for the use of short vectors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPShVec	Meaning
`0b0000`	Short vectors not supported.
`0b0001`	Short vector operation supported.

All other values are reserved.

In Armv8-A the only permitted value is `0b0000`.

Access to this field is RO.

FPSqrt, bits [23:20]

Square Root. Indicates whether the floating-point implementation provides support for the ARMv6 VFP square root operations.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPSqrt	Meaning
`0b0000`	Not supported in hardware.
`0b0001`	Supported.

All other values are reserved.

In Armv8-A the permitted values are `0b0000` and `0b0001`.

The VSQRT.F32 instruction also requires the single-precision floating-point attribute, bits [7:4], and the VSQRT.F64 instruction also requires the double-precision floating-point attribute, bits [11:8].

Access to this field is RO.

FPDivide, bits [19:16]

Indicates whether the floating-point implementation provides support for VFP divide operations.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPDivide	Meaning
`0b0000`	Not supported in hardware.
`0b0001`	Supported.

All other values are reserved.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

In Armv8-A the permitted values are `0b0000` and `0b0001`.

The VDIV.F32 instruction also requires the single-precision floating-point attribute, bits [7:4], and the VDIV.F64 instruction also requires the double-precision floating-point attribute, bits [11:8].

Access to this field is RO.

FPTrap, bits [15:12]

Floating-Point Exception Trapping. Indicates whether the floating-point implementation provides support for exception trapping.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPTrap	Meaning
`0b0000`	Not supported.
`0b0001`	Supported.

All other values are reserved.

A value of `0b0001` indicates that, when the corresponding trap is enabled, a floating-point exception generates an exception.

Access to this field is RO.

FPDP, bits [11:8]

Floating-Point Double-Precision. Indicates whether the floating-point implementation provides support for double-precision operations.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPDP	Meaning
`0b0000`	Not supported in hardware.
`0b0001`	Supported, VFPv2.
`0b0010`	Supported, VFPv3, VFPv4, or Armv8. VFPv3 and Armv8 add an instruction to load a double-precision floating-point constant, and conversions between double-precision and fixed-point values.

All other values are reserved.

In Armv8-A the permitted values are `0b0000` and `0b0010`.

A value of `0b0001` or `0b0010` indicates support for all VFP double-precision instructions in the supported version of VFP, except that, in addition to this field being nonzero:

- VSQRT.F64 is only available if the Square root field is `0b0001`.
- VDIV.F64 is only available if the Divide field is `0b0001`.
- Conversion between double-precision and single-precision is only available if the single-precision field is nonzero.

Access to this field is RO.

FPSP, bits [7:4]

Floating-Point Single-Precision. Indicates whether the floating-point implementation provides support for single-precision operations.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPSP	Meaning
0b0000	Not supported in hardware.
0b0001	Supported, VFPv2.
0b0010	Supported, VFPv3 or VFPv4. VFPv3 adds an instruction to load a single-precision floating-point constant, and conversions between single-precision and fixed-point values.

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0010.

A value of 0b0001 or 0b0010 indicates support for all VFP single-precision instructions in the supported version of VFP, except that, in addition to this field being nonzero:

- VSQRT.F32 is only available if the Square root field is 0b0001.
- VDIV.F32 is only available if the Divide field is 0b0001.
- Conversion between double-precision and single-precision is only available if the double-precision field is nonzero.

Access to this field is RO.

SIMDReg, bits [3:0]

Advanced SIMD registers. Indicates whether the Advanced SIMD and floating-point implementation provides support for the Advanced SIMD and floating-point register bank.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SIMDReg	Meaning
0b0000	The implementation has no Advanced SIMD and floating-point support.
0b0001	The implementation includes floating-point support with 16 x 64-bit registers.
0b0010	The implementation includes Advanced SIMD and floating-point support with 32 x 64-bit registers.

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0010.

Access to this field is RO.

Otherwise:

Bits [63:0]

Reserved, UNKNOWN.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing MVFR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MVFR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MVFR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MVFR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MVFR0_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.140 MVFR1_EL1, AArch32 Media and VFP Feature Register 1

The MVFR1_EL1 characteristics are:

Purpose

Describes the features provided by the AArch32 Advanced SIMD and floating-point implementation.

Must be interpreted with MVFR0_EL1 and MVFR2_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

In an implementation where at least one Exception level supports execution in AArch32 state, but there is no support for Advanced SIMD and floating-point operation, this register is RAZ.

AArch64 System register MVFR1_EL1 bits [31:0] are architecturally mapped to AArch32 System register MVFR1[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to MVFR1_EL1 are UNDEFINED.

Attributes

MVFR1_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:

Bits [63:32]

Reserved, RES0.

SIMDFMAC, bits [31:28]

Advanced SIMD Fused Multiply-Accumulate. Indicates whether the Advanced SIMD implementation provides fused multiply accumulate instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SIMDFMAC	Meaning
0b0000	Not implemented.
0b0001	Implemented.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The Advanced SIMD and floating-point implementations must provide the same level of support for these instructions.

Access to this field is RO.

FPHP, bits [27:24]

Floating-Point Half-Precision. Indicates the level of half-precision floating-point support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPHP	Meaning
`0b0000`	Not supported.
`0b0001`	Floating-point half-precision conversion instructions are supported for conversion between single-precision and half-precision.
`0b0010`	As for `0b0001`, and adds instructions for conversion between double-precision and half-precision.
`0b0011`	As for `0b0010`, and adds support for half-precision floating-point arithmetic.

All other values are reserved.

In Armv8-A, the permitted values are:

- `0b0000` in an implementation without floating-point support.
- `0b0010` in an implementation with floating-point support that does not include the FEAT_FP16 extension.
- `0b0011` in an implementation with floating-point support that includes the FEAT_FP16 extension.

The level of support indicated by this field must be equivalent to the level of support indicated by the SIMDHP field, meaning the permitted values are:

Half-Precision instructions supported	FPHP	SIMDHP
No support	`0b0000`	`0b0000`
Conversions only	`0b0010`	`0b0001`
Conversions and arithmetic	`0b0011`	`0b0010`

Access to this field is RO.

SIMDHP, bits [23:20]

Advanced SIMD Half-Precision. Indicates the level of half-precision floating-point support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SIMDHP	Meaning
`0b0000`	Not supported.
`0b0001`	SIMD half-precision conversion instructions are supported for conversion between single-precision and half-precision.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

0b0010	As for 0b0001, and adds support for half-precision floating-point arithmetic.

All other values are reserved.

In Armv8-A, the permitted values are:

- 0b0000 in an implementation without SIMD floating-point support.
- 0b0001 in an implementation with SIMD floating-point support that does not include the FEAT_FP16 extension.
- 0b0010 in an implementation with SIMD floating-point support that includes the FEAT_FP16 extension.

The level of support indicated by this field must be equivalent to the level of support indicated by the FPHP field, meaning the permitted values are:

Half-Precision instructions supported	FPHP	SIMDHP
No support	0b0000	0b0000
Conversions only	0b0010	0b0001
Conversions and arithmetic	0b0011	0b0010

Access to this field is RO.

SIMDSP, bits [19:16]

Advanced SIMD Single-Precision. Indicates whether the Advanced SIMD and floating-point implementation provides single-precision floating-point instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SIMDSP	Meaning
0b0000	Not implemented.
0b0001	Implemented. This value is permitted only if the SIMDInt field is 0b0001.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

Access to this field is RO.

SIMDInt, bits [15:12]

Advanced SIMD Integer. Indicates whether the Advanced SIMD and floating-point implementation provides integer instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SIMDInt	Meaning
0b0000	Not implemented.
0b0001	Implemented.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SIMDInt	Meaning

All other values are reserved.

In Armv8-A, the permitted values are `0b0000` and `0b0001`.

Access to this field is RO.

SIMDLS, bits [11:8]

Advanced SIMD Load/Store. Indicates whether the Advanced SIMD and floating-point implementation provides load/store instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SIMDLS	Meaning
`0b0000`	Not implemented.
`0b0001`	Implemented.

All other values are reserved.

In Armv8-A, the permitted values are `0b0000` and `0b0001`.

Access to this field is RO.

FPDNaN, bits [7:4]

Default NaN mode. Indicates whether the floating-point implementation provides support only for the Default NaN mode.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPDNaN	Meaning
`0b0000`	Not implemented, or hardware supports only the Default NaN mode.
`0b0001`	Hardware supports propagation of NaN values.

All other values are reserved.

In Armv8-A, the permitted values are `0b0000` and `0b0001`.

Access to this field is RO.

FPFtZ, bits [3:0]

Flush to Zero mode. Indicates whether the floating-point implementation provides support only for the Flush-to-Zero mode of operation.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FPFtZ	Meaning
0b0000	Not implemented, or hardware supports only the Flush-to-Zero mode of operation.
0b0001	Hardware supports full denormalized number arithmetic.

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

Access to this field is RO.

Otherwise:

Bits [63:0]

Reserved, UNKNOWN.

Accessing MVFR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MVFR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        X[t, 64] = MVFR1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MVFR1_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MVFR1_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.141 MVFR2_EL1, AArch32 Media and VFP Feature Register 2

The MVFR2_EL1 characteristics are:

Purpose

Describes the features provided by the AArch32 Advanced SIMD and floating-point implementation.

Must be interpreted with MVFR0_EL1 and MVFR1_EL1.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

In an implementation where at least one Exception level supports execution in AArch32 state, but there is no support for Advanced SIMD and floating-point operation, this register is RAZ.

AArch64 System register MVFR2_EL1 bits [31:0] are architecturally mapped to AArch32 System register MVFR2[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to MVFR2_EL1 are UNDEFINED.

Attributes

MVFR2_EL1 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented:

Bits [63:8]

Reserved, RES0.

FPMisc, bits [7:4]

Indicates whether the floating-point implementation provides support for miscellaneous VFP features.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPMisc	Meaning
0b0000	Not implemented, or no support for miscellaneous features.
0b0001	Support for Floating-point selection.
0b0010	As 0b0001, and Floating-point Conversion to Integer with Directed Rounding modes.
0b0011	As 0b0010, and Floating-point Round to Integer Floating-point.
0b0100	As 0b0011, and Floating-point MaxNum and MinNum.

All other values are reserved.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

In Armv8-A, the permitted values are `0b0000` and `0b0100`.

Access to this field is RO.

SIMDMisc, bits [3:0]

Indicates whether the Advanced SIMD implementation provides support for miscellaneous Advanced SIMD features.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SIMDMisc	Meaning
`0b0000`	Not implemented, or no support for miscellaneous features.
`0b0001`	Floating-point Conversion to Integer with Directed Rounding modes.
`0b0010`	As `0b0001`, and Floating-point Round to Integer Floating-point.
`0b0011`	As `0b0010`, and Floating-point MaxNum and MinNum.

All other values are reserved.

In Armv8-A, the permitted values are `0b0000` and `0b0011`.

Access to this field is RO.

Otherwise:

Bits [63:0]

Reserved, UNKNOWN.

Accessing MVFR2_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MVFR2_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0011	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MVFR2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1'
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_IDTE3) && SCR_EL3.TID3 == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MVFR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MVFR2_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.142 PAR_EL1, Physical Address Register

The PAR_EL1 characteristics are:

Purpose

Returns the output address (OA) from an Address translation instruction that executed successfully, or fault information if the instruction did not execute successfully.

Configuration

AArch64 System register PAR_EL1 is a 128-bit register that can also be accessed as a 64-bit value. If it is accessed as a 64-bit register, accesses read and write bits [63:0] and do not modify bits [127:64].

Single stage AT Instructions (ATS1*) report their result using the 128-bit format of PAR_EL1 if the translation system that they target uses VMSAv9-128.

ATS12* Instructions report their result using the 128-bit format PAR_EL1 if either of the following is true:

- if stage 2 translations are enabled and the stage 2 translation system uses VMSAv9-128.
- if stage 2 translations are disabled and the stage 1 translation system uses VMSAv9-128.

Otherwise, 64-bit format of PAR_EL1 is used.

AArch64 System register PAR_EL1 bits [63:0] are architecturally mapped to AArch32 System register PAR[63:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to PAR_EL1 are UNDEFINED.

Attributes

PAR_EL1 is a:

- 128-bit register when FEAT_D128 is implemented, GetPAR_EL1_D128() == ‘1’, and GetPAR_EL1_F() == ‘0’.
- 128-bit register when FEAT_D128 is implemented, GetPAR_EL1_D128() == ‘1’, and GetPAR_EL1_F() == ‘1’.
- 128-bit register when FEAT_D128 is implemented, GetPAR_EL1_D128() == ‘0’, and GetPAR_EL1_F() == ‘0’.
- 128-bit register when FEAT_D128 is implemented, GetPAR_EL1_D128() == ‘0’, and GetPAR_EL1_F() == ‘1’.
- 64-bit register when FEAT_D128 is not implemented and GetPAR_EL1_F() == ‘0’.
- 64-bit register when FEAT_D128 is not implemented and GetPAR_EL1_F() == ‘1’.

Field descriptions

When FEAT_D128 is implemented, GetPAR_EL1_D128() == '1', and GetPAR_EL1_F() == '0':

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

This section describes the register value returned by the successful execution of an Address translation instruction. Software might subsequently write a different value to the register, and that write does not affect the operation of the PE.

On a successful conversion, the PAR_EL1 can return a value that indicates the resulting attributes, rather than the values that appear in the Translation table descriptors. More precisely:

- The PAR_EL1.{ATTR, SH} fields are permitted to report the resulting attributes, as determined by any permitted implementation choices and any applicable configuration bits, instead of reporting the values that appear in the Translation table descriptors.
- See the PAR_EL1.NS bit description for constraints on the value it returns.

Bits [127:120]

Reserved, RES0.

PA, bits [119:76]

Output address. The output address (OA) corresponding to the supplied input address. This field returns address bits[55:12].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [75:65]

Reserved, RES0.

D128, bit [64]

Indicates if the PAR_EL1 uses the 128-bit format.

D128	Meaning
`0b1`	PAR_EL1 uses the 128-bit format. PAR_EL1[127:0] holds valid data.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ATTR, bits [63:56]

Memory attributes for the returned output address. This field uses the same encoding as the Attr<n> fields in MAIR_EL1, MAIR_EL2, and MAIR_EL3.

If FEAT_MTE_PERM is implemented and the instruction performed a stage 2 translation, the following additional encoding is defined:

ATTR	Meaning
`0b11100000`	Tagged NoTagAccess Normal Inner Write-Back, Outer Write-Back, Read-Allocate, Write-Allocate Non-transient memory.

Note

This encoding in MAIR_ELx is Reserved.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The value returned in this field can be the resulting attribute that is actually implemented by the implementation, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the Translation table descriptor.

Note

The attributes presented are consistent with the stages of translation applied in the address translation instruction. If the instruction performed a stage 1 translation only, the attributes are from the stage 1 translation. If the instruction performed a stage 1 and stage 2 translation, the attributes are from the combined stage 1 and stage 2 translation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [55:52, 6:4]

Reserved, RES0.

Bits [51:12]

Reserved, RES0.

NSE, bit [11]

When FEAT_RME is implemented:

Reports the NSE attribute for a translation table descriptor from the EL3 translation regime.

For a description of the values derived by evaluating NS and NSE together, see PAR_EL1.NS.

For a result from a Secure, Non-secure, or Realm translation regime, this bit is unknown.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

IMPLEMENTATION DEFINED, bit [10]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NS, bit [9]

When FEAT_RME is implemented:

Non-secure. The NS attribute for a translation table entry from a Secure translation regime, a Realm translation regime, and the EL3 translation regime.

For a result from an EL3 translation regime, NS and NSE are evaluated together to report the physical address space:

NSE	NS	Meaning
0b0	0b0	When Secure state is implemented, Secure. Otherwise reserved.
0b0	0b1	Non-secure.
0b1	0b0	Root.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

NSE	NS	Meaning
0b1	0b1	Realm.

For a result from a Secure translation regime, when SCR_EL3.EEL2 is 1, this bit distinguishes between the Secure and Non-secure intermediate physical address space of the translation for the instructions:

- In AArch64 state: AT S1E1R, AT S1E1W, AT S1E1RP, AT S1E1WP, AT S1E0R, and AT S1E0W.
- In AArch32 state: ATS1CPR, ATS1CPW, ATS1CPRP, ATS1CPWP, ATS1CUR, and ATS1CUW.

Otherwise, this bit reflects the Security state of the physical address space of the translation. This means it reflects the effect of the NSTable bits of earlier levels of the translation table walk if those NSTable bits have an effect on the translation.

For a result from a Non-secure translation regime, this bit is UNKNOWN.

For a result from an S1E1 or S1E0 operation on the Realm EL1&0 translation regime, this bit is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Non-secure. The NS attribute for a translation table entry from a Secure translation regime.

For a result from a Secure translation regime, when SCR_EL3.EEL2 is 1, this bit distinguishes between the Secure and Non-secure intermediate physical address space of the translation for the instructions:

- In AArch64 state: AT S1E1R, AT S1E1W, AT S1E1RP, AT S1E1WP, AT S1E0R, and AT S1E0W.
- In AArch32 state: ATS1CPR, ATS1CPW, ATS1CPRP, ATS1CPWP, ATS1CUR, and ATS1CUW.

Otherwise, this bit reflects the Security state of the physical address space of the translation. This means it reflects the effect of the NSTable bits of earlier levels of the translation table walk if those NSTable bits have an effect on the translation.

For a result from a Non-secure translation regime, this bit is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH, bits [8:7]

Shareability attribute, for the returned output address.

SH	Meaning
0b00	Non-shareable.
0b10	Outer Shareable.
0b11	Inner Shareable.

The value 0b01 is reserved.

Note

This field returns the value 0b10 for:

- Any type of Device memory.
- Normal memory with both Inner Non-cacheable and Outer Non-cacheable attributes.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The value returned in this field can be the resulting attribute, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the Translation table descriptor.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [3:1]

Reserved, RES0.

F, bit [0]

Indicates whether the instruction performed a successful address translation.

F	Meaning
0b0	Address translation completed successfully.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_D128 is implemented, GetPAR_EL1_D128() == '1', and GetPAR_EL1_F() == '1':

This section describes the register value returned by a fault on the execution of an Address translation instruction. Software might subsequently write a different value to the register, and that write does not affect the operation of the PE.

Bits [127:65]

Reserved, RES0.

D128, bit [64]

Indicates if the PAR_EL1 uses the 128-bit format.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D128	Meaning
0b1	PAR_EL1 uses the 128-bit format. PAR_EL1[127:0] holds valid data.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [63:56]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [55:52]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [51:48]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [47:16]

Reserved, RES0.

DirtyBit, bit [15]

When FEAT_S1PIE is implemented or FEAT_S2PIE is implemented:

DirtyBit flag.

If PAR_EL1.FST indicates a Permission fault for a stage of translation that is using Indirect Permissions, and dirty state is managed by software, then this field holds information about the fault.

DirtyBit	Meaning
0b0	The Permission Fault is not due to dirty state.
0b1	The Permission Fault is due to dirty state.

For any other fault or Access, this field is RES0.

Note

At stage 1, dirty state is indicated by the nDirty bit in Block and Page descriptors. At stage 2, dirty state is indicated by the Dirty bit in Block and Page descriptors.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

Overlay, bit [14]

When FEAT_S1POE is implemented or FEAT_S2POE is implemented:

Overlay flag.

If PAR_EL1.FST indicates a Permission fault for a stage of translation, then this field holds information about the fault.

Overlay	Meaning
`0b0`	The Data Abort is not due to Overlay Permissions.
`0b1`	The Data Abort is due to Overlay Permissions.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TopLevel, bit [13]

When FEAT_THE is implemented:

Fault due to TopLevel. Indicates if the fault was due to TopLevel.

TopLevel	Meaning
`0b0`	Fault is not due to TopLevel.
`0b1`	Fault is due to TopLevel.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AssuredOnly, bit [12]

When FEAT_THE is implemented:

AssuredOnly flag.

If PAR_EL1.S indicates a stage 2 fault, then this field holds information about the fault.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

AssuredOnly	Meaning
0b0	The Data Abort is not due to AssuredOnly.
0b1	The Data Abort is due to AssuredOnly.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [11]

Reserved, RES1.

Bit [10]

Reserved, RES0.

S, bit [9]

Indicates the translation stage at which the translation aborted:

S	Meaning
0b0	Translation aborted because of a fault in the stage 1 translation.
0b1	Translation aborted because of a fault in the stage 2 translation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PTW, bit [8]

If this bit is set to 1, it indicates the translation aborted because of a stage 2 fault during a stage 1 translation table walk.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [7]

Reserved, RES0.

FST, bits [6:1]

Fault status code, as shown in the Data Abort ESR encoding.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FST	Meaning	Applies when
0b000000	Address size fault, level 0 of translation or translation table base register.	
0b000001	Address size fault, level 1.	
0b000010	Address size fault, level 2.	
0b000011	Address size fault, level 3.	
0b000100	Translation fault, level 0.	
0b000101	Translation fault, level 1.	
0b000110	Translation fault, level 2.	
0b000111	Translation fault, level 3.	
0b001001	Access flag fault, level 1.	
0b001010	Access flag fault, level 2.	
0b001011	Access flag fault, level 3.	
0b001000	Access flag fault, level 0.	FEAT_LPA2 is implemented
0b001100	Permission fault, level 0.	FEAT_LPA2 is implemented
0b001101	Permission fault, level 1.	
0b001110	Permission fault, level 2.	
0b001111	Permission fault, level 3.	
0b011011	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1.	FEAT_LPA2 is implemented and FEAT_RAS is not implemented
0b011100	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 0.	FEAT_RAS is not implemented
0b011101	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 1.	FEAT_RAS is not implemented
0b011110	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 2.	FEAT_RAS is not implemented
0b011111	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 3.	FEAT_RAS is not implemented
0b100010	Granule Protection Fault on translation table walk or hardware update of translation table, level -2.	FEAT_D128 is implemented and FEAT_RME is implemented
0b100011	Granule Protection Fault on translation table walk or hardware update of translation table, level -1.	FEAT_RME is implemented and FEAT_LPA2 is implemented
0b100100	Granule Protection Fault on translation table walk or hardware update of translation table, level 0.	FEAT_RME is implemented
0b100101	Granule Protection Fault on translation table walk or hardware update of translation table, level 1.	FEAT_RME is implemented
0b100110	Granule Protection Fault on translation table walk or hardware update of translation table, level 2.	FEAT_RME is implemented
0b100111	Granule Protection Fault on translation table walk or hardware update of translation table, level 3.	FEAT_RME is implemented
0b101000	Granule Protection Fault, not on translation table walk or hardware update of translation table.	FEAT_RME is implemented

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FST	Meaning	Applies when
0b101001	Address size fault, level -1.	FEAT_LPA2 is implemented
0b101010	Translation fault, level -2.	FEAT_D128 is implemented
0b101011	Translation fault, level -1.	FEAT_LPA2 is implemented
0b101100	Address Size fault, level -2.	FEAT_D128 is implemented
0b110000	TLB conflict abort.	
0b110001	Unsupported atomic hardware update fault.	FEAT_HAFDBS is implemented
0b111101	Section Domain fault, from an AArch32 stage 1 EL1&0 translation regime using Short-descriptor translation table format.	FEAT_AA32EL1 is implemented
0b111110	Page Domain fault, from an AArch32 stage 1 EL1&0 translation regime using Short-descriptor translation table format.	FEAT_AA32EL1 is implemented

Note

The encodings for FST do not include Synchronous External abort on translation table walk or hardware update of translation table, because these MMU faults are reported as a Data Abort exception instead of being recorded in PAR_EL1. See Exceptions to reporting the fault in PAR_EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [0]

Indicates whether the instruction performed a successful address translation.

F	Meaning
0b1	Address translation aborted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_D128 is implemented, GetPAR_EL1_D128() == '0', and GetPAR_EL1_F() == '0':

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

This section describes the register value returned by the successful execution of an Address translation instruction. Software might subsequently write a different value to the register, and that write does not affect the operation of the PE.

On a successful conversion, the PAR_EL1 can return a value that indicates the resulting attributes, rather than the values that appear in the Translation table descriptors. More precisely:

- The PAR_EL1.{ATTR, SH} fields are permitted to report the resulting attributes, as determined by any permitted implementation choices and any applicable configuration bits, instead of reporting the values that appear in the Translation table descriptors.
- See the PAR_EL1.NS bit description for constraints on the value it returns.

Bits [127:65]

Reserved, RES0.

D128, bit [64]

Indicates if the PAR_EL1 uses the 128-bit format.

D128	Meaning
0b0	PAR_EL1 uses the 64-bit format. PAR_EL1[63:0] holds valid data.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ATTR, bits [63:56]

Memory attributes for the returned output address. This field uses the same encoding as the Attr<n> fields in MAIR_EL1, MAIR_EL2, and MAIR_EL3.

If FEAT_MTE_PERM is implemented and the instruction performed a stage 2 translation, the following additional encoding is defined:

ATTR	Meaning
0b11100000	Tagged NoTagAccess Normal Inner Write-Back, Outer Write-Back, Read-Allocate, Write-Allocate Non-transient memory.

Note

This encoding in MAIR_ELx is Reserved.

The value returned in this field can be the resulting attribute that is actually implemented by the implementation, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the Translation table descriptor.

Note

The attributes presented are consistent with the stages of translation applied in the address translation instruction. If the instruction performed a stage 1 translation only, the attributes are from the stage 1 translation. If the instruction performed a stage 1 and stage 2 translation, the attributes are from the combined stage 1 and stage 2 translation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bits [55:52, 6:4]

Reserved, RES0.

PA[51:48], bits [51:48]

When FEAT_LPA is implemented:

Extension to PA[47:12]. For more information, see PA[47:12].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PA[47:12], bits [47:12]

Output address. The output address (OA) corresponding to the supplied input address. This field returns address bits[47:12].

When FEAT_LPA is implemented and 52-bit addresses are in use, PA[51:48] forms the upper part of the address value. Otherwise, when 52-bit addresses are not in use, PA[51:48] is RES0.

For implementations with fewer than 48 physical address bits, the corresponding upper bits in this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSE, bit [11]

When FEAT_RME is implemented:

Reports the NSE attribute for a translation table entry from the EL3 translation regime.

For a description of the values derived by evaluating NS and NSE together, see PAR_EL1.NS.

For a result from a Secure, Non-secure, or Realm translation regime, this bit is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

IMPLEMENTATION DEFINED, bit [10]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NS, bit [9]

When FEAT_RME is implemented:

Non-secure. The NS attribute for a translation table entry from a Secure translation regime, a Realm translation regime, and the EL3 translation regime.

For a result from an EL3 translation regime, NS and NSE are evaluated together to report the physical address space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

NSE	NS	Meaning
0b0	0b0	When Secure state is implemented, Secure. Otherwise reserved.
0b0	0b1	Non-secure.
0b1	0b0	Root.
0b1	0b1	Realm.

For a result from a Secure translation regime, when SCR_EL3.EEL2 is 1, this bit distinguishes between the Secure and Non-secure intermediate physical address space of the translation for the instructions:

- In AArch64 state: AT S1E1R, AT S1E1W, AT S1E1RP, AT S1E1WP, AT S1E0R, and AT S1E0W.
- In AArch32 state: ATS1CPR, ATS1CPW, ATS1CPRP, ATS1CPWP, ATS1CUR, and ATS1CUW.

Otherwise, this bit reflects the Security state of the physical address space of the translation. This means it reflects the effect of the NSTable bits of earlier levels of the translation table walk if those NSTable bits have an effect on the translation.

For a result from a Non-secure translation regime, this bit is UNKNOWN.

For a result from an S1E1 or S1E0 operation on the Realm EL1&0 translation regime, this bit is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Non-secure. The NS attribute for a translation table entry from a Secure translation regime.

For a result from a Secure translation regime, when SCR_EL3.EEL2 is 1, this bit distinguishes between the Secure and Non-secure intermediate physical address space of the translation for the instructions:

- In AArch64 state: AT S1E1R, AT S1E1W, AT S1E1RP, AT S1E1WP, AT S1E0R, and AT S1E0W.
- In AArch32 state: ATS1CPR, ATS1CPW, ATS1CPRP, ATS1CPWP, ATS1CUR, and ATS1CUW.

Otherwise, this bit reflects the Security state of the physical address space of the translation. This means it reflects the effect of the NSTable bits of earlier levels of the translation table walk if those NSTable bits have an effect on the translation.

For a result from a Non-secure translation regime, this bit is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH, bits [8:7]

Shareability attribute, for the returned output address.

SH	Meaning
0b00	Non-shareable.
0b10	Outer Shareable.
0b11	Inner Shareable.

The value 0b01 is reserved.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Note

This field returns the value `0b10` for:

- Any type of Device memory.
- Normal memory with both Inner Non-cacheable and Outer Non-cacheable attributes.

The value returned in this field can be the resulting attribute, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the Translation table descriptor.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [3:1]

Reserved, RES0.

F, bit [0]

Indicates whether the instruction performed a successful address translation.

F	Meaning
`0b0`	Address translation completed successfully.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_D128 is implemented, GetPAR_EL1_D128() == '0', and GetPAR_EL1_F() == '1':

This section describes the register value returned by a fault on the execution of an Address translation instruction. Software might subsequently write a different value to the register, and that write does not affect the operation of the PE.

Bits [127:65]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D128, bit [64]

Indicates if the PAR_EL1 uses the 128-bit format.

D128	Meaning
0b0	PAR_EL1 uses the 64-bit format. PAR_EL1[63:0] holds valid data.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [63:56]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [55:52]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [51:48]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [47:16]

Reserved, RES0.

DirtyBit, bit [15]

When FEAT_S1PIE is implemented or FEAT_S2PIE is implemented:

DirtyBit flag.

If PAR_EL1.FST indicates a Permission fault for a stage of translation that is using Indirect Permissions, and dirty state is managed by software, then this field holds information about the fault.

DirtyBit	Meaning
0b0	The Permission Fault is not due to nDirty State or Dirty State.
0b1	The Permission Fault is due to nDirty State or Dirty State.

For any other fault or Access, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

Overlay, bit [14]

When FEAT_S1POE is implemented or FEAT_S2POE is implemented:

Overlay flag.

If PAR_EL1.FST indicates a Permission fault for a stage of translation, then this field holds information about the fault.

Overlay	Meaning
`0b0`	The Data Abort is not due to Overlay Permissions.
`0b1`	The Data Abort is due to Overlay Permissions.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TopLevel, bit [13]

When FEAT_THE is implemented:

Fault due to TopLevel. Indicates if the fault was due to TopLevel.

TopLevel	Meaning
`0b0`	Fault is not due to TopLevel.
`0b1`	Fault is due to TopLevel.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AssuredOnly, bit [12]

When FEAT_THE is implemented:

AssuredOnly flag.

If PAR_EL1.S indicates a stage 2 fault, then this field holds information about the fault.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

AssuredOnly	Meaning
0b0	The Data Abort is not due to AssuredOnly.
0b1	The Data Abort is due to AssuredOnly.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [11]

Reserved, RES1.

Bit [10]

Reserved, RES0.

S, bit [9]

Indicates the translation stage at which the translation aborted:

S	Meaning
0b0	Translation aborted because of a fault in the stage 1 translation.
0b1	Translation aborted because of a fault in the stage 2 translation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PTW, bit [8]

If this bit is set to 1, it indicates the translation aborted because of a stage 2 fault during a stage 1 translation table walk.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [7]

Reserved, RES0.

FST, bits [6:1]

Fault status code, as shown in the Data Abort exception ESR encoding.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FST	Meaning	Applies when
0b000000	Address size fault, level 0 of translation or translation table base register.	
0b000001	Address size fault, level 1.	
0b000010	Address size fault, level 2.	
0b000011	Address size fault, level 3.	
0b000100	Translation fault, level 0.	
0b000101	Translation fault, level 1.	
0b000110	Translation fault, level 2.	
0b000111	Translation fault, level 3.	
0b001001	Access flag fault, level 1.	
0b001010	Access flag fault, level 2.	
0b001011	Access flag fault, level 3.	
0b001000	Access flag fault, level 0.	FEAT_LPA2 is implemented
0b001100	Permission fault, level 0.	FEAT_LPA2 is implemented
0b001101	Permission fault, level 1.	
0b001110	Permission fault, level 2.	
0b001111	Permission fault, level 3.	
0b011011	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1.	FEAT_LPA2 is implemented and FEAT_RAS is not implemented
0b011100	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 0.	FEAT_RAS is not implemented
0b011101	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 1.	FEAT_RAS is not implemented
0b011110	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 2.	FEAT_RAS is not implemented
0b011111	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 3.	FEAT_RAS is not implemented
0b100010	Granule Protection Fault on translation table walk or hardware update of translation table, level -2.	FEAT_D128 is implemented and FEAT_RME is implemented
0b100011	Granule Protection Fault on translation table walk or hardware update of translation table, level -1.	FEAT_RME is implemented and FEAT_LPA2 is implemented
0b100100	Granule Protection Fault on translation table walk or hardware update of translation table, level 0.	FEAT_RME is implemented
0b100101	Granule Protection Fault on translation table walk or hardware update of translation table, level 1.	FEAT_RME is implemented
0b100110	Granule Protection Fault on translation table walk or hardware update of translation table, level 2.	FEAT_RME is implemented
0b100111	Granule Protection Fault on translation table walk or hardware update of translation table, level 3.	FEAT_RME is implemented
0b101000	Granule Protection Fault, not on translation table walk or hardware update of translation table.	FEAT_RME is implemented

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FST	Meaning	Applies when
0b101001	Address size fault, level -1.	FEAT_LPA2 is implemented
0b101010	Translation fault, level -2.	FEAT_D128 is implemented
0b101011	Translation fault, level -1.	FEAT_LPA2 is implemented
0b101100	Address Size fault, level -2.	FEAT_D128 is implemented
0b110000	TLB conflict abort.	
0b110001	Unsupported atomic hardware update fault.	FEAT_HAFDBS is implemented
0b111101	Section Domain fault, from an AArch32 stage 1 EL1&0 translation regime using Short-descriptor translation table format.	FEAT_AA32EL1 is implemented
0b111110	Page Domain fault, from an AArch32 stage 1 EL1&0 translation regime using Short-descriptor translation table format.	FEAT_AA32EL1 is implemented

Note

The encodings for FST do not include Synchronous External abort on translation table walk or hardware update of translation table, because these MMU faults are reported as a Data Abort exception instead of being recorded in PAR_EL1. See Exceptions to reporting the fault in PAR_EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [0]

Indicates whether the instruction performed a successful address translation.

F	Meaning
0b1	Address translation aborted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_D128 is not implemented and GetPAR_EL1_F() == '0':

This section describes the register value returned by the successful execution of an Address translation instruction. Software might subsequently write a different value to the register, and that write does not affect the operation of the PE.

On a successful conversion, the PAR_EL1 can return a value that indicates the resulting attributes, rather than the values that appear in the Translation table descriptors. More precisely:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- The PAR_EL1.{ATTR, SH} fields are permitted to report the resulting attributes, as determined by any permitted implementation choices and any applicable configuration bits, instead of reporting the values that appear in the Translation table descriptors.
- See the PAR_EL1.NS bit description for constraints on the value it returns.

ATTR, bits [63:56]

Memory attributes for the returned output address. This field uses the same encoding as the Attr<n> fields in MAIR_EL1, MAIR_EL2, and MAIR_EL3.

If FEAT_MTE_PERM is implemented and the instruction performed a stage 2 translation, the following additional encoding is defined:

ATTR	Meaning
`0b11100000`	Tagged NoTagAccess Normal Inner Write-Back, Outer Write-Back, Read-Allocate, Write-Allocate Non-transient memory.

Note

This encoding in MAIR_ELx is Reserved.

The value returned in this field can be the resulting attribute that is actually implemented by the implementation, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the Translation table descriptor.

Note

The attributes presented are consistent with the stages of translation applied in the address translation instruction. If the instruction performed a stage 1 translation only, the attributes are from the stage 1 translation. If the instruction performed a stage 1 and stage 2 translation, the attributes are from the combined stage 1 and stage 2 translation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [55:52, 6:4]

Reserved, RES0.

PA[51:48], bits [51:48]

When FEAT_LPA is implemented:

Extension to PA[47:12]. For more information, see PA[47:12].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PA[47:12], bits [47:12]

Output address. The output address (OA) corresponding to the supplied input address. This field returns address bits[47:12].

When FEAT_LPA is implemented and 52-bit addresses are in use, PA[51:48] forms the upper part of the address value. Otherwise, when 52-bit addresses are not in use, PA[51:48] is RES0.

For implementations with fewer than 48 physical address bits, the corresponding upper bits in this field are RES0.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSE, bit [11]

When FEAT_RME is implemented:

Reports the NSE attribute for a translation table entry from the EL3 translation regime.

For a description of the values derived by evaluating NS and NSE together, see PAR_EL1.NS.

For a result from a Secure, Non-secure, or Realm translation regime, this bit is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

IMPLEMENTATION DEFINED, bit [10]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NS, bit [9]

When FEAT_RME is implemented:

Non-secure. The NS attribute for a translation table entry from a Secure translation regime, a Realm translation regime, and the EL3 translation regime.

For a result from an EL3 translation regime, NS and NSE are evaluated together to report the physical address space:

NSE	NS	Meaning
0b0	0b0	When Secure state is implemented, Secure. Otherwise reserved.
0b0	0b1	Non-secure.
0b1	0b0	Root.
0b1	0b1	Realm.

For a result from a Secure translation regime, when SCR_EL3.EEL2 is 1, this bit distinguishes between the Secure and Non-secure intermediate physical address space of the translation for the instructions:

- In AArch64 state: AT S1E1R, AT S1E1W, AT S1E1RP, AT S1E1WP, AT S1E0R, and AT S1E0W.
- In AArch32 state: ATS1CPR, ATS1CPW, ATS1CPRP, ATS1CPWP, ATS1CUR, and ATS1CUW.

Otherwise, this bit reflects the Security state of the physical address space of the translation. This means it reflects the effect of the NSTable bits of earlier levels of the translation table walk if those NSTable bits have an effect on the translation.

For a result from a Non-secure translation regime, this bit is UNKNOWN.

For a result from an S1E1 or S1E0 operation on the Realm EL1&0 translation regime, this bit is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Non-secure. The NS attribute for a translation table entry from a Secure translation regime.

For a result from a Secure translation regime, when SCR_EL3.EEL2 is 1, this bit distinguishes between the Secure and Non-secure intermediate physical address space of the translation for the instructions:

- In AArch64 state: AT S1E1R, AT S1E1W, AT S1E1RP, AT S1E1WP, AT S1E0R, and AT S1E0W.
- In AArch32 state: ATS1CPR, ATS1CPW, ATS1CPRP, ATS1CPWP, ATS1CUR, and ATS1CUW.

Otherwise, this bit reflects the Security state of the physical address space of the translation. This means it reflects the effect of the NSTable bits of earlier levels of the translation table walk if those NSTable bits have an effect on the translation.

For a result from a Non-secure translation regime, this bit is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH, bits [8:7]

Shareability attribute, for the returned output address.

SH	Meaning
`0b00`	Non-shareable.
`0b10`	Outer Shareable.
`0b11`	Inner Shareable.

The value `0b01` is reserved.

Note

This field returns the value `0b10` for:

- Any type of Device memory.
- Normal memory with both Inner Non-cacheable and Outer Non-cacheable attributes.

The value returned in this field can be the resulting attribute, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the Translation table descriptor.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [3:1]

Reserved, RES0.

F, bit [0]

Indicates whether the instruction performed a successful address translation.

F	Meaning
`0b0`	Address translation completed successfully.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_D128 is not implemented and GetPAR_EL1_F() == '1':

This section describes the register value returned by a fault on the execution of an Address translation instruction. Software might subsequently write a different value to the register, and that write does not affect the operation of the PE.

IMPLEMENTATION DEFINED, bits [63:56]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [55:52]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [51:48]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [47:16]

Reserved, RES0.

DirtyBit, bit [15]

When FEAT_S1PIE is implemented or FEAT_S2PIE is implemented:

DirtyBit flag.

If PAR_EL1.FST indicates a Permission fault for a stage of translation that is using Indirect Permissions, and dirty state is managed by software, then this field holds information about the fault.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

DirtyBit	Meaning
0b0	The Permission Fault is not due to nDirty State or Dirty State.
0b1	The Permission Fault is due to nDirty State or Dirty State.

For any other fault or Access, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Overlay, bit [14]

When FEAT_S1POE is implemented or FEAT_S2POE is implemented:

Overlay flag.

If PAR_EL1.FST indicates a Permission fault for a stage of translation, then this field holds information about the fault.

Overlay	Meaning
0b0	The Data Abort is not due to Overlay Permissions.
0b1	The Data Abort is due to Overlay Permissions.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TopLevel, bit [13]

When FEAT_THE is implemented:

Fault due to TopLevel. Indicates if the fault was due to TopLevel.

TopLevel	Meaning
0b0	Fault is not due to TopLevel.
0b1	Fault is due to TopLevel.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

AssuredOnly, bit [12]

When FEAT_THE is implemented:

AssuredOnly flag.

If PAR_EL1.S indicates a stage 2 fault, then this field holds information about the fault.

AssuredOnly	Meaning
0b0	The Data Abort is not due to AssuredOnly.
0b1	The Data Abort is due to AssuredOnly.

For any other fault, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [11]

Reserved, RES1.

Bit [10]

Reserved, RES0.

S, bit [9]

Indicates the translation stage at which the translation aborted:

S	Meaning
0b0	Translation aborted because of a fault in the stage 1 translation.
0b1	Translation aborted because of a fault in the stage 2 translation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PTW, bit [8]

If this bit is set to 1, it indicates the translation aborted because of a stage 2 fault during a stage 1 translation table walk.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [7]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FST, bits [6:1]

Fault status code, as shown in the Data Abort exception ESR encoding.

FST	Meaning	Applies when
0b000000	Address size fault, level 0 of translation or translation table base register.	
0b000001	Address size fault, level 1.	
0b000010	Address size fault, level 2.	
0b000011	Address size fault, level 3.	
0b000100	Translation fault, level 0.	
0b000101	Translation fault, level 1.	
0b000110	Translation fault, level 2.	
0b000111	Translation fault, level 3.	
0b001001	Access flag fault, level 1.	
0b001010	Access flag fault, level 2.	
0b001011	Access flag fault, level 3.	
0b001000	Access flag fault, level 0.	FEAT_LPA2 is implemented
0b001100	Permission fault, level 0.	FEAT_LPA2 is implemented
0b001101	Permission fault, level 1.	
0b001110	Permission fault, level 2.	
0b001111	Permission fault, level 3.	
0b011011	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1.	FEAT_LPA2 is implemented and FEAT_RAS is not implemented
0b011100	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 0.	FEAT_RAS is not implemented
0b011101	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 1.	FEAT_RAS is not implemented
0b011110	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 2.	FEAT_RAS is not implemented
0b011111	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 3.	FEAT_RAS is not implemented
0b100010	Granule Protection Fault on translation table walk or hardware update of translation table, level -2.	FEAT_D128 is implemented and FEAT_RME is implemented
0b100011	Granule Protection Fault on translation table walk or hardware update of translation table, level -1.	FEAT_RME is implemented and FEAT_LPA2 is implemented
0b100100	Granule Protection Fault on translation table walk or hardware update of translation table, level 0.	FEAT_RME is implemented
0b100101	Granule Protection Fault on translation table walk or hardware update of translation table, level 1.	FEAT_RME is implemented

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FST	Meaning	Applies when
0b100110	Granule Protection Fault on translation table walk or hardware update of translation table, level 2.	FEAT_RME is implemented
0b100111	Granule Protection Fault on translation table walk or hardware update of translation table, level 3.	FEAT_RME is implemented
0b101000	Granule Protection Fault, not on translation table walk or hardware update of translation table.	FEAT_RME is implemented
0b101001	Address size fault, level -1.	FEAT_LPA2 is implemented
0b101010	Translation fault, level -2.	FEAT_D128 is implemented
0b101011	Translation fault, level -1.	FEAT_LPA2 is implemented
0b101100	Address Size fault, level -2.	FEAT_D128 is implemented
0b110000	TLB conflict abort.	
0b110001	Unsupported atomic hardware update fault.	FEAT_HAFDBS is implemented
0b111101	Section Domain fault, from an AArch32 stage 1 EL1&0 translation regime using Short-descriptor translation table format.	FEAT_AA32EL1 is implemented
0b111110	Page Domain fault, from an AArch32 stage 1 EL1&0 translation regime using Short-descriptor translation table format.	FEAT_AA32EL1 is implemented

Note

The encodings for FST do not include Synchronous External abort on translation table walk or hardware update of translation table, because these MMU faults are reported as a Data Abort exception instead of being recorded in PAR_EL1. See Exceptions to reporting the fault in PAR_EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [0]

Indicates whether the instruction performed a successful address translation.

F	Meaning
0b1	Address translation aborted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PAR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PAR_EL1
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b000	0b0111	0b0100	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        ↪HFGRTR_EL2.PAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = PAR_EL1<63:0>;
elsif PSTATE.EL == EL2 then
    X[t, 64] = PAR_EL1<63:0>;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PAR_EL1<63:0>;
```

MSR PAR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0111	0b0100	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        ↪HFGWTR_EL2.PAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        PAR_EL1<63:0> = X[t, 64];
elsif PSTATE.EL == EL2 then
    PAR_EL1<63:0> = X[t, 64];
elsif PSTATE.EL == EL3 then
    PAR_EL1<63:0> = X[t, 64];
```

When FEAT_D128 is implemented MRRS <Xt>, <Xt+1>, PAR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0111	0b0100	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
        ↪&& HFGRTR_EL2.PAR_EL1 == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            AArch64.SystemAccessTrap(EL2, 0x14);
        elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.D128En == '0') then
            AArch64.SystemAccessTrap(EL2, 0x14);
        elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x14);
        else
            X[t, t2, 128] = PAR_EL1<127:0>;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x14);
        else
            X[t, t2, 128] = PAR_EL1<127:0>;
    elsif PSTATE.EL == EL3 then
        X[t, t2, 128] = PAR_EL1<127:0>;
```

When FEAT_D128 is implemented MSRR PAR_EL1, <Xt>, <Xt+1>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0111	0b0100	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
         ↪&& HFGWTR_EL2.PAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.D128En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        PAR_EL1<127:0> = X[t, t2, 128];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        PAR_EL1<127:0> = X[t, t2, 128];
elsif PSTATE.EL == EL3 then
    PAR_EL1<127:0> = X[t, t2, 128];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.143 PFAR_EL1, Physical Fault Address Register (EL1)

The PFAR_EL1 characteristics are:

Purpose

Records the faulting physical address for a synchronous External abort, or SError exception taken to EL1.

Configuration

This register is present only when FEAT_PFAR is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PFAR_EL1 are UNDEFINED.

Attributes

PFAR_EL1 is a 64-bit register.

Field descriptions

NS, bit [63]

When FEAT_RME is implemented:

Together with PFAR_EL1.NSE, reports the physical address space of the access that triggered the exception.

NSE	NS	Meaning
0b0	0b0	When Secure state is implemented, Secure. Otherwise reserved.
0b0	0b1	Non-secure.
0b1	0b0	Reserved.
0b1	0b1	Realm.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When EL3 is implemented or FEAT_Secure is implemented:

Non-secure. Reports the physical address space of the access that triggered the exception.

NS	Meaning
0b0	Secure physical address space.
0b1	Non-secure physical address space.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSE, bit [62]

When FEAT_RME is implemented:

Together with PFAR_EL1.NS, reports the physical address space of the access that triggered the exception.

For a description of the values derived by evaluating NS and NSE together, see MFAR_EL3.NS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [61:56]

Reserved, RES0.

PA[55:52], bits [55:52]

When FEAT_D128 is implemented:

When FEAT_D128 is implemented, extension to PFAR_EL1.PA[47:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PA[51:48], bits [51:48]

When FEAT_LPA is implemented:

When FEAT_LPA is implemented, extension to PFAR_EL1.PA[47:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PA, bits [47:0]

Physical Address. Bits [47:0] of the aborting physical address.

For implementations with fewer than 48 physical address bits, the corresponding upper bits in this field are RES0.

The recorded address can be any address within the same naturally-aligned fault granule as the faulting physical address, where the size of the fault granule is IMPLEMENTATION DEFINED and no larger than the larger than:

- The size of the range of values permitted to be recorded in FAR_EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing PFAR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name `PFAR_EL1` or `PFAR_EL12` are not guaranteed to be ordered with respect to accesses using the other accessor name.

PFAR_EL1 is not valid and reads UNKNOWN if ESR_EL1.PFV is recorded as 0.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PFAR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0110	0b0000	0b101

```
if !(IsFeatureImplemented(FEAT_PFAR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PFAREn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HFGRTR2_EL2.nPFAR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PFAREn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x2D0];
    else
        X[t, 64] = PFAR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PFAREn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PFAREn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = PFAR_EL2;
    else
        X[t, 64] = PFAR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PFAR_EL1;
```

```
MSR PFAR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0110	0b0000	0b101

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_PFAR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PFAREn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HFGWTR2_EL2.nPFAR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PFAREn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x2D0] = X[t, 64];
    else
        PFAR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PFAREn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PFAREn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        PFAR_EL2 = X[t, 64];
    else
        PFAR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PFAR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, PFAR_EL12

op0	op1	CRn	CRm	op2
0b11	0b101	0b0110	0b0000	0b101

```
if !(IsFeatureImplemented(FEAT_PFAR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x2D0];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PFAREn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.PFAREn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            X[t, 64] = PFAR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = PFAR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented MSR PFAR_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b0110	0b0000	0b101

```
if !(IsFeatureImplemented(FEAT_PFAR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x2D0] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PFAREn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.PFAREn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PFAR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        PFAR_EL1 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.144 PFAR_EL2, Physical Fault Address Register (EL2)

The PFAR_EL2 characteristics are:

Purpose

Records the faulting physical address for a synchronous External abort, or SError exception taken to EL2.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_PFAR is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PFAR_EL2 are UNDEFINED.

Attributes

PFAR_EL2 is a 64-bit register.

Field descriptions

NS, bit [63]

When FEAT_RME_GDI is implemented:

Together with PFAR_EL2.NSE and PFAR_EL2.NSE2, reports the physical address space of the access that triggered the exception.

NSE2	NSE	NS	Meaning
0b0	0b0	0b0	When Secure state is implemented, Secure. Otherwise reserved.
0b0	0b0	0b1	Non-secure.
0b0	0b1	0b0	Reserved.
0b0	0b1	0b1	Realm.
0b1	0b0	0b0	System Agent.
0b1	0b0	0b1	NS Protected.
0b1	0b1	0b0	Reserved.
0b1	0b1	0b1	Reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_RME is implemented:

Together with PFAR_EL2.NSE, reports the physical address space of the access that triggered the exception.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

NSE	NS	Meaning
0b0	0b0	When Secure state is implemented, Secure. Otherwise reserved.
0b0	0b1	Non-secure.
0b1	0b0	Reserved.
0b1	0b1	Realm.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When EL3 is implemented or FEAT_Secure is implemented:

Non-secure. Reports the physical address space of the access that triggered the exception.

NS	Meaning
0b0	Secure physical address space.
0b1	Non-secure physical address space.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSE, bit [62]

When FEAT_RME is implemented:

Together with PFAR_EL2.NS, reports the physical address space of the access that triggered the exception.

For a description of the values derived by evaluating NS and NSE together, see MFAR_EL3.NS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSE2, bit [61]

When FEAT_RME_GDI is implemented:

Together with PFAR_EL2.NS and PFAR_EL2.NSE, reports the physical address space of the access that triggered the exception.

For a description of the values derived by evaluating NS, NSE, and NSE2 together, see PFAR_EL2.NS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bits [60:56]

Reserved, RES0.

PA[55:52], bits [55:52]

When FEAT_D128 is implemented:

When FEAT_D128 is implemented, extension to PFAR_EL2.PA[47:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PA[51:48], bits [51:48]

When FEAT_LPA is implemented:

When FEAT_LPA is implemented, extension to PFAR_EL2.PA[47:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PA, bits [47:0]

Physical Address. Bits [47:0] of the aborting physical address.

For implementations with fewer than 48 physical address bits, the corresponding upper bits in this field are RES0.

The recorded address can be any address within the same naturally-aligned fault granule as the faulting physical address, where the size of the fault granule is IMPLEMENTATION DEFINED and no larger than the larger than:

- The size of the range of values permitted to be recorded in FAR_EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PFAR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name `PFAR_EL2` or `PFAR_EL1` are not guaranteed to be ordered with respect to accesses using the other accessor name.

PFAR_EL2 is not valid and reads UNKNOWN if ESR_EL2.PFV is recorded as 0.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PFAR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0110	0b0000	0b101

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_PFAR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PFAREn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PFAREn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PFAR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PFAR_EL2;
```

```
MSR PFAR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0110	0b0000	0b101

```
if !(IsFeatureImplemented(FEAT_PFAR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PFAREn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PFAREn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PFAR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PFAR_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.145 PIR_EL1, Permission Indirection Register 1 (EL1)

The PIR_EL1 characteristics are:

Purpose

Stage 1 Permission Indirection Register for privileged access of the EL1&0 translation regime.

Configuration

This register is present only when FEAT_S1PIE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PIR_EL1 are UNDEFINED.

Attributes

PIR_EL1 is a 64-bit register.

Field descriptions

Perm<m> , bits [4m+3:4m], for m = 15 to 0

Represents stage 1 Base Permissions.

Perm<m>	Meaning
0b0000	No access. Overlay applied.
0b0001	Read. Overlay applied.
0b0010	Execute. Overlay applied.
0b0011	Read and Execute. Overlay applied.
0b0100	Reserved - treated as No access. Overlay applied.
0b0101	Read and Write. Overlay applied.
0b0110	Read, Write, and Execute. Overlay applied. WXN control applied.
0b0111	Read, Write, and Execute. Overlay applied.
0b1000	Read. Overlay not applied.
0b1001	Read, GCS Read, and GCS Write. Overlay not applied.
0b1010	Read and Execute. Overlay not applied.
0b1011	Reserved - treated as No access. Overlay not applied.
0b1100	Read and Write. Overlay not applied.
0b1101	Reserved - treated as No access. Overlay not applied.
0b1110	Read, Write, and Execute. Overlay not applied.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

0b1111	Reserved - treated as No access. Overlay not applied.

This field is permitted to be cached in a TLB.

When stage 1 Indirect Permission mechanism is disabled, this register is ignored.

Unless otherwise specified, the WXN control is not applied.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PIR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name PIR_EL1 or PIR_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PIR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.nPIR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x2A0];
    else
        X[t, 64] = PIR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = PIR_EL2;
    else
        X[t, 64] = PIR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PIR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

MSR PIR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.nPIR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x2A0] = X[t, 64];
    else
        PIR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        PIR_EL2 = X[t, 64];
    else
        PIR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PIR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, PIR_EL12

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x2A0];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if ELIsInHost(EL2) then
            if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
                UNDEFINED;
            elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
                if EL3SDDUndef() then
                    UNDEFINED;
                else
                    AArch64.SystemAccessTrap(EL3, 0x18);
            else
                X[t, 64] = PIR_EL1;
        else
            UNDEFINED;
    elsif PSTATE.EL == EL3 then
        if ELIsInHost(EL2) then
            X[t, 64] = PIR_EL1;
        else
            UNDEFINED;
```

When FEAT_VHE is implemented MSR PIR_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x2A0] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PIR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        PIR_EL1 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.146 PIR_EL2, Permission Indirection Register 2 (EL2)

The PIR_EL2 characteristics are:

Purpose

Stage 1 Permission Indirection Register for privileged access of the EL2 or EL2&0 translation regime.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_S1PIE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PIR_EL2 are UNDEFINED.

Attributes

PIR_EL2 is a 64-bit register.

Field descriptions

Perm<m> , bits [4m+3:4m], for m = 15 to 0

Represents stage 1 Base Permissions.

Perm<m>	Meaning
0b0000	No access. Overlay applied.
0b0001	Read. Overlay applied.
0b0010	Execute. Overlay applied.
0b0011	Read and Execute. Overlay applied.
0b0100	Reserved - treated as No access. Overlay applied.
0b0101	Read and Write. Overlay applied.
0b0110	Read, Write, and Execute. Overlay applied. WXN control applied.
0b0111	Read, Write, and Execute. Overlay applied.
0b1000	Read. Overlay not applied.
0b1001	Read, GCS Read, and GCS Write. Overlay not applied.
0b1010	Read and Execute. Overlay not applied.
0b1011	Reserved - treated as No access. Overlay not applied.
0b1100	Read and Write. Overlay not applied.
0b1101	Reserved - treated as No access. Overlay not applied.
0b1110	Read, Write, and Execute. Overlay not applied.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Perm<m>	Meaning
0b1111	Reserved - treated as No access. Overlay not applied.

This field is permitted to be cached in a TLB.

When stage 1 Indirect Permission mechanism is disabled, this register is ignored.

Unless otherwise specified, the WXN control is not applied.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PIR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name PIR_EL2 or PIR_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PIR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PIR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PIR_EL2;
```

```
MSR PIR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0010	0b011

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PIR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PIR_EL2 = X[t, 64];
```

MRS <Xt>, PIR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.nPIR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x2A0];
    else
        X[t, 64] = PIR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = PIR_EL2;
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        X[t, 64] = PIR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PIR_EL1;
```

```
MSR PIR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.nPIR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x2A0] = X[t, 64];
    else
        PIR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        PIR_EL2 = X[t, 64];
    else
        PIR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PIR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.147 PIR_EL3, Permission Indirection Register 3 (EL3)

The PIR_EL3 characteristics are:

Purpose

Stage 1 Permission Indirection Register for privileged access of the EL3 translation regime.

Configuration

This register is present only when FEAT_S1PIE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PIR_EL3 are UNDEFINED.

Attributes

PIR_EL3 is a 64-bit register.

Field descriptions

Perm<m> , bits [4m+3:4m], for m = 15 to 0

Represents stage 1 Base Permissions.

Perm<m>	Meaning
0b0000	No access. Overlay applied.
0b0001	Read. Overlay applied.
0b0010	Execute. Overlay applied.
0b0011	Read and Execute. Overlay applied.
0b0100	Reserved - treated as No access. Overlay applied.
0b0101	Read and Write. Overlay applied.
0b0110	Read, Write, and Execute. Overlay applied. WXN control applied.
0b0111	Read, Write, and Execute. Overlay applied.
0b1000	Read. Overlay not applied.
0b1001	Read, GCS Read, and GCS Write. Overlay not applied.
0b1010	Read and Execute. Overlay not applied.
0b1011	Reserved - treated as No access. Overlay not applied.
0b1100	Read and Write. Overlay not applied.
0b1101	Reserved - treated as No access. Overlay not applied.
0b1110	Read, Write, and Execute. Overlay not applied.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

0b1111	Reserved - treated as No access. Overlay not applied.

This field is permitted to be cached in a TLB.

When stage 1 Indirect Permission mechanism is disabled, this register is ignored.

Unless otherwise specified, the WXN control is not applied.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PIR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PIR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PIR_EL3;
```

```
MSR PIR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.PIR_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PIR_EL3 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.148 PIRE0_EL1, Permission Indirection Register 0 (EL1)

The PIRE0_EL1 characteristics are:

Purpose

Stage 1 Permission Indirection Register for unprivileged access of the EL1&0 translation regime.

Configuration

This register is present only when FEAT_S1PIE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PIRE0_EL1 are UNDEFINED.

Attributes

PIRE0_EL1 is a 64-bit register.

Field descriptions

Perm<m> , bits [4m+3:4m], for m = 15 to 0

Represents stage 1 Base Permissions.

Perm<m>	Meaning
0b0000	No access. Overlay applied.
0b0001	Read. Overlay applied.
0b0010	Execute. Overlay applied.
0b0011	Read and Execute. Overlay applied.
0b0100	Reserved - treated as No access. Overlay applied.
0b0101	Read and Write. Overlay applied.
0b0110	Read, Write, and Execute. Overlay applied. WXN control applied.
0b0111	Read, Write, and Execute. Overlay applied.
0b1000	Read. Overlay not applied.
0b1001	Read, GCS Read, and GCS Write. Overlay not applied.
0b1010	Read and Execute. Overlay not applied.
0b1011	Reserved - treated as No access. Overlay not applied.
0b1100	Read and Write. Overlay not applied.
0b1101	Reserved - treated as No access. Overlay not applied.
0b1110	Read, Write, and Execute. Overlay not applied.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

0b1111	Reserved - treated as No access. Overlay not applied.

This field is permitted to be cached in a TLB.

When stage 1 Indirect Permission mechanism is disabled, this register is ignored.

Unless otherwise specified, the WXN control is not applied.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PIRE0_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name PIRE0_EL1 or PIRE0_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PIRE0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b010

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.nPIRE0_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x290];
    else
        X[t, 64] = PIRE0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = PIRE0_EL2;
    else
        X[t, 64] = PIRE0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PIRE0_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MSR PIRE0_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b010

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.nPIRE0_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x290] = X[t, 64];
    else
        PIRE0_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        PIRE0_EL2 = X[t, 64];
    else
        PIRE0_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PIRE0_EL1 = X[t, 64];
```

```
When FEAT_VHE is implemented MRS <Xt>, PIRE0_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0010	0b010

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x290];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if ELIsInHost(EL2) then
            if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
                UNDEFINED;
            elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
                if EL3SDDUndef() then
                    UNDEFINED;
                else
                    AArch64.SystemAccessTrap(EL3, 0x18);
            else
                X[t, 64] = PIRE0_EL1;
        else
            UNDEFINED;
    elsif PSTATE.EL == EL3 then
        if ELIsInHost(EL2) then
            X[t, 64] = PIRE0_EL1;
        else
            UNDEFINED;
```

When FEAT_VHE is implemented MSR PIRE0_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0010	0b010

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x290] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PIRE0_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        PIRE0_EL1 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.149 PIRE0_EL2, Permission Indirection Register 0 (EL2)

The PIRE0_EL2 characteristics are:

Purpose

Stage 1 Permission Indirection Register for unprivileged access of the EL2&0 translation regime.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_S1PIE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PIRE0_EL2 are UNDEFINED.

Attributes

PIRE0_EL2 is a 64-bit register.

Field descriptions

Perm<m> , bits [4m+3:4m], for m = 15 to 0

Represents stage 1 Base Permissions.

Perm<m>	Meaning
0b0000	No access. Overlay applied.
0b0001	Read. Overlay applied.
0b0010	Execute. Overlay applied.
0b0011	Read and Execute. Overlay applied.
0b0100	Reserved - treated as No access. Overlay applied.
0b0101	Read and Write. Overlay applied.
0b0110	Read, Write, and Execute. Overlay applied. WXN control applied.
0b0111	Read, Write, and Execute. Overlay applied.
0b1000	Read. Overlay not applied.
0b1001	Read, GCS Read, and GCS Write. Overlay not applied.
0b1010	Read and Execute. Overlay not applied.
0b1011	Reserved - treated as No access. Overlay not applied.
0b1100	Read and Write. Overlay not applied.
0b1101	Reserved - treated as No access. Overlay not applied.
0b1110	Read, Write, and Execute. Overlay not applied.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Perm<m>	Meaning
0b1111	Reserved - treated as No access. Overlay not applied.

This field is permitted to be cached in a TLB.

When stage 1 Indirect Permission mechanism is disabled, this register is ignored.

Unless otherwise specified, the WXN control is not applied.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PIRE0_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name PIRE0_EL2 or PIRE0_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PIRE0_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0010	0b010

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PIRE0_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PIRE0_EL2;
```

```
MSR PIRE0_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0010	0b010

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PIRE0_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PIRE0_EL2 = X[t, 64];
```

MRS <Xt>, PIRE0_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b010

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.nPIRE0_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x290];
    else
        X[t, 64] = PIRE0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = PIRE0_EL2;
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        X[t, 64] = PIRE0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PIRE0_EL1;
```

```
MSR PIRE0_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b010

```
if !(IsFeatureImplemented(FEAT_S1PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.nPIRE0_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x290] = X[t, 64];
    else
        PIRE0_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        PIRE0_EL2 = X[t, 64];
    else
        PIRE0_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PIRE0_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.150 POR_EL0, Permission Overlay Register 0 (EL0)

The POR_EL0 characteristics are:

Purpose

Stage 1 Permission Overlay Register for unprivileged access of EL1&0 or EL2&0 translation regime.

Configuration

This register is present only when FEAT_S1POE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to POR_EL0 are UNDEFINED.

Attributes

POR_EL0 is a 64-bit register.

Field descriptions

Perm<m> , bits [4m+3:4m], for m = 15 to 0

Perm Represents stage 1 Overlay Permissions.

Perm<m>	Meaning
0b0000	No access.
0b0001	Read.
0b0010	Execute.
0b0011	Read, Execute.
0b0100	Write.
0b0101	Write, Read.
0b0110	Write, Execute.
0b0111	Read, Write, Execute.
0b1xxx	Reserved - treated as No access

If FEAT_D128 is implemented and VMSAv9-128 is in use, then fields Perm[8] to Perm[15] are used for POIndex values 8 to 15.

Otherwise, the fields Perm[8] to Perm[15] are RES0.

This field is not permitted to be cached in a TLB.

When the stage 1 Overlay mechanism is disabled, this field is IGNORED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing POR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, POR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1010	0b0010	0b100

```
if !(IsFeatureImplemented(FEAT_S1POE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL0) && CPACR_EL1.E0POE == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HFGRTR_EL2.nPOR_EL0 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CPTR_EL2.E0POE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = POR_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.nPOR_EL0 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = POR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = POR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = POR_EL0;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MSR POR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1010	0b0010	0b100

```
if !(IsFeatureImplemented(FEAT_S1POE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL0) && CPACR_EL1.E0POE == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HFGWTR_EL2.nPOR_EL0 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CPTR_EL2.E0POE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        POR_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.nPOR_EL0 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        POR_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        POR_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    POR_EL0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.151 POR_EL1, Permission Overlay Register 1 (EL1)

The POR_EL1 characteristics are:

Purpose

Stage 1 Permission Overlay Register for privileged access of the EL1&0 translation regime.

Configuration

This register is present only when FEAT_S1POE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to POR_EL1 are UNDEFINED.

Attributes

POR_EL1 is a 64-bit register.

Field descriptions

Perm<m> , bits [4m+3:4m], for m = 15 to 0

Perm Represents stage 1 Overlay Permissions.

Perm<m>	Meaning
0b0000	No access.
0b0001	Read.
0b0010	Execute.
0b0011	Read, Execute.
0b0100	Write.
0b0101	Write, Read.
0b0110	Write, Execute.
0b0111	Read, Write, Execute.
0b1xxx	Reserved - treated as No access

If FEAT_D128 is implemented and VMSAv9-128 is in use, then fields Perm[8] to Perm[15] are used for POIndex values 8 to 15.

Otherwise, the fields Perm[8] to Perm[15] are RES0.

This field is not permitted to be cached in a TLB.

When the stage 1 Overlay mechanism is disabled, this field is IGNORED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing POR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name `POR_EL1` or `POR_EL12` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, POR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b100

```
if !(IsFeatureImplemented(FEAT_S1POE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.nPOR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x2A8];
    else
        X[t, 64] = POR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = POR_EL2;
    else
        X[t, 64] = POR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = POR_EL1;
```

```
MSR POR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b100

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_S1POE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.nPOR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x2A8] = X[t, 64];
    else
        POR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        POR_EL2 = X[t, 64];
    else
        POR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    POR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, POR_EL12

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0010	0b100

```
if !(IsFeatureImplemented(FEAT_S1POE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x2A8];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = POR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = POR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented MSR POR_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0010	0b100

```
if !(IsFeatureImplemented(FEAT_S1POE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x2A8] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            POR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        POR_EL1 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.152 POR_EL2, Permission Overlay Register 2 (EL2)

The POR_EL2 characteristics are:

Purpose

Stage 1 Permission Overlay Register for privileged access of the EL2 or EL2&0 translation regime.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_S1POE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to POR_EL2 are UNDEFINED.

Attributes

POR_EL2 is a 64-bit register.

Field descriptions

Perm<m> , bits [4m+3:4m], for m = 15 to 0

Perm Represents stage 1 Overlay Permissions.

Perm<m>	Meaning
0b0000	No access.
0b0001	Read.
0b0010	Execute.
0b0011	Read, Execute.
0b0100	Write.
0b0101	Write, Read.
0b0110	Write, Execute.
0b0111	Read, Write, Execute.
0b1xxx	Reserved - treated as No access

If FEAT_D128 is implemented and VMSAv9-128 is in use, then fields Perm[8] to Perm[15] are used for POIndex values 8 to 15.

Otherwise, the fields Perm[8] to Perm[15] are RES0.

This field is not permitted to be cached in a TLB.

When the stage 1 Overlay mechanism is disabled, this field is IGNORED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing POR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name `POR_EL2` or `POR_EL1` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, POR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0010	0b100

```
if !(IsFeatureImplemented(FEAT_S1POE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = POR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = POR_EL2;
```

```
MSR POR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0010	0b100

```
if !(IsFeatureImplemented(FEAT_S1POE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        POR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    POR_EL2 = X[t, 64];
```

MRS <Xt>, POR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b100

```
if !(IsFeatureImplemented(FEAT_S1POE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
     ↪HFGRTR_EL2.nPOR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x2A8];
    else
        X[t, 64] = POR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = POR_EL2;
    else
        X[t, 64] = POR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = POR_EL1;
```

MSR POR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b100

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_S1POE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.nPOR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x2A8] = X[t, 64];
    else
        POR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        POR_EL2 = X[t, 64];
    else
        POR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    POR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.153 POR_EL3, Permission Overlay Register 3 (EL3)

The POR_EL3 characteristics are:

Purpose

Stage 1 Permission Overlay Register for privileged access of the EL3 translation regime.

Configuration

This register is present only when FEAT_S1POE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to POR_EL3 are UNDEFINED.

Attributes

POR_EL3 is a 64-bit register.

Field descriptions

Perm<m> , bits [4m+3:4m], for m = 15 to 0

Perm Represents stage 1 Overlay Permissions.

Perm<m>	Meaning
0b0000	No access.
0b0001	Read.
0b0010	Execute.
0b0011	Read, Execute.
0b0100	Write.
0b0101	Write, Read.
0b0110	Write, Execute.
0b0111	Read, Write, Execute.
0b1xxx	Reserved - treated as No access

If FEAT_D128 is implemented and VMSAv9-128 is in use, then fields Perm[8] to Perm[15] are used for POIndex values 8 to 15.

Otherwise, the fields Perm[8] to Perm[15] are RES0.

This field is not permitted to be cached in a TLB.

When stage 1 Overlay mechanism is disabled, this register is ignored.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing POR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, POR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b0010	0b100

```
if !(IsFeatureImplemented(FEAT_S1POE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = POR_EL3;
```

```
MSR POR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b0010	0b100

```
if !(IsFeatureImplemented(FEAT_S1POE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    POR_EL3 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.154 RCWMASK_EL1, Read Check Write Instruction Mask (EL1)

The RCWMASK_EL1 characteristics are:

Purpose

Contains the mask used by RCW instructions.

Configuration

RCWMASK_EL1 is a 128-bit register that can also be accessed as a 64-bit value. If it is accessed as a 64-bit register, accesses read and write bits [63:0] and do not modify bits [127:64].

This register is present only when FEAT_THE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to RCWMASK_EL1 are UNDEFINED.

Attributes

RCWMASK_EL1 is a:

- 128-bit register when FEAT_D128 is implemented.
- 64-bit register otherwise.

Field descriptions

When FEAT_D128 is implemented:

RCWMASK, bits [127:0]

Mask used to decide which bit-fields are writable to the 128-bit Descriptor by RCW or RCWS instructions.

If RCWMASK_EL1 is indirectly read by 128-bit variants of RCW or RCWS instructions:

- The Effective value of RCWMASK[n] is the same as RCWMASK_EL1[n], except as follows:
  - If n >= 17, and n <= 55, the Effective value of RCWMASK[n] is the same as RCWMASK_EL1[16].
  - If n is in {126:125, 120:119, 114, 107:101, 90:56, 1:0}, the Effective value of RCWMASK[n] is 0.
  - If n >= 121, n <= 124, and FEAT_S1POE is not implemented, the Effective value of RCWMASK[n] is 0.
  - If FEAT_MEC is not implemented, the Effective value of RCWMASK[108] is 0.

If RCWMASK_EL1 is indirectly read by 64-bit variants of RCW or RCWS instructions:

- The Effective value of RCWMASK[n] is the same as RCWMASK_EL1[n], except as follows:
  - If n >= 18, and n <= 49, the Effective value of RCWMASK[n] is the same as RCWMASK_EL1[17].

RCWMASK_EL1 register bits {126:125, 120:119, 114, 107:101, 90:64, 52, 49:18, 0} are RES0.

If FEAT_S1POE is not implemented, RCWMASK_EL1 register bits {124:121} are RES0.

If FEAT_MEC is not implemented, RCWMASK_EL1[108] is RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RCWMASK, bits [63:0]

Mask used to decide which bit-fields are writable to the 64-bit Descriptor by RCW or RCWS Instructions.

The Effective value of RCWMASK[n] is the same as RCWMASK_EL1[n], except as follows:

- If n >= 18, and n <= 49, the Effective value of RCWMASK[n] is the same as RCWMASK_EL1[17].

RCWMASK_EL1 register bits {52, 49:18, 0} are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing RCWMASK_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, RCWMASK_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b110

```
if !(IsFeatureImplemented(FEAT_THE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.RCWMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.nRCWMASK_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.RCWMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = RCWMASK_EL1<63:0>;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.RCWMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.RCWMASKEn == '0' then
        if EL3SDDUndef() then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = RCWMASK_EL1<63:0>;
elsif PSTATE.EL == EL3 then
    X[t, 64] = RCWMASK_EL1<63:0>;
```

MSR RCWMASK_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b110

```
if !(IsFeatureImplemented(FEAT_THE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.RCWMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.nRCWMASK_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.RCWMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        RCWMASK_EL1<63:0> = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.RCWMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.RCWMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        RCWMASK_EL1<63:0> = X[t, 64];
elsif PSTATE.EL == EL3 then
    RCWMASK_EL1<63:0> = X[t, 64];
```

When FEAT_D128 is implemented MRRS <Xt>, <Xt+1>, RCWMASK_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b110

```
if !(IsFeatureImplemented(FEAT_THE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.RCWMASKEn == '0' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            UNDEFINED;
        elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
            UNDEFINED;
        elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
             ↪&& HFGRTR_EL2.nRCWMASK_EL1 == '0' then
            AArch64.SystemAccessTrap(EL2, 0x14);
        elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.D128En == '0') then
            AArch64.SystemAccessTrap(EL2, 0x14);
        elsif HaveEL(EL3) && SCR_EL3.RCWMASKEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x14);
        elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x14);
        else
            X[t, t2, 128] = RCWMASK_EL1<127:0>;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.RCWMASKEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.RCWMASKEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x14);
        elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x14);
        else
            X[t, t2, 128] = RCWMASK_EL1<127:0>;
    elsif PSTATE.EL == EL3 then
        X[t, t2, 128] = RCWMASK_EL1<127:0>;
```

When FEAT_D128 is implemented MSRR RCWMASK_EL1, <Xt>, <Xt+1>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b110

```
    if !(IsFeatureImplemented(FEAT_THE) && IsFeatureImplemented(FEAT_AA64)) then
        UNDEFINED;
    elsif PSTATE.EL == EL0 then
        UNDEFINED;
    elsif PSTATE.EL == EL1 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.RCWMASKEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
            UNDEFINED;
        elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
             ↪&& HFGWTR_EL2.nRCWMASK_EL1 == '0' then
            AArch64.SystemAccessTrap(EL2, 0x14);
        elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.D128En == '0') then
            AArch64.SystemAccessTrap(EL2, 0x14);
        elsif HaveEL(EL3) && SCR_EL3.RCWMASKEn == '0' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        RCWMASK_EL1<127:0> = X[t, t2, 128];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.RCWMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.RCWMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        RCWMASK_EL1<127:0> = X[t, t2, 128];
elsif PSTATE.EL == EL3 then
    RCWMASK_EL1<127:0> = X[t, t2, 128];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.155 RCWSMASK_EL1, Software Read Check Write Instruction Mask (EL1)

The RCWSMASK_EL1 characteristics are:

Purpose

Contains the software mask used by RCWS instructions.

Configuration

RCWSMASK_EL1 is a 128-bit register that can also be accessed as a 64-bit value. If it is accessed as a 64-bit register, accesses read and write bits [63:0] and do not modify bits [127:64].

This register is present only when FEAT_THE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to RCWSMASK_EL1 are UNDEFINED.

Attributes

RCWSMASK_EL1 is a:

- 128-bit register when FEAT_D128 is implemented.
- 64-bit register otherwise.

Field descriptions

When FEAT_D128 is implemented:

RCWSMASK, bits [127:0]

Software Mask used to decide which bit-fields are writable to the 128-bit Descriptor by RCWS instructions.

If RCWSMASK_EL1 is indirectly read by 128-bit variants of RCWS instructions:

- The Effective value of RCWSMASK[n] is the same as RCWSMASK_EL1[n], except as follows:
  - If n >= 17, and n <= 55, the Effective value of RCWSMASK[n] is the same as RCWSMASK_EL1[16].
  - If n is in {126:125, 120:119, 114, 107:101, 90:56, 1:0}, the Effective value of RCWSMASK[n] is 0.
  - If n >= 121, n <= 124, and FEAT_S1POE is not implemented, the Effective value of RCWSMASK[n] is 0.
  - If FEAT_MEC is not implemented, the Effective value of RCWSMASK bit RCWSMASK_EL1[108] is 0.

If RCWSMASK_EL1 is indirectly read by 64-bit variants of RCWS instructions:

- The Effective value of RCWSMASK[n] is the same as RCWSMASK_EL1[n], except as follows:
  - If n >= 18, and n <= 49, the Effective value of RCWSMASK[n] is the same as RCWSMASK_EL1[17].
  - If n == 52 and Protection is enabled, the Effective value of RCWSMASK[52] is 0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

RCWSMASK_EL1 register bits {126:125, 120:119, 114, 107:101, 90:64, 49:18, 0} are RES0.

If FEAT_S1POE is not implemented, RCWSMASK_EL1 register bits {124:121} are RES0.

If FEAT_MEC is not implemented, RCWSMASK_EL1[108] is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RCWSMASK, bits [63:0]

Software Mask used to decide which bit-fields are writable to the 64-bit Descriptor by RCWS Instruction.

The Effective value of RCWSMASK[n] is the same as RCWSMASK_EL1[n], except as follows

- If n >= 18, and n <= 49, the Effective value of RCWSMASK[n] is the same as RCWSMASK_EL1[17].
- If n == 52 and Protection is enabled, the Effective value of RCWSMASK[52] is 0.

RCWSMASK_EL1 register bits {49:18, 0} are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing RCWSMASK_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, RCWSMASK_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_THE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.RCWMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HFGRTR2_EL2.nRCWSMASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.RCWMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    else
        X[t, 64] = RCWSMASK_EL1<63:0>;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.RCWMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.RCWMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = RCWSMASK_EL1<63:0>;
elsif PSTATE.EL == EL3 then
    X[t, 64] = RCWSMASK_EL1<63:0>;
```

MSR RCWSMASK_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_THE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.RCWMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HFGWTR2_EL2.nRCWSMASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.RCWMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        RCWSMASK_EL1<63:0> = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.RCWMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.RCWMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        RCWSMASK_EL1<63:0> = X[t, 64];
elsif PSTATE.EL == EL3 then
    RCWSMASK_EL1<63:0> = X[t, 64];
```

When FEAT_D128 is implemented MRRS <Xt>, <Xt+1>, RCWSMASK_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b011

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_THE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.RCWMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
         ↪'0') || HFGRTR2_EL2.nRCWSMASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.D128En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.RCWMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        X[t, t2, 128] = RCWSMASK_EL1<127:0>;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.RCWMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.RCWMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        X[t, t2, 128] = RCWSMASK_EL1<127:0>;
elsif PSTATE.EL == EL3 then
    X[t, t2, 128] = RCWSMASK_EL1<127:0>;
```

When FEAT_D128 is implemented MSRR RCWSMASK_EL1, <Xt>, <Xt+1>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_THE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.RCWMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
        ↪'0') || HFGWTR2_EL2.nRCWSMASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.D128En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.RCWMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        RCWSMASK_EL1<127:0> = X[t, t2, 128];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.RCWMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.RCWMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        RCWSMASK_EL1<127:0> = X[t, t2, 128];
elsif PSTATE.EL == EL3 then
    RCWSMASK_EL1<127:0> = X[t, t2, 128];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.156 REVIDR_EL1, Revision ID Register

The REVIDR_EL1 characteristics are:

Purpose

Provides implementation-specific minor revision information.

Configuration

If REVIDR_EL1 has the same value as MIDR_EL1, then its contents have no significance.

AArch64 System register REVIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register REVIDR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to REVIDR_EL1 are UNDEFINED.

Attributes

REVIDR_EL1 is a 64-bit register.

Field descriptions

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Accessing REVIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, REVIDR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0000	0b110

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.REVIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = REVIDR_EL1;
elsif PSTATE.EL == EL2 then
    X[t, 64] = REVIDR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = REVIDR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.157 RGSR_EL1, Random Allocation Tag Seed Register.

The RGSR_EL1 characteristics are:

Purpose

Random Allocation Tag Seed Register.

Configuration

When GCR_EL1.RRND==`0b1`, updates to RGSR_EL1 are implementation-specific.

Direct and indirect reads and writes to the register appear to occur in program order relative to other instructions, without the need for any explicit synchronization.

This register is present only when FEAT_MTE2 is implemented. Otherwise, direct accesses to RGSR_EL1 are UNDEFINED.

Attributes

RGSR_EL1 is a 64-bit register.

Field descriptions

When GCR_EL1.RRND == '0':

Bits [63:24]

Reserved, RES0.

SEED, bits [23:8]

Seed register used for generating values returned by RandomTag().

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [7:4]

Reserved, RES0.

TAG, bits [3:0]

Tag generated by the most recent IRG instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bits [63:56]

Reserved, RES0.

IMPLEMENTATION DEFINED, bits [55:8]

IMPLEMENTATION DEFINED.

Note

Software is recommended to avoid writing SEED[15:0] with a value of zero, unless this has been generated by the PE in response to an earlier value with SEED being nonzero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [7:4]

Reserved, RES0.

TAG, bits [3:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing RGSR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, RGSR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0000	0b101

```
if !IsFeatureImplemented(FEAT_MTE2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1')
        ↪then
        UNDEFINED;
    elsif EL2Enabled() && !ELIsInHost(EL0) && !(IsFeatureImplemented(FEAT_MTE2) && HCR_EL2.ATA == '1')
        ↪then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1') then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = RGSR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1')
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1') then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = RGSR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = RGSR_EL1;
```

```
MSR RGSR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0000	0b101

```
if !IsFeatureImplemented(FEAT_MTE2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1')
     ↪then
        UNDEFINED;
    elsif EL2Enabled() && !ELIsInHost(EL0) && !(IsFeatureImplemented(FEAT_MTE2) && HCR_EL2.ATA == '1')
     ↪then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1') then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        RGSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1')
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1') then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        RGSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    RGSR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.158 RMR_EL1, Reset Management Register (EL1)

The RMR_EL1 characteristics are:

Purpose

When this register is implemented:

- A write to the register at EL1 can request a Warm reset.
- If EL1 can use all Execution states, this register specifies the Execution state that the PE boots into on a Warm reset.

Configuration

When EL1 is the highest implemented Exception level:

- If EL1 can use all Execution states then this register must be implemented.
- If EL1 cannot use AArch32 then it is IMPLEMENTATION DEFINED whether the register is implemented.

When the highest implemented Exception level is EL1, AArch64 System register RMR_EL1 bits [31:0] are architecturally mapped to AArch32 System register RMR[31:0].

This register is present only when the highest implemented Exception level is EL1 and FEAT_AA64 is implemented. Otherwise, direct accesses to RMR_EL1 are UNDEFINED.

Attributes

RMR_EL1 is a 64-bit register.

Field descriptions

Bits [63:2]

Reserved, RES0.

RR, bit [1]

Reset Request. Setting this bit to 1 requests a Warm reset.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

AA64, bit [0]

When EL1 can use AArch32, determines which Execution state the PE boots into after a Warm reset:

AA64	Meaning
`0b0`	AArch32.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

AA64	Meaning
0b1	AArch64.

On coming out of the Warm reset, execution starts at the IMPLEMENTATION DEFINED reset vector address of the specified Execution state.

If EL1 can only use AArch64 state, this bit is RAO/WI.

The reset behavior of this field is:

- On a Cold reset, this field resets to '1'.

Accessing RMR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, RMR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b0000	0b010

```
if !(IsHighestEL(EL1) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL1 && IsHighestEL(EL1) then
    X[t, 64] = RMR_EL1;
else
    UNDEFINED;
```

```
MSR RMR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b0000	0b010

```
if !(IsHighestEL(EL1) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL1 && IsHighestEL(EL1) then
    RMR_EL1 = X[t, 64];
else
    UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.159 RMR_EL2, Reset Management Register (EL2)

The RMR_EL2 characteristics are:

Purpose

When this register is implemented:

- A write to the register at EL2 can request a Warm reset.
- If EL2 can use all Execution states, this register specifies the Execution state that the PE boots into on a Warm reset.

Configuration

When EL2 is the highest implemented Exception level:

- If EL2 can use all Execution states then this register must be implemented.
- If EL2 cannot use AArch32 then it is IMPLEMENTATION DEFINED whether the register is implemented.

If EL2 is not implemented, this register is RES0 from EL3.

When the highest implemented Exception level is EL2, AArch64 System register RMR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HRMR[31:0].

This register is present only when the highest implemented Exception level is EL2 and FEAT_AA64 is implemented. Otherwise, direct accesses to RMR_EL2 are UNDEFINED.

Attributes

RMR_EL2 is a 64-bit register.

Field descriptions

Bits [63:2]

Reserved, RES0.

RR, bit [1]

Reset Request. Setting this bit to 1 requests a Warm reset.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

AA64, bit [0]

When EL2 can use AArch32, determines which Execution state the PE boots into after a Warm reset:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

AA64	Meaning
0b0	AArch32.
0b1	AArch64.

On coming out of the Warm reset, execution starts at the IMPLEMENTATION DEFINED reset vector address of the specified Execution state.

If EL2 can only use AArch64 state, this bit is RAO/WI.

The reset behavior of this field is:

- On a Cold reset, this field resets to '1'.

Accessing RMR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, RMR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1100	0b0000	0b010

```
if !(IsHighestEL(EL2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL1 && IsHighestEL(EL2) && EffectiveHCR_EL2_NVx() IN {'xx1'} then
    AArch64.SystemAccessTrap(EL2, 0x18);
elsif PSTATE.EL == EL2 && IsHighestEL(EL2) then
    X[t, 64] = RMR_EL2;
else
    UNDEFINED;
```

```
MSR RMR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1100	0b0000	0b010

```
if !(IsHighestEL(EL2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL1 && IsHighestEL(EL2) && EffectiveHCR_EL2_NVx() IN {'xx1'} then
    AArch64.SystemAccessTrap(EL2, 0x18);
elsif PSTATE.EL == EL2 && IsHighestEL(EL2) then
    RMR_EL2 = X[t, 64];
else
    UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.160 RMR_EL3, Reset Management Register (EL3)

The RMR_EL3 characteristics are:

Purpose

If EL3 is implemented and this register is implemented:

- A write to the register at EL3 can request a Warm reset.
- If EL3 can use all Execution states, this register specifies the Execution state that the PE boots into on a Warm reset.

Configuration

When EL3 is implemented:

- If EL3 can use all Execution states then this register must be implemented.
- If EL3 cannot use AArch32, then it is IMPLEMENTATION DEFINED whether the register is implemented.

Otherwise, direct accesses to RMR_EL3 are UNDEFINED.

When EL3 is implemented, AArch64 System register RMR_EL3 bits [31:0] are architecturally mapped to AArch32 System register RMR[31:0].

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to RMR_EL3 are UNDEFINED.

Attributes

RMR_EL3 is a 64-bit register.

Field descriptions

Bits [63:2]

Reserved, RES0.

RR, bit [1]

Reset Request. Setting this bit to 1 requests a Warm reset.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

AA64, bit [0]

When EL3 can use AArch32, determines which Execution state the PE boots into after a Warm reset:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

AA64	Meaning
0b0	AArch32.
0b1	AArch64.

On coming out of the Warm reset, execution starts at the IMPLEMENTATION DEFINED reset vector address of the specified Execution state.

If EL3 can only use AArch64 state, this bit is RAO/WI.

The reset behavior of this field is:

- On a Cold reset, this field resets to '1'.

Accessing RMR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, RMR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1100	0b0000	0b010

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL3 && IsHighestEL(EL3) then
    X[t, 64] = RMR_EL3;
else
    UNDEFINED;
```

```
MSR RMR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1100	0b0000	0b010

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL3 && IsHighestEL(EL3) then
    RMR_EL3 = X[t, 64];
else
    UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.161 RNDR, Random Number

The RNDR characteristics are:

Purpose

Random Number. Returns a 64-bit random number from an approved Random Bit Generator, where the Deterministic Random Bit Generator within the Random Bit Generator is reseeded from an approved entropy source at an IMPLEMENTATION DEFINED rate. See 'Properties of the generated random number'.

If the hardware returns a genuine random number, PSTATE.NZCV is set to `0b0000`.

If the instruction cannot return a genuine random number in a reasonable period of time, PSTATE.NZCV is set to `0b0100` and the data value returned is 0.

Configuration

This register is present only when (FEAT_RNG is implemented or FEAT_RNG_TRAP is implemented) and FEAT_AA64 is implemented. Otherwise, direct accesses to RNDR are UNDEFINED.

Attributes

RNDR is a 64-bit register.

Field descriptions

RNDR, bits [63:0]

Random Number. Returns a 64-bit Random Number from an approved Random Bit Generator, where the Deterministic Random Bit Generator within the Random Bit Generator is reseeded from an approved entropy source at an IMPLEMENTATION DEFINED rate. See 'Properties of the generated random number'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing RNDR

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, RNDR
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b0010	0b0100	0b000

```
if !((IsFeatureImplemented(FEAT_RNG) || IsFeatureImplemented(FEAT_RNG_TRAP)) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_RNG_TRAP) && SCR_EL3.TRNDR == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsFeatureImplemented(FEAT_RNG) then
        UNDEFINED;
    else
        X[t, 64] = RNDR;
elsif PSTATE.EL == EL1 then
    if IsFeatureImplemented(FEAT_RNG_TRAP) && SCR_EL3.TRNDR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsFeatureImplemented(FEAT_RNG) then
        UNDEFINED;
    else
        X[t, 64] = RNDR;
elsif PSTATE.EL == EL2 then
    if IsFeatureImplemented(FEAT_RNG_TRAP) && SCR_EL3.TRNDR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsFeatureImplemented(FEAT_RNG) then
        UNDEFINED;
    else
        X[t, 64] = RNDR;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RNG_TRAP) && SCR_EL3.TRNDR == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsFeatureImplemented(FEAT_RNG) then
        UNDEFINED;
    else
        X[t, 64] = RNDR;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.162 RNDRRS, Random Number Full Entropy

The RNDRRS characteristics are:

Purpose

Random Number with fresh full entropy. Returns a 64-bit random number from an approved Random Bit Generator, using either a Non-deterministic Random Bit Generator or one where the Deterministic Random Bit Generator is reseeded, where possible, from an approved entropy source before the return of the random number. See 'Properties of the generated random number'.

If the hardware returns a genuine random number, PSTATE.NZCV is set to `0b0000`.

If the instruction cannot return a genuine random number in a reasonable period of time, PSTATE.NZCV is set to `0b0100` and the data value returned is 0.

When FEAT_RNG_TRAP is implemented and SCR_EL3.TRNDR is 1, reads of this register are trapped to EL3.

Configuration

This register is present only when (FEAT_RNG is implemented or FEAT_RNG_TRAP is implemented) and FEAT_AA64 is implemented. Otherwise, direct accesses to RNDRRS are UNDEFINED.

Attributes

RNDRRS is a 64-bit register.

Field descriptions

RNDRRS, bits [63:0]

Random Number with fresh full entropy. Returns a 64-bit random number from an approved Random Bit Generator, using either a Non-deterministic Random Bit Generator or one where the Deterministic Random Bit Generator is reseeded, where possible, from an approved entropy source before the return of the random number. See 'Properties of the generated random number'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing RNDRRS

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, RNDRRS
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b0010	0b0100	0b001

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !((IsFeatureImplemented(FEAT_RNG) || IsFeatureImplemented(FEAT_RNG_TRAP)) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_RNG_TRAP) && SCR_EL3.TRNDR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsFeatureImplemented(FEAT_RNG) then
        UNDEFINED;
    else
        X[t, 64] = RNDRRS;
elsif PSTATE.EL == EL1 then
    if IsFeatureImplemented(FEAT_RNG_TRAP) && SCR_EL3.TRNDR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsFeatureImplemented(FEAT_RNG) then
        UNDEFINED;
    else
        X[t, 64] = RNDRRS;
elsif PSTATE.EL == EL2 then
    if IsFeatureImplemented(FEAT_RNG_TRAP) && SCR_EL3.TRNDR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsFeatureImplemented(FEAT_RNG) then
        UNDEFINED;
    else
        X[t, 64] = RNDRRS;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RNG_TRAP) && SCR_EL3.TRNDR == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsFeatureImplemented(FEAT_RNG) then
        UNDEFINED;
    else
        X[t, 64] = RNDRRS;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.163 RVBAR_EL1, Reset Vector Base Address Register (if EL2 and EL3 not implemented)

The RVBAR_EL1 characteristics are:

Purpose

If EL1 is the highest Exception level implemented, contains the IMPLEMENTATION DEFINED address that execution starts from after reset when executing in AArch64 state.

Configuration

This register is present only when the highest implemented Exception level is EL1 and FEAT_AA64 is implemented. Otherwise, direct accesses to RVBAR_EL1 are UNDEFINED.

Attributes

RVBAR_EL1 is a 64-bit register.

Field descriptions

ResetAddress, bits [63:0]

The IMPLEMENTATION DEFINED address that execution starts from after reset when executing in 64-bit state. Bits[1:0] of this register are 00, as this address must be aligned, and the address must be within the physical address size supported by the PE.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Accessing RVBAR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, RVBAR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b0000	0b001

```
if !(IsHighestEL(EL1) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL1 && IsHighestEL(EL1) then
    X[t, 64] = RVBAR_EL1;
else
    UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.164 RVBAR_EL2, Reset Vector Base Address Register (if EL3 not implemented)

The RVBAR_EL2 characteristics are:

Purpose

If EL2 is the highest Exception level implemented, contains the IMPLEMENTATION DEFINED address that execution starts from after reset when executing in AArch64 state.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when the highest implemented Exception level is EL2 and FEAT_AA64 is implemented. Otherwise, direct accesses to RVBAR_EL2 are UNDEFINED.

Attributes

RVBAR_EL2 is a 64-bit register.

Field descriptions

ResetAddress, bits [63:0]

The IMPLEMENTATION DEFINED address that execution starts from after reset when executing in 64-bit state. Bits[1:0] of this register are 00, as this address must be aligned, and the address must be within the physical address size supported by the PE.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Accessing RVBAR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, RVBAR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1100	0b0000	0b001

```
if !(IsHighestEL(EL2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL1 && IsHighestEL(EL2) && EffectiveHCR_EL2_NVx() IN {'xx1'} then
    AArch64.SystemAccessTrap(EL2, 0x18);
elsif PSTATE.EL == EL2 && IsHighestEL(EL2) then
    X[t, 64] = RVBAR_EL2;
else
    UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.165 RVBAR_EL3, Reset Vector Base Address Register (if EL3 implemented)

The RVBAR_EL3 characteristics are:

Purpose

If EL3 is the highest Exception level implemented, contains the IMPLEMENTATION DEFINED address that execution starts from after reset when executing in AArch64 state.

Configuration

Only implemented if the highest Exception level implemented is EL3.

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to RVBAR_EL3 are UNDEFINED.

Attributes

RVBAR_EL3 is a 64-bit register.

Field descriptions

ResetAddress, bits [63:0]

The IMPLEMENTATION DEFINED address that execution starts from after reset when executing in 64-bit state. Bits[1:0] of this register are 00, as this address must be aligned, and the address must be within the physical address size supported by the PE.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Accessing RVBAR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, RVBAR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1100	0b0000	0b001

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL3 && IsHighestEL(EL3) then
    X[t, 64] = RVBAR_EL3;
else
    UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.166 S2PIR_EL2, Stage 2 Permission Indirection Register (EL2)

The S2PIR_EL2 characteristics are:

Purpose

Stage 2 Permission Indirection Register for EL1&0 translation regime.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_S2PIE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to S2PIR_EL2 are UNDEFINED.

Attributes

S2PIR_EL2 is a 64-bit register.

Field descriptions

Perm<m> , bits [4m+3:4m], for m = 15 to 0

Represents stage 2 Base Permissions.

Perm<m>	Meaning
0b0000	No Access.
0b0001	Reserved - treated as No Access.
0b0010	MRO.
0b0011	MRO-TL1.
0b0100	WO.
0b0101	Reserved - treated as No Access.
0b0110	MRO-TL0.
0b0111	MRO-TL01.
0b1000	RO.
0b1001	RO+uX.
0b1010	RO+pX.
0b1011	RO+puX.
0b1100	RW.
0b1101	RW+uX.
0b1110	RW+pX.
0b1111	RW+puX.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Perm<m>	Meaning

This field is permitted to be cached in a TLB.

When stage 2 Indirect Permission mechanism is disabled, the contents of this register are ignored.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing S2PIR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, S2PIR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0010	0b101

```
if !(IsFeatureImplemented(FEAT_S2PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x2B0];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = S2PIR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = S2PIR_EL2;
```

```
MSR S2PIR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0010	0b101

```
if !(IsFeatureImplemented(FEAT_S2PIE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x2B0] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        S2PIR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    S2PIR_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.167 S2POR_EL1, Stage 2 Permission Overlay Register (EL1)

The S2POR_EL1 characteristics are:

Purpose

Stage 2 Permission Overlay Register for EL1&0 translation regime.

Configuration

This register is present only when FEAT_S2POE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to S2POR_EL1 are UNDEFINED.

Attributes

S2POR_EL1 is a 64-bit register.

Field descriptions

Perm<m> , bits [4m+3:4m], for m = 15 to 0

Configures stage 2 Overlay Permissions.

Perm<m>	Meaning
0b0000	No Access.
0b0001	Reserved - treated as No Access.
0b0010	MRO.
0b0011	MRO-TL1.
0b0100	WO.
0b0101	Reserved - treated as No Access.
0b0110	MRO-TL0.
0b0111	MRO-TL01.
0b1000	RO.
0b1001	RO+uX.
0b1010	RO+pX.
0b1011	RO+puX.
0b1100	RW.
0b1101	RW+uX.
0b1110	RW+pX.
0b1111	RW+puX.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

This field is not permitted to be cached in a TLB.

When stage 2 Permission Overlay mechanism is disabled, this register is ignored.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing S2POR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, S2POR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b101

```
if !(IsFeatureImplemented(FEAT_S2POE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.nS2POR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x2B8];
    else
        X[t, 64] = S2POR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = S2POR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = S2POR_EL1;
```

```
MSR S2POR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0010	0b101

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_S2POE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.nS2POR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x2B8] = X[t, 64];
    else
        S2POR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.PIEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.PIEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        S2POR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    S2POR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.168 S3_<op1>_<Cn>_<Cm>_<op2>, IMPLEMENTATION DEFINED Registers

The S3_<op1><*Cn*><Cm>_<op2> characteristics are:

Purpose

This area of the instruction set space is reserved for IMPLEMENTATION DEFINED registers.

Configuration

When FEAT_SYSREG128 is implemented, each register in this space is a 128-bit register that can also be accessed as a 64-bit value. If it is accessed as a 64-bit register, accesses read and write bits [63:0] and do not modify bits [127:64].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to S3_<op1>_<Cn>_<Cm>_<op2> are UNDEFINED.

Attributes

S3_<op1><*Cn*><Cm>_<op2> is a:

- 128-bit register when FEAT_SYSREG128 is implemented.
- 64-bit register otherwise.

Field descriptions

When FEAT_SYSREG128 is implemented:

127 … 96	95 … 64	63 … 32	31 … 0
IMPLEMENTATION DEFINED	IMPLEMENTATION DEFINED	IMPLEMENTATION DEFINED	IMPLEMENTATION DEFINED

IMPLEMENTATION DEFINED, bits [127:0]

IMPLEMENTATION DEFINED.

Otherwise:

63 … 32	31 … 0
IMPLEMENTATION DEFINED	IMPLEMENTATION DEFINED

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing S3_<op1>_<Cn>_<Cm>_<op2>

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, S3_<op1>_C<Cn>_C<Cm>_<op2>
```

op0	op1	CRn	CRm	op2
0b11	op1[2:0]	0b1x11	Cm[3:0]	op2[2:0]

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.TIDCP == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.TIDCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.ImpDefSysRegRead(op0, op1, CRn, CRm, op2, t);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TIDCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.ImpDefSysRegRead(op0, op1, CRn, CRm, op2, t);
elsif PSTATE.EL == EL2 then
    AArch64.ImpDefSysRegRead(op0, op1, CRn, CRm, op2, t);
elsif PSTATE.EL == EL3 then
    AArch64.ImpDefSysRegRead(op0, op1, CRn, CRm, op2, t);
```

```
MSR S3_<op1>_C<Cn>_C<Cm>_<op2>, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	op1[2:0]	0b1x11	Cm[3:0]	op2[2:0]

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && SCTLR_EL1.TIDCP == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.TIDCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.ImpDefSysRegWrite(op0, op1, CRn, CRm, op2, t);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TIDCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.ImpDefSysRegWrite(op0, op1, CRn, CRm, op2, t);
elsif PSTATE.EL == EL2 then
    AArch64.ImpDefSysRegWrite(op0, op1, CRn, CRm, op2, t);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL3 then
    AArch64.ImpDefSysRegWrite(op0, op1, CRn, CRm, op2, t);
```

When FEAT_SYSREG128 is implemented MRRS <Xt>, <Xt+1>, S3_<op1>_C<Cn>_C<Cm>_<op2>

op0	op1	CRn	CRm	op2
0b11	op1[2:0]	0b1x11	Cm[3:0]	op2[2:0]

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnIDCP128 == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL0) && SCTLR_EL1.TIDCP == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x14);
        else
            AArch64.SystemAccessTrap(EL1, 0x14);
    elsif ELIsInHost(EL0) && SCTLR_EL2.TIDCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif !ELIsInHost(EL0) && (!IsSCTLR2EL1Enabled() || SCTLR2_EL1.EnIDCP128 == '0') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x14);
        else
            AArch64.SystemAccessTrap(EL1, 0x14);
    elsif ELIsInHost(EL0) && (!IsSCTLR2EL2Enabled() || SCTLR2_EL2.EnIDCP128 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && !ELIsInHost(EL0) && (!IsHCRXEL2Enabled() || HCRX_EL2.EnIDCP128 == '0')
     ↪then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.EnIDCP128 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        AArch64.ImpDefSysRegRead128(op0, op1, CRn, CRm, op2, t, t2);
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnIDCP128 == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TIDCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.EnIDCP128 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.EnIDCP128 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        AArch64.ImpDefSysRegRead128(op0, op1, CRn, CRm, op2, t, t2);
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnIDCP128 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.EnIDCP128 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        AArch64.ImpDefSysRegRead128(op0, op1, CRn, CRm, op2, t, t2);
elsif PSTATE.EL == EL3 then
    AArch64.ImpDefSysRegRead128(op0, op1, CRn, CRm, op2, t, t2);
```

When FEAT_SYSREG128 is implemented MSRR S3_<op1>_C<Cn>_C<Cm>_<op2>, <Xt>, <Xt+1>

op0	op1	CRn	CRm	op2
0b11	op1[2:0]	0b1x11	Cm[3:0]	op2[2:0]

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnIDCP128 == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL0) && SCTLR_EL1.TIDCP == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x14);
        else
            AArch64.SystemAccessTrap(EL1, 0x14);
    elsif ELIsInHost(EL0) && SCTLR_EL2.TIDCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif !ELIsInHost(EL0) && (!IsSCTLR2EL1Enabled() || SCTLR2_EL1.EnIDCP128 == '0') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x14);
        else
            AArch64.SystemAccessTrap(EL1, 0x14);
    elsif ELIsInHost(EL0) && (!IsSCTLR2EL2Enabled() || SCTLR2_EL2.EnIDCP128 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && !ELIsInHost(EL0) && (!IsHCRXEL2Enabled() || HCRX_EL2.EnIDCP128 == '0')
     ↪then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.EnIDCP128 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        AArch64.ImpDefSysRegWrite128(op0, op1, CRn, CRm, op2, t, t2);
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnIDCP128 == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TIDCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.EnIDCP128 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.EnIDCP128 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        AArch64.ImpDefSysRegWrite128(op0, op1, CRn, CRm, op2, t, t2);
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnIDCP128 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.EnIDCP128 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    else
        AArch64.ImpDefSysRegWrite128(op0, op1, CRn, CRm, op2, t, t2);
elsif PSTATE.EL == EL3 then
    AArch64.ImpDefSysRegWrite128(op0, op1, CRn, CRm, op2, t, t2);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.169 SCR_EL3, Secure Configuration Register

The SCR_EL3 characteristics are:

Purpose

Defines the configuration of the current Security state. It specifies:

- The Security state of EL0, EL1, and EL2. The Security state is Secure, Non-secure, or Realm.
- The Execution state at lower Exception levels.
- Whether IRQ, FIQ, SError exceptions, and External abort exceptions are taken to EL3.
- Whether various operations are trapped to EL3.

Configuration

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SCR_EL3 are UNDEFINED.

Attributes

SCR_EL3 is a 64-bit register.

Field descriptions

Bit [63]

Reserved, RES0.

NSE, bit [62]

When FEAT_RME is implemented:

This field, evaluated with SCR_EL3.NS, selects the Security state of EL2 and lower Exception levels.

For a description of the values derived by evaluating NS and NSE together, see SCR_EL3.NS.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

The Effective value of this bit is `0b0`.

Access to this field is RES0.

HACDBSEn, bit [61]

When FEAT_HACDBS is implemented:

Enables access to the HACDBSBR_EL2 and HACDBSCONS_EL2 registers at EL2.

HACDBSEn	Meaning
`0b0`	EL2 accesses to the specified registers are trapped to EL3.
`0b1`	This control does not cause any instructions to be trapped.

Traps are reported using EC syndrome value `0x18`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HDBSSEn, bit [60]

When FEAT_HDBSS is implemented:

Enables access to HDBSSBR_EL2 and HDBSSPROD_EL2 registers at EL2.

HDBSSEn	Meaning
`0b0`	EL2 accesses to the specified registers are trapped to EL3.
`0b1`	This control does not cause any instructions to be trapped.

Traps are reported using EC syndrome value `0x18`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FGTEn2, bit [59]

When FEAT_FGT2 is implemented:

Fine-Grained Traps Enable 2.

When EL2 is implemented, enables the traps to EL2 controlled by HDFGRTR2_EL2, HDFGWTR2_EL2, HFGITR2_EL2, HFGRTR2_EL2, and HFGWTR2_EL2, and controls access to those registers.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FGTEn2	Meaning
0b0	EL2 accesses to the specified registers are trapped to EL3. The values in these registers are treated as 0.
0b1	EL2 accesses to the specified registers are not trapped to EL3 by this mechanism.

Traps caused by accesses to the fine-grained trap registers are reported using EC syndrome value 0x18 and its associated ISS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnDSE, bit [58]

When FEAT_E3DSE is implemented:

Enable for delegated SError exceptions pended by SCR_EL3.DSE.

EnDSE	Meaning
0b0	Delegated SError exceptions pended by SCR_EL3.DSE are disabled.
0b1	Delegated SError exceptions pended by SCR_EL3.DSE are enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DSE, bit [57]

When FEAT_E3DSE is implemented:

Delegated SError exception for EL2, EL1, and EL0.

DSE	Meaning
0b0	This mechanism is not making a delegated SError exception pending.
0b1	A delegated SError exception for EL2, EL1, and EL0 is pending because of this mechanism.

When EL2 is implemented and enabled in the current Security state, delegated SError exceptions pended by this field are affected by HCR_EL2.AMO and HCRX_EL2.TMEA.

Virtual SError exceptions pended by HCR_EL2.VSE have priority over delegated SError exceptions pended by this field.

This field is ignored by the PE and treated as zero when SCR_EL3.EnDSE == 0

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [56]

Reserved, RES0.

EnIDCP128, bit [55]

When FEAT_SYSREG128 is implemented:

Enables access to IMPLEMENTATION DEFINED 128-bit System registers.

EnIDCP128	Meaning
`0b0`	Accesses at EL2, EL1, EL0 to IMPLEMENTATION DEFINED 128-bit System registers are trapped to EL3 using EC syndrome value `0x14`, unless the access generates a higher priority exception. Disables the functionality of the 128-bit IMPLEMENTATION DEFINED System registers that are accessible at EL3.
`0b1`	No accesses are trapped by this control.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SRMASKEn, bit [54]

When FEAT_SRMASK is implemented:

Enables access to the following MASK registers:

- CPACRMASK_EL1, and CPACRMASK_EL12.
- SCTLRMASK_EL1, and SCTLRMASK_EL12.
- SCTLR2MASK_EL1, and SCTLR2MASK_EL12.
- TCRMASK_EL1, and TCRMASK_EL12.
- TCR2MASK_EL1, and TCR2MASK_EL12.
- ACTLRMASK_EL1 and ACTLRMASK_EL12, if they are implemented.
- CPTRMASK_EL2.
- SCTLRMASK_EL2.
- SCTLR2MASK_EL2.
- TCRMASK_EL2.
- TCR2MASK_EL2.
- ACTLRMASK_EL2, if it is implemented.

SRMASKEn	Meaning
`0b0`	EL2 accesses to the specified registers are trapped to EL3. EL1 accesses to the specified EL1 registers are trapped to EL3. The values in the registers are treated as 0.
`0b1`	No accesses are trapped by this control.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SRMASKEn	Meaning

Traps are reported using EC syndrome value `0x18`.

Traps generated by this control have a lower priority than traps generated by the HCRX_EL2.SRMASKEn, HFGRTR2_EL2 and HFGWTR2_EL2 controls.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PFAREn, bit [53]

When FEAT_PFAR is implemented:

Enable access to Physical Fault Address Registers. When disabled, accesses to Physical Fault Address Registers generate a trap to EL3.

PFAREn	Meaning
`0b0`	Accesses of the specified Physical Fault Address Registers at EL2 and EL1 are trapped to EL3, unless the instruction generates a higher priority exception.
`0b1`	This control does not cause any instructions to be trapped.

In AArch64 state, the instructions affected by this control are: `MRS` and `MSR` accesses to PFAR_EL1, PFAR_EL2, and PFAR_EL12.

Unless the instruction generates a higher priority exception, trapped instructions generate an exception to EL3.

Trapped instructions are reported using EC syndrome value `0x18`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TWERR, bit [52]

When FEAT_RASv2 is implemented:

Trap writes of Error Record registers. Enables a trap to EL3 on writes of Error Record registers.

TWERR	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	Writes of the specified Error Record registers at EL2 and EL1 are trapped to EL3, unless the instruction generates a higher priority exception.

In AArch64 state, the instructions affected by this control are: `MSR` accesses to ERRSELR_EL1, ERXADDR_EL1, ERXCTLR_EL1, ERXMISC0_EL1, ERXMISC1_EL1, ERXMISC2_EL1, ERXMISC3_EL1, and ERXSTATUS_EL1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

In AArch32 state, the instructions affected by this control are: MCR accesses to ERRSELR, ERXADDR, ERXADDR2, ERXCTLR, ERXCTLR2, ERXMISC0, ERXMISC1, ERXMISC2, ERXMISC3, ERXMISC4, ERXMISC5, ERXMISC6, ERXMISC7, and ERXSTATUS.

Unless the instruction generates a higher priority exception, trapped instructions generate an exception to EL3.

Trapped AArch64 instructions are reported using EC syndrome value `0x18`.

Trapped AArch32 instructions are reported using EC syndrome value `0x03`.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TMEA, bit [51]

When FEAT_DoubleFault2 is implemented:

Trap Masked External Aborts. Controls whether a masked error exception at a lower Exception level is taken to EL3.

TMEA	Meaning
`0b0`	Synchronous External abort exceptions and SError exceptions at EL2, EL1, and EL0 are unaffected by this mechanism. That is, these exceptions are not taken to EL3 unless routed to EL3 by another control.
`0b1`	When executing at Exception levels below EL3, all of the following apply: • When PSTATE.A is 1, synchronous External abort exceptions are taken to EL3, unless they are taken from EL1 or EL0 and routed to EL2 by another control. • Masked physical SError exceptions are taken to EL3, unless they are taken from EL1 or EL0 and routed to EL2 by another control.

This field has no effect on the routing of virtual or delegated SError exceptions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnFPM, bit [50]

When FEAT_FPMR is implemented:

Enables direct and indirect accesses to FPMR from EL2, EL1, and EL0.

When accesses to FPMR are disabled by this control:

- Direct accesses to FPMR from EL2, EL1, and EL0 are trapped to EL3 and reported with EC syndrome value `0x18`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- Execution of FP8 data-processing instructions that indirectly access FPMR is UNDEFINED at EL2, EL1 and EL0.

EnFPM	Meaning
0b0	Direct and indirect accesses to FPMR are disabled at EL2, EL1 and EL0.
0b1	This control does not cause any instructions to be trapped.

Traps are not taken if there is a higher priority exception generated by the access.

If EL3 is not implemented, the Effective value of this field is 0b1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MECEn, bit [49]

When FEAT_MEC is implemented:

Enables access to the following EL2 MECID registers, from EL2:

- MECID_P0_EL2.
- MECID_A0_EL2
- MECID_P1_EL2
- MECID_A1_EL2
- VMECID_P_EL2
- VMECID_A_EL2

Accesses to these registers are trapped and reported using EC syndrome value 0x18.

MECEn	Meaning
0b0	EL2 accesses to any of the specified registers are trapped to EL3. The values of the specified registers are treated as 0 for all purposes other than direct reads or writes to the register from EL3.
0b1	This control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

GPF, bit [48]

When FEAT_RME is implemented:

Controls the reporting of Granule protection faults at EL0, EL1 and EL2.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

GPF	Meaning
0b0	This control does not cause exceptions to be routed from EL0, EL1 or EL2 to EL3.
0b1	GPFs at EL0, EL1 and EL2 are routed to EL3 and reported as Granule Protection Check exceptions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

D128En, bit [47]

When FEAT_D128 is implemented:

128-bit System Register trap control. Enables access to 128-bit System Registers via MRRS, MSRR instructions.

MRRS and MSRR accesses from EL1 and EL2 using AArch64 to the following registers are trapped and reported using EC syndrome value 0x14:

- TTBR0_EL1.
- TTBR1_EL1.
- RCWMASK_EL1, RCWSMASK_EL1.
- PAR_EL1.

MRRS and MSRR accesses from EL2 using AArch64 to the following registers are trapped and reported using EC syndrome value 0x14:

- TTBR1_EL2 and accesses using the register name TTBR1_EL12.
- TTBR0_EL2 and accesses using the register name TTBR0_EL12.
- VTTBR_EL2.

D128En	Meaning
0b0	EL1 and EL2 accesses to the specified registers are disabled, and trapped to EL3.
0b1	This control does not cause any instructions to be trapped.

Traps are not taken if there is a higher priority exception generated by the access.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AIEn, bit [46]

When FEAT_AIE is implemented:

MAIR2_ELx, AMAIR2_ELx Register access trap control.

Accesses from EL1 and EL2 using AArch64 to the following registers are trapped and reported using EC syndrome value 0x18:

- AMAIR2_EL1.
- MAIR2_EL1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accesses from EL2 using AArch64 to the following registers are trapped and reported using EC syndrome value `0x18`:

- AMAIR2_EL2 and accesses using the register name AMAIR2_EL12.
- MAIR2_EL2 and accesses using the register name MAIR2_EL12.

AIEn	Meaning
`0b0`	EL1 and EL2 accesses to the specified registers are disabled, and trapped to EL3. The values in these registers are treated as 0.
`0b1`	This control does not cause any instructions to be trapped.

Traps are not taken if there is a higher priority exception generated by the access.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PIEn, bit [45]

When FEAT_S1PIE is implemented, or FEAT_S2PIE is implemented, or FEAT_S1POE is implemented, or FEAT_S2POE is implemented:

Permission Indirection, Overlay Register access trap control. Enables access to Permission Indirection and Overlay registers.

Accesses from EL0, EL1 and EL2 using AArch64 to the following registers are trapped and reported using EC syndrome value `0x18`:

- POR_EL0.

Accesses from EL1 and EL2 using AArch64 to the following registers are trapped and reported using EC syndrome value `0x18`:

- PIRE0_EL1.
- PIR_EL1.
- POR_EL1.
- S2POR_EL1.

Accesses from EL2 using AArch64 to the following registers are trapped and reported using EC syndrome value `0x18`:

- PIRE0_EL2 and accesses using the register name PIRE0_EL12.
- PIR_EL2 and accesses using the register name PIR_EL12.
- POR_EL2 and accesses using the register name POR_EL12.
- S2PIR_EL2.

PIEn	Meaning
`0b0`	EL0, EL1 and EL2 accesses to the specified registers are disabled, and trapped to EL3.
`0b1`	This control does not cause any instructions to be trapped.

If this field is 0, it is IMPLEMENTATION SPECIFIC whether the values of the named registers are treated as zero.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Traps are not taken if there is a higher priority exception generated by the access.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SCTLR2En, bit [44]

When FEAT_SCTLR2 is implemented:

SCTLR2_ELx register trap control. Enables access to SCTLR2_EL1 and SCTLR2_EL2 registers.

SCTLR2En	Meaning
0b0	EL1 and EL2 accesses to SCTLR2_EL1 and SCTLR2_EL2 registers are disabled, and trapped to EL3. The values in these registers are treated as 0.
0b1	This control does not cause any instructions to be trapped.

Traps are reported using EC syndrome value 0x18.

Traps are not taken if there is a higher priority exception generated by the access.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCR2En, bit [43]

When FEAT_TCR2 is implemented:

TCR2_ELx register trap control. Enables access to TCR2_EL1 and TCR2_EL2 registers.

TCR2En	Meaning
0b0	EL1 and EL2 accesses to TCR2_EL1 and TCR2_EL2 registers are disabled, and trapped to EL3. The values in these registers are treated as 0.
0b1	This control does not cause any instructions to be trapped.

Traps are reported using EC syndrome value 0x18.

Traps are not taken if there is a higher priority exception generated by the access.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RCWMASKEn, bit [42]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_THE is implemented:

RCW and RCWS Mask register trap control. Enables access to RCWMASK_EL1, RCWSMASK_EL1.

RCWMASKEn	Meaning
0b0	EL1 and EL2 accesses to RCWMASK_EL1 and RCWSMASK_EL1 registers are disabled, and trapped to EL3.
0b1	This control does not cause any instructions to be trapped.

Traps for MRS, MSR access are reported using EC syndrome value 0x18.

Traps for MRRS, MSRR acceess are reported using EC syndrome value 0x14.

Traps are not taken if there is a higher priority exception generated by the access.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnTP2, bit [41]

When FEAT_SME is implemented:

Traps instructions executed at EL2, EL1, and EL0 that access TPIDR2_EL0 to EL3. The exception is reported using EC syndrome value 0x18.

EnTP2	Meaning
0b0	This control causes execution of these instructions at EL2, EL1, and EL0 to be trapped.
0b1	This control does not cause execution of any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRNDR, bit [40]

When FEAT_RNG_TRAP is implemented:

Controls trapping of reads of RNDR and RNDRRS. The exception is reported using EC syndrome value 0x18.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TRNDR	Meaning
0b0	This control does not cause RNDR and RNDRRS to be trapped. When FEAT_RNG is implemented: • ID_AA64ISAR0_EL1.RNDR returns the value 0b0001. When FEAT_RNG is not implemented: • ID_AA64ISAR0_EL1.RNDR returns the value 0b0000. • MRS reads of RNDR and RNDRRS are UNDEFINED.
0b1	ID_AA64ISAR0_EL1.RNDR returns the value 0b0001. Any attempt to read RNDR or RNDRRS is trapped to EL3.

When FEAT_RNG is not implemented, Arm recommends that SCR_EL3.TRNDR is initialized before entering Exception levels below EL3 and not subsequently changed.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

GCSEn, bit [39]

When FEAT_GCS is implemented:

Guarded Control Stack enable. Controls access to the Guarded Control Stack registers from EL2, EL1, and EL0, and controls whether the Guarded Control Stack is enabled.

The Guarded Control Stack registers trapped by this mechanism are:

- GCSCRE0_EL1.
- GCSCR_EL1.
- GCSCR_EL2.
- GCSCR_EL12.
- GCSPR_EL0.
- GCSPR_EL1.
- GCSPR_EL2.
- GCSPR_EL12.

GCSEn	Meaning
0b0	Trap read and write accesses to all Guarded Control Stack registers to EL3. All Guarded Control Stack behavior is disabled at EL2, EL1, and EL0.
0b1	This control does not cause any instructions to be trapped, and does not disable Guarded Control Stack behavior at EL2, EL1, or EL0.

Traps are reported using EC syndrome value 0x18.

Traps are not taken if there is a higher priority exception generated by the access.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

HXEn, bit [38]

When FEAT_HCX is implemented:

Enables access to the HCRX_EL2 register at EL2 from EL3.

HXEn	Meaning
0b0	Accesses at EL2 to HCRX_EL2 are trapped to EL3.
0b1	This control does not cause any instructions to be trapped.

When EL3 is not implemented, the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ADEn, bit [37]

When FEAT_LS64_ACCDATA is implemented:

Enables access to the ACCDATA_EL1 register at EL1 and EL2.

ADEn	Meaning
0b0	Accesses to ACCDATA_EL1 at EL1 and EL2 are trapped to EL3, unless the accesses are trapped to EL2 by the EL2 fine-grained trap.
0b1	This control does not cause accesses to ACCDATA_EL1 to be trapped.

If the HFGWTR_EL2.nACCDATA_EL1 or HFGRTR_EL2.nACCDATA_EL1 traps are enabled, they take priority over this trap.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnAS0, bit [36]

When FEAT_LS64_ACCDATA is implemented:

Traps execution of an ST64BV0 instruction at EL0, EL1, or EL2 to EL3.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EnAS0	Meaning
0b0	EL0 execution of an ST64BV0 instruction is trapped to EL3, unless it is trapped to EL1 by SCTLR_EL1.EnAS0, or to EL2 by either HCRX_EL2.EnAS0 or SCTLR_EL2.EnAS0. EL1 execution of an ST64BV0 instruction is trapped to EL3, unless it is trapped to EL2 by HCRX_EL2.EnAS0. EL2 execution of an ST64BV0 instruction is trapped to EL3.
0b1	This control does not cause any instructions to be trapped.

A trap of an ST64BV0 instruction is reported using EC syndrome value 0x0A, with an ISS code of 0x0000001.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AMVOFFEN, bit [35]

When FEAT_AMUv1p1 is implemented:

Activity Monitors Virtual Offsets Enable.

AMVOFFEN	Meaning
0b0	Accesses to AMEVCNTVOFF0<n>_EL2 and AMEVCNTVOFF1<n>_EL2 at EL2 are trapped to EL3. Indirect reads of the virtual offset registers are zero.
0b1	Accesses to AMEVCNTVOFF0<n>_EL2 and AMEVCNTVOFF1<n>_EL2 are not affected by this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [34]

Reserved, RES0.

TWEDEL, bits [33:30]

When FEAT_TWED is implemented:

TWE Delay. A 4-bit unsigned number that, when SCR_EL3.TWEDEn is 1, encodes the minimum delay in taking a trap of WFE* caused by SCR_EL3.TWE as $2^{(\overline{\text{TWEDEL}} + 8)}$ cycles.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TWEDEn, bit [29]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_TWED is implemented:

TWE Delay Enable. Enables a configurable delayed trap of the WFE* instruction caused by SCR_EL3.TWE.

Traps are reported using EC syndrome value `0x01`.

TWEDEn	Meaning
`0b0`	The delay for taking the trap is IMPLEMENTATION DEFINED.
`0b1`	The delay for taking the trap is at least the number of cycles defined in SCR_EL3.TWEDEL.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ECVEn, bit [28]

When FEAT_ECV_POFF is implemented:

ECV Enable. Enables access to the CNTPOFF_EL2 register.

ECVEn	Meaning
`0b0`	EL2 accesses to CNTPOFF_EL2 are trapped to EL3, and the value of CNTPOFF_EL2 is treated as 0 for all purposes other than direct reads or writes to the register from EL3.
`0b1`	EL2 accesses to CNTPOFF_EL2 are not trapped to EL3 by this mechanism.

When FEAT_ECV_POFF is not implemented, the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FGTEn, bit [27]

When FEAT_FGT is implemented:

Fine-Grained Traps Enable. When EL2 is implemented, enables the traps to EL2 controlled by HAFGRTR_EL2, HDFGRTR_EL2, HDFGWTR_EL2, HFGRTR_EL2, HFGITR_EL2, and HFGWTR_EL2, and controls access to those registers.

Note

If EL2 is not implemented but EL3 is implemented, FEAT_FGT implements the MDCR_EL3.TDCC traps.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FGTEn	Meaning
0b0	EL2 accesses to HAFGRTR_EL2, HDFGRTR_EL2, HDFGWTR_EL2, HFGRTR_EL2, HFGITR_EL2 and HFGWTR_EL2 registers are trapped to EL3, and the traps to EL2 controlled by those registers are disabled.
0b1	EL2 accesses to HAFGRTR_EL2, HDFGRTR_EL2, HDFGWTR_EL2, HFGRTR_EL2, HFGITR_EL2 and HFGWTR_EL2 registers are not trapped to EL3 by this mechanism.

Traps caused by accesses to the fine-grained trap registers are reported using EC syndrome value 0x18 and its associated ISS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ATA, bit [26]

When FEAT_MTE2 is implemented:

Memory tagging enable override:

- Overrides enabling of Memory tagging at EL2, EL1 and EL0.
- If not trapped to EL2, accesses to the following registers at EL1 and EL2 are trapped to EL3 and reported using EC syndrome value 0x18:
  - GCR_EL1.
  - RGSR_EL1.
  - TFSRE0_EL1.
  - TFSR_EL1.
  - TFSR_EL2.
  - Accesses with the register name TFSR_EL12 that are not UNDEFINED.

ATA	Meaning
0b0	Disables the use of Memory tagging at EL2, EL1 and EL0. The specified registers are trapped to EL3 unless trapped to a lower Exception level.
0b1	This field does not disable the use of Memory tagging at EL2, EL1 and EL0. The field does not trap the specified registers to EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnSCXT, bit [25]

When FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented:

Enables access to the SCXTNUM_EL2, SCXTNUM_EL1, and SCXTNUM_EL0 registers.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EnSCXT	Meaning
0b0	Accesses at EL0, EL1 and EL2 to SCXTNUM_EL0, SCXTNUM_EL1, or SCXTNUM_EL2 registers are trapped to EL3 if they are not trapped by a higher priority exception, and the values of these registers are treated as 0.
0b1	This control does not cause any accesses to be trapped, or register values to be treated as 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [24]

Reserved, RES0.

TID5, bit [23]

When FEAT_IDTE3 is implemented and FEAT_MTE2 is implemented:

Trap ID group 5. EL2 and EL1 reads of the group 5 ID register GMID_EL1 are trapped to EL3, reported using EC syndrome value 0x18, unless the instruction generates a higher priority exception.

TID5	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	The specified EL2 accesses to ID group 5 registers are trapped to EL3. When EL2 is not enabled in the current Security state, or when HCR_EL2.TID5 is 0, EL1 read accesses to the specified registers are trapped to EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TID3, bit [22]

When FEAT_IDTE3 is implemented:

Trap ID group 3. EL2 and EL1 reads of the following group 3 registers are trapped to EL3, reported using EC syndrome value 0x18, unless the instruction generates a higher priority exception:

- ID_PFR0_EL1, ID_PFR1_EL1, ID_DFR0_EL1, ID_AFR0_EL1, ID_MMFR0_EL1, ID_MMFR1_EL1, ID_MMFR2_EL1, ID_MMFR3_EL1, ID_ISAR0_EL1, ID_ISAR1_EL1, ID_ISAR2_EL1, ID_ISAR3_EL1, ID_ISAR4_EL1, ID_ISAR5_EL1, MVFR0_EL1, MVFR1_EL1, and MVFR2_EL1.
- ID_AA64PFR0_EL1, ID_AA64PFR1_EL1, ID_AA64DFR0_EL1, ID_AA64DFR1_EL1, ID_AA64ISAR0_EL1, ID_AA64ISAR1_EL1, ID_AA64MMFR0_EL1, ID_AA64MMFR1_EL1, ID_AA64AFR0_EL1, and ID_AA64AFR1_EL1.
- ID_PFR2_EL1, and ID_MMFR4_EL1.
- ID_MMFR5_EL1.
- ID_AA64MMFR3_EL1.
- ID_AA64MMFR4_EL1.
- ID_AA64PFR2_EL1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- ID_AA64MMFR2_EL1 and ID_ISAR6_EL1.
- ID_DFR1_EL1.
- ID_AA64DFR2_EL1.
- ID_AA64ZFR0_EL1.
- ID_AA64SMFR0_EL1.
- ID_AA64FPFR0_EL1.
- ID_AA64ISAR2_EL1.
- ID_AA64ISAR3_EL1.
- This field traps all MRS accesses to registers in the following range that are not already mentioned in this field description: op0 == 3, op1 == 0, CRn == 0, CRm == {2-7}, op2 == {0-7}.

TID3	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	The specified EL2 read accesses to ID group 3 registers are trapped to EL3. When EL2 is not enabled in the current Security state, or when HCR_EL2.TID3 is 0, EL1 read accesses to the specified registers are trapped to EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FIEN, bit [21]

When FEAT_RASv1p1 is implemented:

Fault Injection enable. Trap accesses to the registers ERXPFGCDN_EL1, ERXPFGCTL_EL1, and ERXPFGF_EL1 from EL1 and EL2 to EL3, reported using EC syndrome value `0x18`.

FIEN	Meaning
`0b0`	Accesses to the specified registers from EL1 and EL2 generate a Trap exception to EL3.
`0b1`	This control does not cause any instructions to be trapped.

If EL3 is not implemented, the Effective value of SCR_EL3.FIEN is `0b1`.

If ERRIDR_EL1.NUM is zero, meaning no error records are implemented, or no error record accessible using System registers is owned by a node that implements the RAS Common Fault Injection Model Extension, then this bit might be RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NMEA, bit [20]

When FEAT_DoubleFault is implemented:

Non-maskable External Aborts. Controls whether PSTATE.A masks SError exceptions at EL3.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

NMEA	Meaning
0b0	SError exceptions are not taken at EL3 if PSTATE.A == 1.
0b1	SError exceptions are taken at EL3 regardless of the value of PSTATE.A.

This field is ignored by the PE and treated as zero when all of the following are true:

- FEAT_DoubleFault2 is not implemented.
- SCR_EL3.EA is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

EASE, bit [19]

When FEAT_DoubleFault is implemented:

External aborts to SError exception vector.

EASE	Meaning
0b0	Synchronous External abort exceptions taken to EL3 are taken to the appropriate synchronous exception vector offset from VBAR_EL3.
0b1	Synchronous External abort exceptions taken to EL3 are taken to the appropriate SError exception vector offset from VBAR_EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

EEL2, bit [18]

When FEAT_SEL2 is implemented:

Secure EL2 Enable.

EEL2	Meaning
0b0	All behaviors associated with Secure EL2 are disabled. All registers, including timer registers, defined by FEAT_SEL2 are UNDEFINED, and those timers are disabled.
0b1	All behaviors associated with Secure EL2 are enabled.

When the value of this bit is 1, then:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- When SCR_EL3.NS == 0, the SCR_EL3.RW bit is treated as 1 for all purposes other than reading or writing the register.
- If Secure EL1 is using AArch32, then any of the following operations, executed in Secure EL1, is trapped to Secure EL2, using EC syndrome value `0x03` :
  - A read or write of the SCR.
  - A read or write of the NSACR.
  - A read or write of the MVBAR.
  - A read or write of the SDCR.
  - Execution of an ATS12NSO** instruction.
- If Secure EL1 is using AArch32, then any of the following operations, executed in Secure EL1, is trapped to Secure EL2 using EC syndrome value `0x00` :
  - Execution of an SRS instruction that uses R13_mon.
  - Execution of an MRS (Banked register) or MSR (Banked register) instruction that would access SPSR_mon, R13_mon, or R14_mon.

Note

If the Effective value of SCR_EL3.EEL2 is 0, then these operations executed in Secure EL1 using AArch32 are trapped to EL3.

A Secure only implementation that does not implement EL3 but implements EL2, behaves as if SCR_EL3.EEL2 == 1.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

API, bit [17]

When FEAT_PAuth is implemented:

Controls the use of the following instructions related to Pointer Authentication.

- PACGA.
- AUTDA, AUTDB, AUTDZA, AUTDZB, AUTIA, AUTIA1716, AUTIASP, AUTIAZ, AUTIB, AUTIB1716, AUTIBSP, AUTIBZ, AUTIZA, AUTIZB, PACDA, PACDB, PACDZA, PACDZB, PACIA, PACIA1716, PACIASP, PACIAZ, PACIB, PACIB1716, PACIBSP, PACIBZ, PACIZA, PACIZB, RETAA, RETAB, BRAA, BRAB, BLRAA, BLRAB, BRAAZ, BRABZ, BLRAAZ, BLRABZ, ERETAA, ERETAB, LDRAA and LDRAB, when any of the following are true:
  - In EL0, when the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and the associated SCTLR_EL1.En<N><M> == 1.
  - In EL0, when the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, and the associated SCTLR_EL2.En<N><M> == 1.
  - In EL1, when the associated SCTLR_EL1.En<N><M> == 1.
  - In EL2, when the associated SCTLR_EL2.En<N><M> == 1.
- When FEAT_PAuth_LR is implemented, AUTIASPPC, AUTIASPPCR, AUTIA171615, AUTIBSPPC, AUTIBSPPCR, AUTIB171615, PACIASPPC, PACNBIASPPC, PACIA171615, PACIBSPPC, PACNBIBSPPC, PACIB171615, RETAASPPC, RETAASPPCR, RETABSPPC, RETABSPPCR, when any of the following are true:
  - In EL0, when the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, and the associated SCTLR_EL1.En<N><M> == 1.
  - In EL0, when the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, and the associated SCTLR_EL2.En<N><M> == 1.
  - In EL1, when the associated SCTLR_EL1.En<N><M> == 1.
  - In EL2, when the associated SCTLR_EL2.En<N><M> == 1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

API	Meaning
0b0	The specified instructions are trapped to EL3, when the instructions are enabled, unless they are trapped to EL2 as a result of the higher priority HCR_EL2.API trap.
0b1	This control does not cause any instructions to be trapped.

Traps are reported using EC syndrome value 0x09.

An instruction is trapped only if Pointer Authentication is enabled for that instruction, for more information, see 'PAC generation and verification keys'.

Note

If FEAT_PAuth is implemented but EL3 is not implemented, the system behaves as if this bit is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

APK, bit [16]

When FEAT_PAuth is implemented:

Trap registers holding "key" values for Pointer Authentication. Traps accesses to the following registers, using EC syndrome value 0x18, from EL1 or EL2 to EL3 unless they are trapped to EL2 as a result of the HCR_EL2.APK bit or other traps:

- APIAKeyLo_EL1, APIAKeyHi_EL1, APIBKeyLo_EL1, APIBKeyHi_EL1.
- APDAKeyLo_EL1, APDAKeyHi_EL1, APDBKeyLo_EL1, APDBKeyHi_EL1.
- APGAKeyLo_EL1, and APGAKeyHi_EL1.

APK	Meaning
0b0	Access to the registers holding "key" values for pointer authentication from EL1 or EL2 are trapped to EL3 unless they are trapped to EL2 as a result of the HCR_EL2.APK bit or other traps.
0b1	This control does not cause any instructions to be trapped.

For more information, see 'PAC generation and verification keys'.

Note

If FEAT_PAuth is implemented but EL3 is not implemented, the system behaves as if this bit is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TERR, bit [15]

When FEAT_RAS is implemented:

Trap accesses of Error Record registers. Enables a trap to EL3 on accesses of Error Record registers.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TERR	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Accesses of the specified Error Record registers at EL2 and EL1 are trapped to EL3, unless the instruction generates a higher priority exception.

In AArch64 state, the instructions affected by this control are:

- MRS and MSR accesses to ERRSELR_EL1, ERXADDR_EL1, ERXCTLR_EL1, ERXMISC0_EL1, ERXMISC1_EL1, and ERXSTATUS_EL1.
- MRS accesses to ERRIDR_EL1 and ERXFR_EL1.
- If FEAT_RASv1p1 is implemented, MRS and MSR accesses to ERXMISC2_EL1 and ERXMISC3_EL1.
- If FEAT_RASv2 is implemented, MRS accesses to ERXGSR_EL1.

In AArch32 state, the instructions affected by this control are:

- MRC and MCR accesses to ERRSELR, ERXADDR, ERXADDR2, ERXCTLR, ERXCTLR2, ERXMISC0, ERXMISC1, ERXMISC2, ERXMISC3, and ERXSTATUS.
- MRC accesses to ERRIDR, ERXFR, and ERXFR2.
- If FEAT_RASv1p1 is implemented, MRC and MCR accesses to ERXMISC4, ERXMISC5, ERXMISC6, and ERXMISC7.

Unless the instruction generates a higher priority exception, trapped instructions generate an exception to EL3.

Trapped AArch64 instructions are reported using EC syndrome value 0x18.

Trapped AArch32 instructions are reported using EC syndrome value 0x03.

Accessing this field has the following behavior:

- This field is permitted to be RES0 if all of the following are true:
  - ERRSELR_EL1 and all ERX* registers are implemented as UNDEFINED or RAZ/WI.
  - ERRIDR_EL1.NUM is zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TLOR, bit [14]

When FEAT_LOR is implemented:

Trap LOR registers. Traps Non-secure and Realm accesses to the LORSA_EL1, LOREA_EL1, LORN_EL1, LORC_EL1, and LORID_EL1 registers from EL1 and EL2 to EL3, unless the access has been trapped to EL2.

TLOR	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Non-secure and Realm EL1 and EL2 accesses to the LOR registers that are not UNDEFINED are trapped to EL3, unless it is trapped by HCR_EL2.TLOR.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TWE, bit [13]

Traps EL2, EL1, and EL0 execution of WFE instructions to EL3, from any Security state and both Execution states, reported using EC syndrome value `0x01`.

When FEAT_WFxT is implemented, this trap also applies to the WFET instruction.

TWE	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	Any attempt to execute a WFE instruction at any Exception level lower than EL3 is trapped to EL3, if the instruction would otherwise have caused the PE to enter a low-power state and it is not trapped by SCTLR.nTWE, HCR.TWE, SCTLR_EL1.nTWE, SCTLR_EL2.nTWE, or HCR_EL2.TWE.

In AArch32 state, the attempted execution of a conditional WFE instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

For more information about when WFE instructions can cause the PE to enter a low-power state, see 'Wait for Event mechanism and Send event'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TWI, bit [12]

Traps EL2, EL1, and EL0 execution of WFI instructions to EL3, from any Security state and both Execution states, reported using EC syndrome value `0x01`.

When FEAT_WFxT is implemented, this trap also applies to the WFIT instruction.

TWI	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	Any attempt to execute a WFI instruction at any Exception level lower than EL3 is trapped to EL3, if the instruction would otherwise have caused the PE to enter a low-power state and it is not trapped by SCTLR.nTWI, HCR.TWI, SCTLR_EL1.nTWI, SCTLR_EL2.nTWI, or HCR_EL2.TWI.

In AArch32 state, the attempted execution of a conditional WFI instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

For more information about when WFI instructions can cause the PE to enter a low-power state, see 'Wait for Interrupt'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ST, bit [11]

Traps Secure EL1 accesses to the Counter-timer Physical Secure timer registers to EL3, from AArch64 state only, reported using EC syndrome value `0x18`.

ST	Meaning
`0b0`	Secure EL1 using AArch64 accesses to the CNTPS_TVAL_EL1, CNTPS_CTL_EL1, and CNTPS_CVAL_EL1 are trapped to EL3 when Secure EL2 is disabled. If Secure EL2 is enabled, the behavior is as if the value of this field was `0b1`.
`0b1`	This control does not cause any instructions to be trapped.

Note

Accesses to the Counter-timer Physical Secure timer registers are always enabled at EL3. These registers are not accessible at EL0.

When FEAT_RME is implemented and Secure state is not implemented, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

RW, bit [10]

When FEAT_AA32EL1 is implemented:

Execution state control for lower Exception levels.

RW	Meaning
`0b0`	All lower Exception levels are using AArch32.
`0b1`	The next lower Exception level is using AArch64: • If EL2 is implemented and enabled in the Security state determined by SCR_EL3.{NSE,NS}, then EL2 is using AArch64 and EL2 controls the Execution state for EL1. • If EL2 is not implemented or EL2 is disabled in the Security state determined by SCR_EL3.{NSE,NS}, then EL1 is using AArch64. When executing at EL0, if the next higher Exception level is using AArch64, then the Execution state is determined by PSTATE.nRW. Otherwise, EL0 uses AArch32.

If any of the following apply, then the Effective value of this bit is 1:

- EL2 is implemented and does not support AArch32, and SCR_EL3.NS is 1.
- The Effective value of SCR_EL3.{EEL2,NS} is {1,0}.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAO/WI.

SIF, bit [9]

Secure instruction fetch. When the PE is in Secure state, this bit disables instruction execution from memory marked in the first stage of translation as being Non-secure.

SIF	Meaning
0b0	Secure state instruction execution from memory marked in the first stage of translation as being Non-secure is permitted.
0b1	Secure state instruction execution from memory marked in the first stage of translation as being Non-secure is not permitted.

When FEAT_RME is implemented and Secure state is not implemented, this bit is RES0.

When FEAT_PAN3 is implemented, it is IMPLEMENTATION DEFINED whether SCR_EL3.SIF is also used to determine instruction access permission for the purpose of PAN.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

HCE, bit [8]

Hypervisor Call instruction enable. Enables HVC instructions at EL3 and, if EL2 is enabled in the current Security state, at EL2 and EL1, in both Execution states, reported using EC syndrome value 0x00.

HCE	Meaning
0b0	HVC instructions are UNDEFINED.
0b1	HVC instructions are enabled at EL3, EL2, and EL1.

Note

HVC instructions are always UNDEFINED at EL0 and, if Secure EL2 is disabled, at Secure EL1. Any resulting exception is taken from the current Exception level to the current Exception level.

If EL2 is not implemented, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SMD, bit [7]

Secure Monitor Call disable. Disables SMC instructions at EL1 and above, from any Security state and both Execution states, reported using EC syndrome value 0x00.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SMD	Meaning
0b0	SMC instructions are enabled at EL3, EL2 and EL1.
0b1	SMC instructions are UNDEFINED.

Note

SMC instructions are always UNDEFINED at EL0. Any resulting exception is taken from the current Exception level to the current Exception level.
If HCR_EL2.TSC or HCR.TSC traps attempted EL1 execution of SMC instructions to EL2, that trap has priority over this disable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [6]

Reserved, RES0.

Bits [5:4]

Reserved, RES1.

EA, bit [3]

External Abort and SError exception routing.

EA	Meaning
0b0	When executing at Exception levels below EL3, External aborts and SError exceptions are not taken to EL3. In addition, when executing at EL3: • SError exceptions are not taken. • External aborts are taken to EL3.
0b1	When executing at any Exception level, External aborts and SError exceptions are taken to EL3.

This field has no effect on the routing of virtual or delegated SError exceptions.

For more information, see 'Asynchronous exception routing'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FIQ, bit [2]

Physical FIQ Routing.

FIQ	Meaning
0b0	When executing at Exception levels below EL3, physical FIQ interrupts are not taken to EL3. When executing at EL3, physical FIQ interrupts are not taken.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

0b1	When executing at any Exception level, physical FIQ interrupts are taken to EL3.

For more information, see ‘Asynchronous exception routing’.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRQ, bit [1]

Physical IRQ Routing.

IRQ	Meaning
0b0	When executing at Exception levels below EL3, physical IRQ interrupts are not taken to EL3. When executing at EL3, physical IRQ interrupts are not taken.
0b1	When executing at any Exception level, physical IRQ interrupts are taken to EL3.

For more information, see ‘Asynchronous exception routing’.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NS, bit [0]

When FEAT_RME is implemented:

Non-secure bit. This field is used in combination with SCR_EL3.NSE to select the Security state of EL2 and lower Exception levels.

NSE	NS	Meaning
0b0	0b0	Secure.
0b0	0b1	Non-secure.
0b1	0b0	Reserved.
0b1	0b1	Realm.

When Secure state is not implemented, SCR_EL3.NS is RES1 and its Effective value is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Non-secure bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

NS	Meaning
0b0	Indicates that EL0 and EL1 are in Secure state. When FEAT_SEL2 is implemented and SCR_EL3.EEL2 == 1, then EL2 is using AArch64 and in Secure state.
0b1	Indicates that Exception levels lower than EL3 are in Non-secure state, so memory accesses from those Exception levels cannot access Secure memory.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SCR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SCR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0001	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCR_EL3;
```

```
MSR SCR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0001	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    SCR_EL3 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.170 SCTLR2_EL1, System Control Register (EL1)

The SCTLR2_EL1 characteristics are:

Purpose

Provides top-level control of the system, including its memory system, at EL1 and EL0.

Configuration

This register is present only when FEAT_SCTLR2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SCTLR2_EL1 are UNDEFINED.

Attributes

SCTLR2_EL1 is a 64-bit register.

Field descriptions

Bits [63:13]

Reserved, RES0.

CPTM0, bit [12]

When FEAT_CPA2 is implemented:

This field controls unprivileged Checked Pointer Arithmetic for Multiplication.

CPTM0	Meaning
0b0	Pointer Arithmetic for Multiplication is not checked.
0b1	Pointer Arithmetic for Multiplication is checked.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution.

This field is ignored by the PE and treated as zero when any of the following are true:

- EL3 is implemented and SCR_EL3.SCTLR2En == 0.
- EL2 is implemented and enabled in the current Security state and the Effective value of HCRX_EL2.SCTLR2En is 0.

If the Effective value of SCTLR2_EL1.CPTA0 is 0, then the Effective value of this field is 0.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CPTM, bit [11]

When FEAT_CPA2 is implemented:

This field controls Checked Pointer Arithmetic for Multiplication at EL1.

CPTM	Meaning
`0b0`	Pointer Arithmetic for Multiplication is not checked.
`0b1`	Pointer Arithmetic for Multiplication is checked.

This field is ignored by the PE and treated as zero when any of the following are true:

- EL3 is implemented and SCR_EL3.SCTLR2En == 0.
- EL2 is implemented and enabled in the current Security state and the Effective value of HCRX_EL2.SCTLR2En is 0.

If the Effective value of SCTLR2_EL1.CPTA is 0, then the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CPTA0, bit [10]

When FEAT_CPA2 is implemented:

This field controls unprivileged Checked Pointer Arithmetic for Addition.

CPTA0	Meaning
`0b0`	Pointer Arithmetic for Addition is not checked.
`0b1`	Pointer Arithmetic for Addition is checked.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution.

This field is ignored by the PE and treated as zero when any of the following are true:

- EL3 is implemented and SCR_EL3.SCTLR2En == 0.
- EL2 is implemented and enabled in the current Security state and the Effective value of HCRX_EL2.SCTLR2En is 0.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CPTA, bit [9]

When FEAT_CPA2 is implemented:

This field controls Checked Pointer Arithmetic for Addition at EL1.

CPTA	Meaning
`0b0`	Pointer Arithmetic for Addition is not checked.
`0b1`	Pointer Arithmetic for Addition is checked.

This field is ignored by the PE and treated as zero when any of the following are true:

- EL3 is implemented and SCR_EL3.SCTLR2En == 0.
- EL2 is implemented and enabled in the current Security state and the Effective value of HCRX_EL2.SCTLR2En is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnPACM0, bit [8]

When FEAT_PAuth_LR is implemented:

PACM Enable at EL0. Controls the effect of a PACM instruction at EL0.

EnPACM0	Meaning
`0b0`	The effects of PACM are disabled at EL0.
`0b1`	A PACM instruction at EL0 causes PSTATE.PACM to be set to `0b1`.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

This field is ignored by the PE and treated as zero when any of the following are true:

- EL3 is implemented and SCR_EL3.SCTLR2En == 0.
- EL2 is implemented and enabled in the current Security state and the Effective value of HCRX_EL2.SCTLR2En is 0.
- EL2 is implemented and enabled in the current Security state and the Effective value of HCRX_EL2.PACMEn == 0.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnPACM, bit [7]

When FEAT_PAuth_LR is implemented:

PACM Enable at EL1. Controls the effect of a PACM instruction at EL1.

EnPACM	Meaning
`0b0`	The effects of PACM are disabled at EL1.
`0b1`	A PACM instruction at EL1 causes PSTATE.PACM to be set to `0b1`.

This field is ignored by the PE and treated as zero when any of the following are true:

- EL3 is implemented and SCR_EL3.SCTLR2En == 0.
- EL2 is implemented and enabled in the current Security state and the Effective value of HCRX_EL2.SCTLR2En is 0.
- EL2 is implemented and enabled in the current Security state and the Effective value of HCRX_EL2.PACMEn == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnIDCP128, bit [6]

When FEAT_SYSREG128 is implemented:

Enables access to IMPLEMENTATION DEFINED 128-bit System registers.

EnIDCP128	Meaning
`0b0`	Accesses at EL0 to IMPLEMENTATION DEFINED 128-bit System registers are trapped to EL1 using an ESR_EL1.EC value of `0x14`, unless the access generates a higher priority exception. Disables the functionality of the 128-bit IMPLEMENTATION DEFINED System registers that are accessible at EL1.
`0b1`	No accesses are trapped by this control.

This field is ignored by the PE and treated as zero when any of the following are true:

- EL3 is implemented and SCR_EL3.SCTLR2En == 0.
- EL2 is implemented and enabled in the current Security state and the Effective value of HCRX_EL2.SCTLR2En is 0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EASE, bit [5]

When FEAT_DoubleFault2 is implemented:

External Aborts to SError exception vector.

EASE	Meaning
`0b0`	Synchronous External abort exceptions taken to EL1 are taken to the appropriate synchronous exception vector offset from VBAR_EL1.
`0b1`	Synchronous External abort exceptions taken to EL1 are taken to the appropriate SError exception vector offset from VBAR_EL1.

This field is ignored by the PE and treated as zero when any of the following are true:

- EL3 is implemented and SCR_EL3.SCTLR2En == 0.
- EL2 is implemented and enabled in the current Security state and the Effective value of HCRX_EL2.SCTLR2En is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnANERR, bit [4]

When FEAT_ANERR is implemented:

Enable Asynchronous Normal Read Error.

EnANERR	Meaning
`0b0`	External aborts on Normal memory reads generate synchronous Data Abort exceptions in the EL1&0 translation regime.
`0b1`	External aborts on Normal memory reads generate synchronous Data Abort or asynchronous SError exceptions in the EL1&0 translation regime.

Implementation-specific exceptions to applications of this field are described in ‘Taking error exceptions’.

Setting this field to 0 does not guarantee that the PE is able to take a synchronous Data Abort exception for an External abort on a Normal memory read in every case. There might be implementation-specific circumstances when an error on a load cannot be taken synchronously. These circumstances should be rare enough that treating such occurrences as fatal does not cause a significant increase in failure rate.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

If FEAT_SVE is implemented, SCTLR2_EL1.EnANERR is 0, and the access generating the External abort is due to any Active element of an SVE Non-fault vector load instruction or an Active element that is not the First active element of an SVE First-fault vector load instruction, then no exception is generated and the External abort is reported in the FFR.

Setting this field to 0 might have a performance impact for Normal memory reads.

This field is ignored by the PE and treated as zero when any of the following are true:

- EL3 is implemented and SCR_EL3.SCTLR2En == 0.
- EL2 is implemented and enabled in the current Security state and the Effective value of HCRX_EL2.SCTLR2En is 0.
- EL2 is implemented and enabled in the current Security state and the Effective value of HCRX_EL2.EnSNERR is 1.

Otherwise, this field is ignored by the PE and treated as one when all of the following are true:

- FEAT_ADERR is implemented.
- ID_AA64MMFR3_EL1.ANERR reads as `0b0010`.
- SCTLR2_EL1.EnADERR is 1.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnADERR, bit [3]

When FEAT_ADERR is implemented:

Enable Asynchronous Device Read Error.

EnADERR	Meaning
`0b0`	External aborts on Device memory reads generate synchronous Data Abort exceptions in the EL1&0 translation regime.
`0b1`	External aborts on Device memory reads generate synchronous Data Abort or asynchronous SError exceptions in the EL1&0 translation regime.

Implementation-specific exceptions to applications of this field are described in ‘Taking error exceptions’.

Setting this field to 0 does not guarantee that the PE is able to take a synchronous Data Abort exception for an External abort on a Device memory read in every case. There might be implementation-specific circumstances when an error on a load cannot be taken synchronously. These circumstances should be rare enough that treating such occurrences as fatal does not cause a significant increase in failure rate.

If FEAT_SVE is implemented, SCTLR2_EL1.EnADERR is 0, and the access generating the External abort is due to any Active element of an SVE Non-fault vector load instruction or an Active element that is not the First active element of an SVE First-fault vector load instruction, then no exception is generated and the External abort is reported in the FFR.

Setting this field to 0 might have a performance impact for Device memory reads.

This field is ignored by the PE and treated as zero when any of the following are true:

- EL3 is implemented and SCR_EL3.SCTLR2En == 0.
- EL2 is implemented and enabled in the current Security state and the Effective value of HCRX_EL2.SCTLR2En is 0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- EL2 is implemented and enabled in the current Security state and the Effective value of HCRX_EL2.EnSDERR is 1.

Otherwise, this field is ignored by the PE and treated as one when all of the following are true:

- FEAT_ANERR is implemented.
- ID_AA64MMFR3_EL1.ADERR reads as `0b0010`.
- SCTLR2_EL1.EnANERR is 1.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NMEA, bit [2]

When FEAT_DoubleFault2 is implemented:

Non-maskable External Aborts. Controls whether PSTATE.A masks physical SError exceptions at EL1.

NMEA	Meaning
`0b0`	Physical SError exceptions are not taken at EL1 if PSTATE.A == 1, unless routed to a higher Exception level.
`0b1`	Physical SError exceptions are taken at EL1 regardless of the value of PSTATE.A, unless routed to a higher Exception level.

This field has no effect on the masking of virtual and delegated SError interrupts.

This field is ignored by the PE and treated as zero when any of the following are true:

- EL3 is implemented and SCR_EL3.SCTLR2En == 0.
- EL2 is implemented and enabled in the current Security state and the Effective value of HCRX_EL2.SCTLR2En is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [1:0]

Reserved, RES0.

Accessing SCTLR2_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name `SCTLR2_EL1` or `SCTLR2_EL12` are not guaranteed to be ordered with respect to accesses using the other accessor name.

If FEAT_SRMASK is implemented, accesses to SCTLR2_EL1 are masked by SCTLR2MASK_EL1.

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, SCTLR2_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_SCTLR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SCTLR2En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.SCTLR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SCTLR2En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SCTLR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x278];
    else
        X[t, 64] = SCTLR2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SCTLR2En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SCTLR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = SCTLR2_EL2;
    else
        X[t, 64] = SCTLR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCTLR2_EL1;
```

```
MSR SCTLR2_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_SCTLR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SCTLR2En == '0' then
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        elsif EL2Enabled() && HCR_EL2.TVM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
             ↪HFGWTR_EL2.SCTLR_EL1 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SCTLR2En == '0') then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.SCTLR2En == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EffectiveHCR_EL2_NVx() IN {'111'} then
            NVMem[0x278] = X[t, 64];
        else
            if IsFeatureImplemented(FEAT_SRMASK) then
                SCTLR2_EL1 = (X[t, 64] AND NOT EffectiveSCTLR2MASK_EL1()) OR (SCTLR2_EL1 AND
                     ↪EffectiveSCTLR2MASK_EL1());
            else
                SCTLR2_EL1 = X[t, 64];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SCTLR2En == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.SCTLR2En == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif ELIsInHost(EL2) then
            if IsFeatureImplemented(FEAT_SRMASK) then
                SCTLR2_EL2 = (X[t, 64] AND NOT EffectiveSCTLR2MASK_EL2()) OR (SCTLR2_EL2 AND
                     ↪EffectiveSCTLR2MASK_EL2());
            else
                SCTLR2_EL2 = X[t, 64];
        else
            SCTLR2_EL1 = X[t, 64];
    elsif PSTATE.EL == EL3 then
        SCTLR2_EL1 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, SCTLR2_EL12

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0000	0b011

```
    if !(IsFeatureImplemented(FEAT_SCTLR2) && IsFeatureImplemented(FEAT_AA64)) then
        UNDEFINED;
    elsif PSTATE.EL == EL0 then
        UNDEFINED;
    elsif PSTATE.EL == EL1 then
        if EffectiveHCR_EL2_NVx() == '101' then
            X[t, 64] = NVMem[0x278];
        elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if ELIsInHost(EL2) then
            if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SCTLR2En == '0' then
                UNDEFINED;
            elsif HaveEL(EL3) && SCR_EL3.SCTLR2En == '0' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
                if EL3SDDUndef() then
                    UNDEFINED;
                else
                    AArch64.SystemAccessTrap(EL3, 0x18);
            else
                X[t, 64] = SCTLR2_EL1;
        else
            UNDEFINED;
    elsif PSTATE.EL == EL3 then
        if ELIsInHost(EL2) then
            X[t, 64] = SCTLR2_EL1;
        else
            UNDEFINED;
```

When FEAT_VHE is implemented MSR SCTLR2_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_SCTLR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x278] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SCTLR2En == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.SCTLR2En == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            SCTLR2_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        SCTLR2_EL1 = X[t, 64];
    else
        UNDEFINED;
```

When FEAT_SRMASK is implemented MRS <Xt>, SCTLR2ALIAS_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b111

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_SCTLR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SCTLR2En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
         ↪'0') || HFGRTR2_EL2.nSCTLR2ALIAS_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SCTLR2En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SCTLR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x278];
    else
        X[t, 64] = SCTLR2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SCTLR2En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SCTLR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = SCTLR2_EL2;
    else
        X[t, 64] = SCTLR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCTLR2_EL1;
```

When FEAT_SRMASK is implemented MSR SCTLR2ALIAS_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b111

```
if !(IsFeatureImplemented(FEAT_SCTLR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SCTLR2En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
         ↪'0') || HFGWTR2_EL2.nSCTLR2ALIAS_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SCTLR2En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SCTLR2En == '0' then
        if EL3SDDUndef() then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x278] = X[t, 64];
    else
        if IsFeatureImplemented(FEAT_SRMASK) then
            SCTLR2_EL1 = (X[t, 64] AND NOT EffectiveSCTLR2MASK_EL1()) OR (SCTLR2_EL1 AND
                ↪EffectiveSCTLR2MASK_EL1());
        else
            SCTLR2_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SCTLR2En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SCTLR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_SRMASK) then
            SCTLR2_EL2 = (X[t, 64] AND NOT EffectiveSCTLR2MASK_EL2()) OR (SCTLR2_EL2 AND
                ↪EffectiveSCTLR2MASK_EL2());
        else
            SCTLR2_EL2 = X[t, 64];
    else
        SCTLR2_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCTLR2_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.171 SCTLR2_EL2, System Control Register (EL2)

The SCTLR2_EL2 characteristics are:

Purpose

Provides top-level control of the system, including its memory system, at EL2.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, these controls also apply to execution at EL0.

Configuration

This register has no effect if EL2 is not enabled in the current Security state.

This register is present only when FEAT_SCTLR2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SCTLR2_EL2 are UNDEFINED.

Attributes

SCTLR2_EL2 is a 64-bit register.

Field descriptions

Bits [63:13]

Reserved, RES0.

CPTM0, bit [12]

When FEAT_CPA2 is implemented and ELIsInHost(EL2):

This field controls unprivileged Checked Pointer Arithmetic for Multiplication.

CPTM0	Meaning
0b0	Pointer Arithmetic for Multiplication is not checked.
0b1	Pointer Arithmetic for Multiplication is checked.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

When EL3 is implemented and SCR_EL3.SCTLR2En == 0, this field is ignored by the PE and treated as zero.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

If the Effective value of SCTLR2_EL2.CPTA0 is 0, then the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CPTM, bit [11]

When FEAT_CPA2 is implemented:

This field controls Checked Pointer Arithmetic for Multiplication at EL2.

CPTM	Meaning
`0b0`	Pointer Arithmetic for Multiplication is not checked.
`0b1`	Pointer Arithmetic for Multiplication is checked.

When EL3 is implemented and SCR_EL3.SCTLR2En == 0, this field is ignored by the PE and treated as zero.

If the Effective value of SCTLR2_EL2.CPTA is 0, then the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CPTA0, bit [10]

When FEAT_CPA2 is implemented and ELIsInHost(EL2):

This field controls unprivileged Checked Pointer Arithmetic for Addition.

CPTA0	Meaning
`0b0`	Pointer Arithmetic for Addition is not checked.
`0b1`	Pointer Arithmetic for Addition is checked.

If HCR_EL2.TGE == `0b0`, the field is IGNORED for all purposes other than direct reads and writes of the register.

When EL3 is implemented and SCR_EL3.SCTLR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- When the highest implemented Exception level is EL2, this field resets to `'0'`.
- Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CPTA, bit [9]

When FEAT_CPA2 is implemented:

This field controls Checked Pointer Arithmetic for Addition at EL2.

CPTA	Meaning
`0b0`	Pointer Arithmetic for Addition is not checked.
`0b1`	Pointer Arithmetic for Addition is checked.

When EL3 is implemented and SCR_EL3.SCTLR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnPACM0, bit [8]

When FEAT_PAuth_LR is implemented and ELIsInHost(EL2):

PACM Enable at EL0. Controls the effect of a PACM instruction at EL0.

EnPACM0	Meaning
`0b0`	The effects of PACM are disabled at EL0.
`0b1`	A PACM instruction at EL0 causes PSTATE.PACM to be set to `0b1`.

If HCR_EL2.TGE == `0b0`, the field is IGNORED for all purposes other than direct reads and writes of the register.

When EL3 is implemented and SCR_EL3.SCTLR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnPACM, bit [7]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_PAuth_LR is implemented:

PACM Enable at EL2. Controls the effect of a PACM instruction at EL2.

EnPACM	Meaning
`0b0`	The effects of PACM are disabled at EL2.
`0b1`	A PACM instruction at EL2 causes PSTATE.PACM to be set to `0b1`.

When EL3 is implemented and SCR_EL3.SCTLR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnIDCP128, bit [6]

When FEAT_SYSREG128 is implemented:

Enables access to IMPLEMENTATION DEFINED 128-bit System registers.

EnIDCP128	Meaning
`0b0`	Accesses at EL0 to IMPLEMENTATION DEFINED 128-bit System registers are trapped to EL2 using an ESR_EL2.EC value of `0x14`, unless the access generates a higher priority exception. Disables the functionality of the 128-bit IMPLEMENTATION DEFINED System registers that are accessible at EL2.
`0b1`	No accesses are trapped by this control.

When EL3 is implemented and SCR_EL3.SCTLR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EASE, bit [5]

When FEAT_DoubleFault2 is implemented:

External Aborts to SError exception vector.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EASE	Meaning
0b0	Synchronous External abort exceptions taken to EL2 are taken to the appropriate synchronous exception vector offset from VBAR_EL2.
0b1	Synchronous External abort exceptions taken to EL2 are taken to the appropriate SError exception vector offset from VBAR_EL2.

When EL3 is implemented and SCR_EL3.SCTLR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnANERR, bit [4]

When FEAT_ANERR is implemented:

Enable Asynchronous Normal Read Error.

EnANERR	Meaning
0b0	External aborts on Normal memory reads generate synchronous Data Abort exceptions in the EL2 and EL2&0 translation regimes.
0b1	External aborts on Normal memory reads generate synchronous Data Abort or asynchronous SError exceptions in the EL2 and EL2&0 translation regimes.

Implementation-specific exceptions to applications of this field are described in ‘Taking error exceptions’.

Setting this field to 0 does not guarantee that the PE is able to take a synchronous Data Abort exception for an External abort on a Normal memory read in every case. There might be implementation-specific circumstances when an error on a load cannot be taken synchronously. These circumstances should be rare enough that treating such occurrences as fatal does not cause a significant increase in failure rate.

If FEAT_SVE is implemented, SCTLR2_EL2.EnANERR is 0, and the access generating the External abort is due to any Active element of an SVE Non-fault vector load instruction or an Active element that is not the First active element of an SVE First-fault vector load instruction, then no exception is generated and the External abort is reported in the FFR.

Setting this field to 0 might have a performance impact for Normal memory reads.

When EL3 is implemented and SCR_EL3.SCTLR2En == 0, this field is ignored by the PE and treated as zero.

Otherwise, this field is ignored by the PE and treated as one when all of the following are true:

- FEAT_ADERR is implemented.
- ID_AA64MMFR3_EL1.ANERR reads as 0b0010.
- SCTLR2_EL2.EnADERR is 1.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

EnADERR, bit [3]

When FEAT_ADERR is implemented:

Enable Asynchronous Device Read Error.

EnADERR	Meaning
`0b0`	External aborts on Device memory reads generate synchronous Data Abort exceptions in the EL2 and EL2&0 translation regimes.
`0b1`	External aborts on Device memory reads generate synchronous Data Abort or asynchronous SError exceptions in the EL2 and EL2&0 translation regimes.

Implementation-specific exceptions to applications of this field are described in ‘Taking error exceptions’.

Setting this field to 0 does not guarantee that the PE is able to take a synchronous Data Abort exception for an External abort on a Device memory read in every case. There might be implementation-specific circumstances when an error on a load cannot be taken synchronously. These circumstances should be rare enough that treating such occurrences as fatal does not cause a significant increase in failure rate.

If FEAT_SVE is implemented, SCTLR2_EL2.EnADERR is 0, and the access generating the External abort is due to any Active element of an SVE Non-fault vector load instruction or an Active element that is not the First active element of an SVE First-fault vector load instruction, then no exception is generated and the External abort is reported in the FFR.

Setting this field to 0 might have a performance impact for Device memory reads.

When EL3 is implemented and SCR_EL3.SCTLR2En == 0, this field is ignored by the PE and treated as zero.

Otherwise, this field is ignored by the PE and treated as one when all of the following are true:

- FEAT_ANERR is implemented.
- ID_AA64MMFR3_EL1.ADERR reads as `0b0010`.
- SCTLR2_EL2.EnANERR is 1.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NMEA, bit [2]

When FEAT_DoubleFault2 is implemented:

Non-maskable External Aborts. Controls whether physical SError exceptions are masked at EL2 when PSTATE.A is 1 or HCR_EL2.{AMO, TGE} is {0, 0}.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

NMEA	Meaning
0b0	The masking of physical SError exceptions at EL2 is unaffected by this mechanism.
0b1	Physical SError exceptions are taken at EL2 regardless of whether they are masked by any mechanism, unless they are routed to EL3.

This field has no effect on the masking of delegated SError interrupts.

When EL3 is implemented and SCR_EL3.SCTLR2En == 0, this field is ignored by the PE and treated as zero.

For more information, see 'Asynchronous exception routing'.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EMEC, bit [1]

When FEAT_MEC is implemented:

Enables MEC. When enabled, memory accesses to the Realm physical address space are associated with a MECID.

EMEC	Meaning
0b0	MEC is not enabled for the Realm physical address space.
0b1	MEC is enabled for the Realm physical address space.

This bit is permitted to be cached in a TLB.

When EL3 is implemented and SCR_EL3.SCTLR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [0]

Reserved, RES0.

Accessing SCTLR2_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name SCTLR2_EL2 or SCTLR2_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

If FEAT_SRMASK is implemented, accesses to SCTLR2_EL2 are masked by SCTLR2MASK_EL2.

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, SCTLR2_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_SCTLR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SCTLR2En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SCTLR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SCTLR2_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCTLR2_EL2;
```

```
MSR SCTLR2_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_SCTLR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SCTLR2En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SCTLR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if IsFeatureImplemented(FEAT_SRMASK) then
            SCTLR2_EL2 = (X[t, 64] AND NOT EffectiveSCTLR2MASK_EL2()) OR (SCTLR2_EL2 AND
                 ↪EffectiveSCTLR2MASK_EL2());
        else
            SCTLR2_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL3 then
    SCTLR2_EL2 = X[t, 64];
```

MRS <Xt>, SCTLR2_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_SCTLR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SCTLR2En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.SCTLR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SCTLR2En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SCTLR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x278];
    else
        X[t, 64] = SCTLR2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SCTLR2En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SCTLR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = SCTLR2_EL2;
    else
        X[t, 64] = SCTLR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCTLR2_EL1;
```

MSR SCTLR2_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0000	0b011

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_SCTLR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SCTLR2En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.SCTLR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SCTLR2En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SCTLR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x278] = X[t, 64];
    else
        if IsFeatureImplemented(FEAT_SRMASK) then
            SCTLR2_EL1 = (X[t, 64] AND NOT EffectiveSCTLR2MASK_EL1()) OR (SCTLR2_EL1 AND
                 ↪EffectiveSCTLR2MASK_EL1());
        else
            SCTLR2_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SCTLR2En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SCTLR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_SRMASK) then
            SCTLR2_EL2 = (X[t, 64] AND NOT EffectiveSCTLR2MASK_EL2()) OR (SCTLR2_EL2 AND
                 ↪EffectiveSCTLR2MASK_EL2());
        else
            SCTLR2_EL2 = X[t, 64];
    else
        SCTLR2_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCTLR2_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.172 SCTLR2_EL3, System Control Register (EL3)

The SCTLR2_EL3 characteristics are:

Purpose

Provides top-level control of the system, including its memory system, at EL3.

Configuration

This register is present only when FEAT_SCTLR2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SCTLR2_EL3 are UNDEFINED.

Attributes

SCTLR2_EL3 is a 64-bit register.

Field descriptions

Bits [63:12]

Reserved, RES0.

CPTM, bit [11]

When FEAT_CPA2 is implemented:

This field controls Checked Pointer Arithmetic for Multiplication at EL3.

CPTM	Meaning
0b0	Pointer Arithmetic for Multiplication is not checked.
0b1	Pointer Arithmetic for Multiplication is checked.

If the Effective value of SCTLR2_EL3.CPTA is 0, then the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3, this field resets to '0'.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bit [10]

Reserved, RES0.

CPTA, bit [9]

When FEAT_CPA2 is implemented:

This field controls Checked Pointer Arithmetic for Addition at EL3.

CPTA	Meaning
`0b0`	Pointer Arithmetic for Addition is not checked.
`0b1`	Pointer Arithmetic for Addition is checked.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3, this field resets to `'0'`.

Otherwise:

Reserved, RES0.

Bit [8]

Reserved, RES0.

EnPACM, bit [7]

When FEAT_PAuth_LR is implemented:

PACM Enable at EL3. Controls the effect of a PACM instruction at EL3.

EnPACM	Meaning
`0b0`	The effects of PACM are disabled at EL3.
`0b1`	A PACM instruction at EL3 causes PSTATE.PACM to be set to `0b1`.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3, this field resets to `'0'`.

Otherwise:

Reserved, RES0.

Bits [6:5]

Reserved, RES0.

EnANERR, bit [4]

When FEAT_ANERR is implemented:

Enable Asynchronous Normal Read Error.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EnANERR	Meaning
0b0	External aborts on Normal memory reads generate synchronous Data Abort exceptions in the EL3 translation regime.
0b1	External aborts on Normal memory reads generate synchronous Data Abort or asynchronous SError exceptions in the EL3 translation regime.

Implementation-specific exceptions to applications of this field are described in 'Taking error exceptions'.

Setting this field to 0 does not guarantee that the PE is able to take a synchronous Data Abort exception for an External abort on a Normal memory read in every case. There might be implementation-specific circumstances when an error on a load cannot be taken synchronously. These circumstances should be rare enough that treating such occurrences as fatal does not cause a significant increase in failure rate.

If FEAT_SVE is implemented, SCTLR2_EL3.EnANERR is 0, and the access generating the External abort is due to any Active element of an SVE Non-fault vector load instruction or an Active element that is not the First active element of an SVE First-fault vector load instruction, then no exception is generated and the External abort is reported in the FFR.

Setting this field to 0 might have a performance impact for Normal memory reads.

This field is ignored by the PE and treated as one when all of the following are true:

- FEAT_ADERR is implemented.
- ID_AA64MMFR3_EL1.ANERR reads as `0b0010`.
- SCTLR2_EL3.EnADERR is 1.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3, this field resets to `'0'`.

Otherwise:

Reserved, RES0.

EnADERR, bit [3]

When FEAT_ADERR is implemented:

Enable Asynchronous Device Read Error.

EnADERR	Meaning
0b0	External aborts on Device memory reads generate synchronous Data Abort exceptions in the EL3 translation regime.
0b1	External aborts on Device memory reads generate synchronous Data Abort or asynchronous SError exceptions in the EL3 translation regime.

Implementation-specific exceptions to applications of this field are described in 'Taking error exceptions'.

Setting this field to 0 does not guarantee that the PE is able to take a synchronous Data Abort exception for an External abort on a Device memory read in every case. There might be implementation-specific circumstances when an error on a load cannot be taken synchronously. These circumstances should be rare enough that treating such occurrences as fatal does not cause a significant increase in failure rate.

If FEAT_SVE is implemented, SCTLR2_EL3.EnADERR is 0, and the access generating the External abort is due to any Active element of an SVE Non-fault vector load instruction or an Active element that is not the

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

First active element of an SVE First-fault vector load instruction, then no exception is generated and the External abort is reported in the FFR.

Setting this field to 0 might have a performance impact for Device memory reads.

This field is ignored by the PE and treated as one when all of the following are true:

- FEAT_ANERR is implemented.
- ID_AA64MMFR3_EL1.ADERR reads as `0b0010`.
- SCTLR2_EL3.EnANERR is 1.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3, this field resets to `'0'`.

Otherwise:

Reserved, RES0.

Bit [2]

Reserved, RES0.

EMEC, bit [1]

When FEAT_MEC is implemented:

Enables MEC. When enabled, memory accesses to the Realm physical address space are associated with MECID_RL_A_EL3.

EMEC	Meaning
`0b0`	MEC is not enabled for the Realm physical address space.
`0b1`	MEC is enabled for the Realm physical address space.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3, this field resets to `'0'`.

Otherwise:

Reserved, RES0.

Bit [0]

Reserved, RES0.

Accessing SCTLR2_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SCTLR2_EL3
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_SCTLR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCTLR2_EL3;
```

```
MSR SCTLR2_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_SCTLR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.SCTLR2_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SCTLR2_EL3 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.173 SCTLR2MASK_EL1, Extended System Control Masking Register (EL1)

The SCTLR2MASK_EL1 characteristics are:

Purpose

Mask register to prevent updates of fields in SCTLR2_EL1 on writes to SCTLR2_EL1 or SCTLR2ALIAS_EL1.

Configuration

This register is present only when FEAT_SRMASK is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SCTLR2MASK_EL1 are UNDEFINED.

Attributes

SCTLR2MASK_EL1 is a 64-bit register.

Field descriptions

Bits [63:13]

Reserved, RES0.

CPTM0, bit [12]

When FEAT_CPA2 is implemented:

Mask bit for CPTM0.

CPTM0	Meaning
0b0	SCTLR2_EL1.CPTM0 is writeable.
0b1	SCTLR2_EL1.CPTM0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CPTM, bit [11]

When FEAT_CPA2 is implemented:

Mask bit for CPTM.

CPTM	Meaning
0b0	SCTLR2_EL1.CPTM is writeable.
0b1	SCTLR2_EL1.CPTM is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CPTA0, bit [10]

When FEAT_CPA2 is implemented:

Mask bit for CPTA0.

CPTA0	Meaning
0b0	SCTLR2_EL1.CPTA0 is writeable.
0b1	SCTLR2_EL1.CPTA0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CPTA, bit [9]

When FEAT_CPA2 is implemented:

Mask bit for CPTA.

CPTA	Meaning
0b0	SCTLR2_EL1.CPTA is writeable.
0b1	SCTLR2_EL1.CPTA is not writeable.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnPACM0, bit [8]

When FEAT_PAuth_LR is implemented:

Mask bit for EnPACM0.

EnPACM0	Meaning
`0b0`	SCTLR2_EL1.EnPACM0 is writeable.
`0b1`	SCTLR2_EL1.EnPACM0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnPACM, bit [7]

When FEAT_PAuth_LR is implemented:

Mask bit for EnPACM.

EnPACM	Meaning
`0b0`	SCTLR2_EL1.EnPACM is writeable.
`0b1`	SCTLR2_EL1.EnPACM is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnIDCP128, bit [6]

When FEAT_SYSREG128 is implemented:

Mask bit for EnIDCP128.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EnIDCP128	Meaning
0b0	SCTLR2_EL1.EnIDCP128 is writeable.
0b1	SCTLR2_EL1.EnIDCP128 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EASE, bit [5]

When FEAT_DoubleFault2 is implemented:

Mask bit for EASE.

EASE	Meaning
0b0	SCTLR2_EL1.EASE is writeable.
0b1	SCTLR2_EL1.EASE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnANERR, bit [4]

When FEAT_ANERR is implemented:

Mask bit for EnANERR.

EnANERR	Meaning
0b0	SCTLR2_EL1.EnANERR is writeable.
0b1	SCTLR2_EL1.EnANERR is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EnADERR, bit [3]

When FEAT_ADERR is implemented:

Mask bit for EnADERR.

EnADERR	Meaning
0b0	SCTLR2_EL1.EnADERR is writeable.
0b1	SCTLR2_EL1.EnADERR is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NMEA, bit [2]

When FEAT_DoubleFault2 is implemented:

Mask bit for NMEA.

NMEA	Meaning
0b0	SCTLR2_EL1.NMEA is writeable.
0b1	SCTLR2_EL1.NMEA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [1:0]

Reserved, RES0.

Accessing SCTLR2MASK_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name SCTLR2MASK_EL1 or SCTLR2MASK_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

MRS <Xt>, SCTLR2MASK_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b011

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HFGRTR2_EL2.nSCTLR2MASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x328];
    else
        X[t, 64] = SCTLR2MASK_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = SCTLR2MASK_EL2;
    else
        X[t, 64] = SCTLR2MASK_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCTLR2MASK_EL1;
```

MSR SCTLR2MASK_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b011

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HFGWTR2_EL2.nSCTLR2MASK_EL1 == '0') then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EffectiveHCR_EL2_NVx() IN {'111'} then
            NVMem[0x328] = X[t, 64];
        elsif !IsZero(EffectiveSCTLR2MASK_EL1()) then
            UNDEFINED;
        else
            SCTLR2MASK_EL1 = X[t, 64];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif ELIsInHost(EL2) then
            if !IsZero(EffectiveSCTLR2MASK_EL2()) then
                UNDEFINED;
            else
                SCTLR2MASK_EL2 = X[t, 64];
        else
            SCTLR2MASK_EL1 = X[t, 64];
    elsif PSTATE.EL == EL3 then
        SCTLR2MASK_EL1 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, SCTLR2MASK_EL12

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0100	0b011

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x328];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = SCTLR2MASK_EL1;
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = SCTLR2MASK_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented MSR SCTLR2MASK_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0100	0b011

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x328] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            SCTLR2MASK_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        SCTLR2MASK_EL1 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.174 SCTLR2MASK_EL2, Extended System Control Masking Register (EL2)

The SCTLR2MASK_EL2 characteristics are:

Purpose

Mask register to prevent updates of fields in SCTLR2_EL2 on writes.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

This register is present only when FEAT_SRMASK is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SCTLR2MASK_EL2 are UNDEFINED.

Attributes

SCTLR2MASK_EL2 is a 64-bit register.

Field descriptions

Bits [63:13]

Reserved, RES0.

CPTM0, bit [12]

When FEAT_CPA2 is implemented:

Mask bit for CPTM0.

CPTM0	Meaning
0b0	SCTLR2_EL2.CPTM0 is writeable.
0b1	SCTLR2_EL2.CPTM0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

CPTM, bit [11]

When FEAT_CPA2 is implemented:

Mask bit for CPTM.

CPTM	Meaning
0b0	SCTLR2_EL2.CPTM is writeable.
0b1	SCTLR2_EL2.CPTM is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CPTA0, bit [10]

When FEAT_CPA2 is implemented:

Mask bit for CPTA0.

CPTA0	Meaning
0b0	SCTLR2_EL2.CPTA0 is writeable.
0b1	SCTLR2_EL2.CPTA0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CPTA, bit [9]

When FEAT_CPA2 is implemented:

Mask bit for CPTA.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CPTA	Meaning
0b0	SCTLR2_EL2.CPTA is writeable.
0b1	SCTLR2_EL2.CPTA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnPACM0, bit [8]

When FEAT_PAuth_LR is implemented:

Mask bit for EnPACM0.

EnPACM0	Meaning
0b0	SCTLR2_EL2.EnPACM0 is writeable.
0b1	SCTLR2_EL2.EnPACM0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnPACM, bit [7]

When FEAT_PAuth_LR is implemented:

Mask bit for EnPACM.

EnPACM	Meaning
0b0	SCTLR2_EL2.EnPACM is writeable.
0b1	SCTLR2_EL2.EnPACM is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EnIDCP128, bit [6]

When FEAT_SYSREG128 is implemented:

Mask bit for EnIDCP128.

EnIDCP128	Meaning
0b0	SCTLR2_EL2.EnIDCP128 is writeable.
0b1	SCTLR2_EL2.EnIDCP128 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EASE, bit [5]

When FEAT_DoubleFault2 is implemented:

Mask bit for EASE.

EASE	Meaning
0b0	SCTLR2_EL2.EASE is writeable.
0b1	SCTLR2_EL2.EASE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnANERR, bit [4]

When FEAT_ANERR is implemented:

Mask bit for EnANERR.

EnANERR	Meaning
0b0	SCTLR2_EL2.EnANERR is writeable.
0b1	SCTLR2_EL2.EnANERR is not writeable.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnADERR, bit [3]

When FEAT_ADERR is implemented:

Mask bit for EnADERR.

EnADERR	Meaning
`0b0`	SCTLR2_EL2.EnADERR is writeable.
`0b1`	SCTLR2_EL2.EnADERR is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NMEA, bit [2]

When FEAT_DoubleFault2 is implemented:

Mask bit for NMEA.

NMEA	Meaning
`0b0`	SCTLR2_EL2.NMEA is writeable.
`0b1`	SCTLR2_EL2.NMEA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EMEC, bit [1]

When FEAT_MEC is implemented:

Mask bit for EMEC.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EMEC	Meaning
0b0	SCTLR2_EL2.EMEC is writeable.
0b1	SCTLR2_EL2.EMEC is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [0]

Reserved, RES0.

Accessing SCTLR2MASK_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name SCTLR2MASK_EL2 or SCTLR2MASK_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SCTLR2MASK_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0100	0b011

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SCTLR2MASK_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCTLR2MASK_EL2;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MSR SCTLR2MASK_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0100	0b011

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsZero(EffectiveSCTLR2MASK_EL2()) then
        UNDEFINED;
    else
        SCTLR2MASK_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCTLR2MASK_EL2 = X[t, 64];
```

```
MRS <Xt>, SCTLR2MASK_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b011

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HFGRTR2_EL2.nSCTLR2MASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x328];
    else
        X[t, 64] = SCTLR2MASK_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = SCTLR2MASK_EL2;
    else
        X[t, 64] = SCTLR2MASK_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCTLR2MASK_EL1;
```

MSR SCTLR2MASK_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b011

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HFGWTR2_EL2.nSCTLR2MASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x328] = X[t, 64];
    elsif !IsZero(EffectiveSCTLR2MASK_EL1()) then
        UNDEFINED;
    else
        SCTLR2MASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        if !IsZero(EffectiveSCTLR2MASK_EL2()) then
            UNDEFINED;
        else
            SCTLR2MASK_EL2 = X[t, 64];
    else
        SCTLR2MASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCTLR2MASK_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.175 SCTLR_EL1, System Control Register (EL1)

The SCTLR_EL1 characteristics are:

Purpose

Provides top-level control of the system, including its memory system, at EL1 and EL0.

Configuration

AArch64 System register SCTLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register SCTLR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to SCTLR_EL1 are UNDEFINED.

Attributes

SCTLR_EL1 is a 64-bit register.

Field descriptions

TIDCP, bit [63]

When FEAT_TIDCP1 is implemented:

Trap IMPLEMENTATION DEFINED functionality. When the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, traps EL0 accesses to the encodings reserved for IMPLEMENTATION DEFINED functionality to EL1.

TIDCP	Meaning
0b0	No instructions accessing the System register or System instruction spaces are trapped by this mechanism.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TIDCP	Meaning

0b1	Instructions accessing the following System register or System instruction spaces are trapped to EL1 by this mechanism:
	• In AArch64 state, EL0 access to the encodings in the following reserved encoding spaces are trapped:
	  • IMPLEMENTATION DEFINED System instructions, which are accessed using SYS and SYSL, with CRn == {11, 15}, and are reported using EC syndrome value 0x18.
	  • IMPLEMENTATION DEFINED System instructions, which are accessed using SYSP, with CRn == {11, 15}, and are reported using EC syndrome value 0x14.
	  • IMPLEMENTATION DEFINED System registers, which are accessed using MRS and MSR with the S3_<op1>_<Cn>_<Cm>_<op2> register name, and are reported using EC syndrome value 0x18.
	  • IMPLEMENTATION DEFINED System registers, which are accessed using MRRS and MSRR with the S3_<op1>_<Cn>_<Cm>_<op2> register name, and are reported using EC syndrome value 0x14.
	• In AArch32 state, EL0 MCR and MRC access to the following encodings are trapped and reported using EC syndrome value 0x03:
	  • All coproc==p15, CRn==c9, opc1 == {0-7}, CRm == {c0-c2, c5-c8}, opc2 == {0-7}.
	  • All coproc==p15, CRn==c10, opc1 ==={0-7}, CRm == {c0, c1, c4, c8}, opc2 == {0-7}.
	  • All coproc==p15, CRn==c11, opc1==={0-7}, CRm == {c0-c8, c15}, opc2 == {0-7}.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SPINTMASK, bit [62]

When FEAT_NMI is implemented:

SP Interrupt Mask enable. When SCTLR_EL1.NMI is 1, controls whether PSTATE.SP acts as an interrupt mask, and controls the value of PSTATE.ALLINT on taking an exception to EL1.

SPINTMASK	Meaning
0b0	Does not cause PSTATE.SP to mask interrupts. PSTATE.ALLINT is set to 1 on taking an exception to EL1.
0b1	When PSTATE.SP is 1 and execution is at EL1, an IRQ or FIQ interrupt that is targeted to EL1 is masked regardless of any indication of Superpriority. PSTATE.ALLINT is set to 0 on taking an exception to EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NMI, bit [61]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_NMI is implemented:

Non-maskable Interrupt enable.

NMI	Meaning
`0b0`	This control does not affect interrupt masking behavior.
`0b1`	This control enables all of the following: • The use of the PSTATE.ALLINT interrupt mask. • IRQ and FIQ interrupts to have Superpriority as an additional attribute. • PSTATE.SP to be used as an interrupt mask.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnTP2, bit [60]

When FEAT_SME is implemented:

Traps instructions executed at EL0 that access TPIDR2_EL0 to EL1, or to EL2 when EL2 is implemented and enabled for the current Security state and HCR_EL2.TGE is 1. The exception is reported using EC syndrome value `0x18`.

EnTP2	Meaning
`0b0`	This control causes execution of these instructions at EL0 to be trapped.
`0b1`	This control does not cause execution of any instructions to be trapped.

If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this field has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCSO, bit [59]

When FEAT_MTE_STORE_ONLY is implemented:

Tag Checking Store Only.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TCSO	Meaning
0b0	This field has no effect on Tag checking.
0b1	Explicit Memory Read Effects generated by instructions executed in EL1 are Tag Unchecked.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCSO0, bit [58]

When FEAT_MTE_STORE_ONLY is implemented:

When the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, Tag Checking Store Only in EL0.

TCSO0	Meaning
0b0	This field has no effect on Tag checking.
0b1	Explicit Memory Read Effects generated by instructions executed in EL0 are Tag Unchecked.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EPAN, bit [57]

When FEAT_PAN3 is implemented:

Enhanced Privileged Access Never. When PSTATE.PAN is 1, determines whether an EL1 data access to a page with stage 1 EL0 instruction access permission generates a Permission fault as a result of the Privileged Access Never mechanism.

EPAN	Meaning
0b0	No additional Permission faults are generated by this mechanism.
0b1	An EL1 data access to a page with stage 1 EL0 data access permission or stage 1 EL0 instruction access permission generates a Permission fault. Any speculative data accesses that would generate a Permission fault as a result of PSTATE.PAN = 1 if the accesses were not speculative, will not cause an allocation into a cache. When executing at EL1, this does not prevent unprivileged speculative accesses generated from the EL0 hardware-defined context from causing allocation into a cache.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

EnALS, bit [56]

When FEAT_LS64 is implemented:

When the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, traps execution of an LD64B or ST64B instruction at EL0 to EL1.

EnALS	Meaning
0b0	Execution of an LD64B or ST64B instruction at EL0 is trapped to EL1.
0b1	This control does not cause any instructions to be trapped.

A trap of an LD64B or ST64B instruction is reported using EC syndrome value 0x0A, with an ISS code of 0x0000002.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnAS0, bit [55]

When FEAT_LS64_ACCDATA is implemented:

When the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, traps execution of an ST64BV0 instruction at EL0 to EL1.

EnAS0	Meaning
0b0	Execution of an ST64BV0 instruction at EL0 is trapped to EL1.
0b1	This control does not cause any instructions to be trapped.

A trap of an ST64BV0 instruction is reported using EC syndrome value 0x0A, with an ISS code of 0x0000001.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnASR, bit [54]

When FEAT_LS64_V is implemented:

When the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, traps execution of an ST64BV instruction at EL0 to EL1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EnASR	Meaning
0b0	Execution of an ST64BV instruction at EL0 is trapped to EL1.
0b1	This control does not cause any instructions to be trapped.

A trap of an ST64BV instruction is reported using EC syndrome value 0x0A, with an ISS code of 0x0000000.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [53:50]

Reserved, RES0.

TWEDEL, bits [49:46]

When FEAT_TWED is implemented:

TWE Delay. A 4-bit unsigned number that, when SCTLR_EL1.TWEDEn is 1, encodes the minimum delay in taking a trap of WFE* caused by SCTLR_EL1.nTWE as $2^{(TWEDEL + 8)}$ cycles.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TWEDEn, bit [45]

When FEAT_TWED is implemented:

TWE Delay Enable. Enables a configurable delayed trap of the WFE* instruction caused by SCTLR_EL1.nTWE.

TWEDEn	Meaning
0b0	The delay for taking the trap is IMPLEMENTATION DEFINED.
0b1	The delay for taking the trap is at least the number of cycles defined in SCTLR_EL1.TWEDEL.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DSSBS, bit [44]

When FEAT_SSBS is implemented:

Default PSTATE.SSBS value on Exception Entry.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

DSSBS	Meaning
0b0	PSTATE.SSBS is set to 0 on an exception to EL1.
0b1	PSTATE.SSBS is set to 1 on an exception to EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RES0.

ATA, bit [43]

When FEAT_MTE2 is implemented:

Allocation Tag Access in EL1.

Controls use of Memory tagging in EL1.

ATA	Meaning
0b0	Use of Memory tagging is disabled at EL1.
0b1	Use of Memory tagging is enabled at EL1.

The Effective value of this field is 0 if any of the following are true:

- SCR_EL3.ATA is 0.
- HCR_EL2.ATA is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ATA0, bit [42]

When FEAT_MTE2 is implemented:

Allocation Tag Access in EL0.

Controls access to Allocation Tags and Tag Check operations in EL0 when HCR_EL2,{E2H,TGE} is not {1,1}.

ATA0	Meaning
0b0	Use of Memory tagging is disabled at EL0.
0b1	Use of Memory tagging is enabled at EL0.

The Effective value of this field is 0 if any of the following are true:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- SCR_EL3.ATA is 0.
- HCR_EL2.ATA is 0.

Note

Software may change this control bit on a context switch.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCF, bits [41:40]

When FEAT_MTE2 is implemented:

Tag Check Fault in EL1. Controls the effect of Tag Check Faults due to Loads and Stores in EL1.

TCF	Meaning	Applies when
0b00	Tag Check Faults have no effect on the PE.	
0b01	Tag Check Faults cause a synchronous exception.	
0b10	Tag Check Faults are asynchronously accumulated.	
0b11	Tag Check Faults cause a synchronous exception on reads, and are asynchronously accumulated on writes.	FEAT_MTE_ASYM_FAULT is implemented

If FEAT_MTE3 is not implemented, the value 0b11 is reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCF0, bits [39:38]

When FEAT_MTE2 is implemented:

Tag Check Fault in EL0. When the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, controls the effect of Tag Check Faults due to Loads and Stores in EL0.

TCF0	Meaning	Applies when
0b00	Tag Check Faults have no effect on the PE.	
0b01	Tag Check Faults cause a synchronous exception.	
0b10	Tag Check Faults are asynchronously accumulated.	

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TCF0	Meaning	Applies when
0b11	Tag Check Faults cause a synchronous exception on reads, and are asynchronously accumulated on writes.	FEAT_MTE_ASYM_FAULT is implemented

If FEAT_MTE3 is not implemented, the value 0b11 is reserved.

Note

Software may change this control bit on a context switch.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ITFSB, bit [37]

When FEAT_MTE_ASYNC is implemented:

When synchronous exceptions are not being generated by Tag Check Faults, this field controls whether on exception entry into EL1, all Tag Check Faults due to instructions executed before exception entry, that are reported asynchronously, are synchronized into TFSRE0_EL1 and TFSR_EL1 registers.

ITFSB	Meaning
0b0	Tag Check Faults are not synchronized on entry to EL1.
0b1	Tag Check Faults are synchronized on entry to EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BT1, bit [36]

When FEAT_BTI is implemented:

Configures the Branch Type compatibility of the implicit BTI behavior for the following instructions at EL1:

- PACIASP.
- PACIBSP.
- If FEAT_PAuth_LR is implemented, PACIASPPC.
- If FEAT_PAuth_LR is implemented, PACIBSPPC.

BT1	Meaning
0b0	When the PE is executing at EL1, when the specified instructions have an implicit BTI behavior, they are compatible with the same BTYPE values as BTI.jc.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

BT1	Meaning
0b1	When the PE is executing at EL1, when the specified instructions have an implicit BTI behavior, they are compatible with the same BTYPE values as BTI.c.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BT0, bit [35]

When FEAT_BTI is implemented:

Configures the Branch Type compatibility of the implicit BTI behavior for the following instructions at EL0:

- PACIASP.
- PACIBSP.
- If FEAT_PAuth_LR is implemented, PACIASPPC.
- If FEAT_PAuth_LR is implemented, PACIBSPPC.

BT0	Meaning
0b0	When the PE is executing at EL0, when the specified instructions have an implicit BTI behavior, they are compatible with the same BTYPE values as BTI.jc.
0b1	When the PE is executing at EL0, when the specified instructions have an implicit BTI behavior, they are compatible with the same BTYPE values as BTI.c.

When the value of the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the value of SCTLR_EL1.BT0 has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnFPM, bit [34]

When FEAT_FPMR is implemented:

Enables direct and indirect accesses to FPMR from EL0.

When accesses to FPMR are disabled by this control:

- Direct accesses to FPMR from EL0 are trapped to EL1, or to EL2 when EL2 is implemented and enabled in the current Security state and HCR_EL2.TGE is 1. These exceptions are reported using EC syndrome value 0x18.
- Execution of FP8 data-processing instructions that indirectly access FPMR is UNDEFINED at EL0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EnFPM	Meaning
0b0	Direct and indirect accesses to FPMR are disabled at EL0.
0b1	This control does not cause any instructions to be trapped.

Traps are not taken if there is a higher priority exception generated by the access.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0, and the SCTLR_EL2.EnFPM control is used for this purpose.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MSCEn, bit [33]

When FEAT_MOPS is implemented and !ELIsInHost(EL0):

Memory Copy and Memory Set instructions Enable. Enables execution of the Memory Copy and Memory Set instructions at EL0.

MSCEn	Meaning
0b0	Execution of the Memory Copy and Memory Set instructions is UNDEFINED at EL0.
0b1	This control does not cause any instructions to be UNDEFINED.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the Effective value of this bit is 0b1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CMOW, bit [32]

When FEAT_CMOW is implemented:

Controls the required permissions for the following cache maintenance instructions executed at EL0:

- Any DC instruction that operates by VA and performs a clean and invalidate operation.
- Any IC instruction that operates by VA.

CMOW	Meaning
0b0	This control does not cause any instruction to generate a stage 1 Permission fault.
0b1	For these instructions, when executed at EL0, the absence of stage 1 unprivileged write permission generates a stage 1 permission fault.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

For more information, see:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- Stage 1 permissions.
- Implications of enabling the dirty state management mechanism.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnIA, bit [31]

When FEAT_PAuth is implemented:

Controls enabling of pointer authentication of instruction addresses, using the APIAKey_EL1 key, at EL1 and at EL0 in the EL1&0 translation regime.

EnIA	Meaning
0b0	Pointer authentication of instruction addresses at EL1 and at EL0 in the EL1&0 translation regime, using the APIAKey_EL1 key, is not enabled.
0b1	Pointer authentication of instruction addresses at EL1 and at EL0 in the EL1&0 translation regime, using the APIAKey_EL1 key, is enabled.

Note

This field controls the behavior of the AddPACIA and AuthIA pseudocode functions. When pointer authentication is enabled, AddPACIA returns a copy of a pointer to which a pointer authentication code has been added, and AuthIA returns an authenticated copy of a pointer. When pointer authentication is disabled, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnIB, bit [30]

When FEAT_PAuth is implemented:

Controls enabling of pointer authentication of instruction addresses, using the APIBKey_EL1 key, at EL1 and at EL0 in the EL1&0 translation regime.

EnIB	Meaning
0b0	Pointer authentication of instruction addresses at EL1 and at EL0 in the EL1&0 translation regime, using the APIBKey_EL1 key, is not enabled.
0b1	Pointer authentication of instruction addresses at EL1 and at EL0 in the EL1&0 translation regime, using the APIBKey_EL1 key, is enabled.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Note

This field controls the behavior of the AddPACIB and AuthIB pseudocode functions. When pointer authentication is enabled, AddPACIB returns a copy of a pointer to which a pointer authentication code has been added, and AuthIB returns an authenticated copy of a pointer. When pointer authentication is disabled, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

LSMAOE, bit [29]

When FEAT_LSMAOC is implemented:

Load Multiple and Store Multiple Atomicity and Ordering Enable.

LSMAOE	Meaning
0b0	For all memory accesses at EL0, T32 and A32 Load Multiple and Store Multiple can have an interrupt taken during the sequence memory accesses, and the memory accesses are not required to be ordered.
0b1	The ordering and interrupt behavior of T32 and A32 Load Multiple and Store Multiple at EL0 is as defined for Armv8.0.

This bit is permitted to be cached in a TLB.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

nTLSMD, bit [28]

When FEAT_LSMAOC is implemented:

No Trap Load Multiple and Store Multiple to Device-nGRE/Device-nGnRE/Device-nGnRnE memory.

nTLSMD	Meaning
0b0	All memory accesses by T32 and A32 Load Multiple and Store Multiple at EL0 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are trapped and generate a stage 1 Alignment fault.
0b1	All memory accesses by T32 and A32 Load Multiple and Store Multiple at EL0 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are not trapped.

This bit is permitted to be cached in a TLB.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

EnDA, bit [27]

When FEAT_PAuth is implemented:

Controls enabling of pointer authentication of data addresses, using the APDAKey_EL1 key, at EL1 and at EL0 in the EL1&0 translation regime.

EnDA	Meaning
0b0	Pointer authentication of data addresses at EL1 and at EL0 in the EL1&0 translation regime, using the APDAKey_EL1 key, is not enabled.
0b1	Pointer authentication of data addresses at EL1 and at EL0 in the EL1&0 translation regime, using the APDAKey_EL1 key, is enabled.

Note

This field controls the behavior of the AddPACDA and AuthDA pseudocode functions. When pointer authentication is enabled, AddPACDA returns a copy of a pointer to which a pointer authentication code has been added, and AuthDA returns an authenticated copy of a pointer. When pointer authentication is disabled, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UCI, bit [26]

Traps EL0 execution of cache maintenance instructions, to EL1, or to EL2 when it is implemented and enabled in the current Security state and HCR_EL2.TGE is 1, from AArch64 state only, reported using EC syndrome value 0x18, as follows:

- DC CVAU, DC CIVAC, DC CVAC, and IC IVAU.
- If FEAT_MTE is implemented, DC CIGVAC, DC CIGDVAC, DC CGVAC, and DC CGDVAC.
- If FEAT_DPB is implemented, DC CVAP.
- If FEAT_DPB and FEAT_MTE are implemented, DC CGVAP and DC CGDVAP.
- If FEAT_DPB2 is implemented, DC CVADP.
- If FEAT_DPB2 and FEAT_MTE are implemented, DC CGVADP and DC CGDVADP.
- If FEAT_OCCMO is implemented, DC CIVAOC, DC CIGDVAOC, DC CVAOC and DC CGDVAOC.

UCI	Meaning
0b0	For each of the specified instructions, if the execution of the instruction can be trapped, access at EL0 using AArch64 is trapped.
0b1	This control does not cause any instructions to be trapped.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EE, bit [25]

When FEAT_MixedEnd is implemented:

Endianness of data accesses at EL1, and stage 1 translation table walks in the EL1&0 translation regime.

EE	Meaning
0b0	Explicit data accesses at EL1, and stage 1 translation table walks in the EL1&0 translation regime are little-endian.
0b1	Explicit data accesses at EL1, and stage 1 translation table walks in the EL1&0 translation regime are big-endian.

The EE bit is permitted to be cached in a TLB.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

When FEAT_BigEnd is implemented:

Explicit data accesses at EL1, and stage 1 translation table walks in the EL1&0 translation regime are big-endian.

Reserved, RES1.

Otherwise:

Explicit data accesses at EL1, and stage 1 translation table walks in the EL1&0 translation regime are little-endian.

Reserved, RES0.

E0E, bit [24]

When FEAT_MixedEndEL0 is implemented:

Endianness of data accesses at EL0.

E0E	Meaning
0b0	Explicit data accesses at EL0 are little-endian.
0b1	Explicit data accesses at EL0 are big-endian.

This bit has no effect on the endianness of Explicit Memory Effects generated by unprivileged memory access instructions executed at EL1.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_BigEndEL0 is implemented:

Explicit data accesses at EL0 are big-endian.

Reserved, RES1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Explicit data accesses at EL0 are little-endian.

Reserved, RES0.

SPAN, bit [23]

When FEAT_PAN is implemented:

Set Privileged Access Never, on taking an exception to EL1.

SPAN	Meaning
`0b0`	PSTATE.PAN is set to 1 on taking an exception to EL1.
`0b1`	The value of PSTATE.PAN is left unchanged on taking an exception to EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

EIS, bit [22]

When FEAT_ExS is implemented:

Exception Entry is a context synchronization event.

EIS	Meaning
`0b0`	The taking of an exception to EL1 is not a context synchronization event.
`0b1`	The taking of an exception to EL1 is a context synchronization event.

If SCTLR_EL1.EIS is set to `0b0`:

- Indirect writes to ESR_EL1, FAR_EL1, SPSR_EL1, ELR_EL1 are synchronized on exception entry to EL1, so that a direct read of the register after exception entry sees the indirectly written value caused by the exception entry.
- Memory transactions, including instruction fetches, from an Exception level always use the translation resources associated with that translation regime.
- Exception Catch debug events are synchronous debug events.
- DCPS* and DRPS instructions are context synchronization events.
- Some exception entries reported are not considered IFBEs per the memory model.

The following are not affected by the value of SCTLR_EL1.EIS:

- Changes to the PSTATE information on entry to EL1.
- Behavior of accessing the banked copies of the stack pointer using the SP register name for loads, stores and data processing instructions.
- The memory model requirement for exception entry to generate an Instruction Fetch Barrier Effect for some exception entries. See Basic definitions for the list of exception entries.
- Exit from Debug state.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

IESB, bit [21]

When FEAT_IESB is implemented:

Implicit Error Synchronization event enable. Possible values are:

IESB	Meaning
0b0	Disabled.
0b1	An implicit error synchronization event is added: • At each exception taken to EL1. • Before the operational pseudocode of each ERET instruction executed at EL1.

If FEAT_DoubleFault2 is implemented, the PE is in Non-debug state, and the Effective value of SCTLR2_EL1.NMEA is 1, then SCTLR_EL1.IESB is ignored and the PE behaves as if SCTLR_EL1.IESB is 1 for all purposes other than direct read of the register.

When the PE is in Debug state, the effect of this field is CONSTRAINED UNPREDICTABLE, and its Effective value might be 0 or 1 regardless of the value of the field. If the Effective value of the field is 1, then an implicit error synchronization event is added after each DCPSx instruction taken to EL1 and before each DRPS instruction executed at EL1, in addition to the other cases where it is added.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TSCXT, bit [20]

When FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented:

Trap EL0 Access to the SCXTNUM_EL0 register, when EL0 is using AArch64.

TSCXT	Meaning
0b0	EL0 access to SCXTNUM_EL0 is not disabled by this mechanism.
0b1	EL0 access to SCXTNUM_EL0 is disabled, causing an exception to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1. The value of SCXTNUM_EL0 is treated as 0.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

WXN, bit [19]

Write permission implies XN (Execute-never). For the EL1&0 translation regime, this bit can restrict execute permissions on writeable pages.

WXN	Meaning
0b0	This control has no effect on memory access permissions.
0b1	In the EL1&0 translation regime, any region of memory that is writeable at EL1 is XN at EL1, and any region of memory that is writeable at EL0 is XN at EL0.

This bit applies only when SCTLR_EL1.M bit is set.

The WXN bit is permitted to be cached in a TLB.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

This field is RES0 if TCR2_EL1.PIE is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

nTWE, bit [18]

Traps EL0 execution of WFE instructions to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, from both Execution states, reported using EC syndrome value 0x01.

When FEAT_WFxT is implemented, this trap also applies to the WFET instruction.

nTWE	Meaning
0b0	Any attempt to execute a WFE instruction at EL0 is trapped, if the instruction would otherwise have caused the PE to enter a low-power state.
0b1	This control does not cause any instructions to be trapped.

In AArch32 state, the attempted execution of a conditional WFE instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [17]

Reserved, RES0.

nTWI, bit [16]

Traps EL0 execution of WFI instructions to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, from both Execution states, reported using EC syndrome value 0x01.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_WFxT is implemented, this trap also applies to the WFIT instruction.

nTWI	Meaning
0b0	Any attempt to execute a WFI instruction at EL0 is trapped, if the instruction would otherwise have caused the PE to enter a low-power state.
0b1	This control does not cause any instructions to be trapped.

In AArch32 state, the attempted execution of a conditional WFI instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

UCT, bit [15]

Traps EL0 accesses to the CTR_EL0 to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, from AArch64 state only, reported using EC syndrome value 0x18.

UCT	Meaning
0b0	Accesses to the CTR_EL0 from EL0 using AArch64 are trapped.
0b1	This control does not cause any instructions to be trapped.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DZE, bit [14]

Traps EL0 execution of DC ZVA instructions to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, from AArch64 state only, reported using EC syndrome value 0x18.

If FEAT_MTE is implemented, this trap also applies to DC GVA and DC GZVA.

DZE	Meaning
0b0	Any attempt to execute an instruction that this trap applies to at EL0 using AArch64 is trapped. Reading DCZID_EL0.DZP from EL0 returns 1, indicating that the instructions this trap applies to are not supported.
0b1	This control does not cause any instructions to be trapped.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EnDB, bit [13]

When FEAT_PAuth is implemented:

Controls enabling of pointer authentication of data addresses, using the APDBKey_EL1 key, at EL1 and at EL0 in the EL1&0 translation regime.

EnDB	Meaning
0b0	Pointer authentication of data addresses at EL1 and at EL0 in the EL1&0 translation regime, using the APDBKey_EL1 key, is not enabled.
0b1	Pointer authentication of data addresses at EL1 and at EL0 in the EL1&0 translation regime, using the APDBKey_EL1 key, is enabled.

Note

This field controls the behavior of the AddPACDB and AuthDB pseudocode functions. When pointer authentication is enabled, AddPACDB returns a copy of a pointer to which a pointer authentication code has been added, and AuthDB returns an authenticated copy of a pointer. When pointer authentication is disabled, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

I, bit [12]

Stage 1 instruction access Cacheability control, for accesses at EL0 and EL1:

I	Meaning
0b0	All instruction access to Stage 1 Normal memory from EL0 and EL1 are Stage 1 Non-cacheable. If the value of SCTLR_EL1.M is 0, instruction accesses from stage 1 of the EL1&0 translation regime are to Normal, Outer Shareable, Inner Non-cacheable, Outer Non-cacheable memory.
0b1	This control has no effect on the Stage 1 Cacheability of instruction access to Stage 1 Normal memory from EL0 and EL1. If the value of SCTLR_EL1.M is 0, instruction accesses from stage 1 of the EL1&0 translation regime are to Normal, Outer Shareable, Inner Write-Through, Outer Write-Through memory.

When the value of the HCR_EL2.DC bit is 1, then instruction access to Normal memory from EL0 and EL1 are Cacheable regardless of the value of the SCTLR_EL1.I bit.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

The reset behavior of this field is:

- On a Warm reset:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- When the highest implemented Exception level is EL1, this field resets to `'0'`.
- Otherwise, this field resets to an architecturally UNKNOWN value.

EOS, bit [11]

When FEAT_ExS is implemented:

Exception Exit is a context synchronization event.

EOS	Meaning
`0b0`	An exception return from EL1 is not a context synchronization event
`0b1`	An exception return from EL1 is a context synchronization event

If SCTLR_EL1.EOS is set to `0b0`:

- Memory transactions, including instruction fetches, from an Exception level always use the translation resources associated with that translation regime.
- Exception Catch debug events are synchronous debug events.
- DCPS* and DRPS instructions are context synchronization events.

The following are not affected by the value of SCTLR_EL1.EOS:

- The indirect write of the PSTATE and PC values from SPSR_EL1 and ELR_EL1 on exception return is synchronized.
- Behavior of accessing the banked copies of the stack pointer using the SP register name for loads, stores and data processing instructions.
- Exit from Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

EnRCTX, bit [10]

When FEAT_SPECRES is implemented:

Enable EL0 access to the following System instructions:

- CFPRCTX, DVPRCTX and CPPRCTX instructions.
- If FEAT_SPECRES2 is implemented, COSPRCTX.
- CFP RCTX, DVP RCTX and CPP RCTX instructions.
- If FEAT_SPECRES2 is implemented, COSP RCTX.

EnRCTX	Meaning
`0b0`	EL0 access to these instructions is disabled, and these instructions are trapped to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1.
`0b1`	EL0 access to these instructions is enabled.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UMA, bit [9]

User Mask Access. Traps EL0 execution of MSR and MRS instructions that access the PSTATE.{D, A, I, F} masks to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, from AArch64 state only, reported using EC syndrome value `0x18`.

UMA	Meaning
`0b0`	Any attempt at EL0 using AArch64 to execute an `MRS`, `MSR(register)`, or `MSR(immediate)` instruction that accesses the DAIF is trapped.
`0b1`	This control does not cause any instructions to be trapped.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the value of this bit is treated as 0 for all purposes other than reading the value of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SED, bit [8]

When FEAT_AA32EL0 is implemented:

SETEND instruction disable. Disables SETEND instructions at EL0 using AArch32.

SED	Meaning
`0b0`	SETEND instruction execution is enabled at EL0 using AArch32.
`0b1`	SETEND instructions are UNDEFINED at EL0 using AArch32 and any attempt at EL0 to access a SETEND instruction generates an exception to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, reported using EC syndrome value `0x00`.

If the implementation does not support mixed-endian operation at any Exception level, this bit is RES1.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

ITD, bit [7]

When FEAT_AA32EL0 is implemented:

IT Disable. Disables some uses of IT instructions at EL0 using AArch32.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ITD	Meaning
`0b0`	All IT instruction functionality is enabled at EL0 using AArch32.

`0b1`	Any attempt at EL0 using AArch32 to execute any of the following is UNDEFINED and generates an exception, reported using EC syndrome value `0x00`, to EL1 or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1:
	• All encodings of the IT instruction with hw1[3:0]!=1000.
	• All encodings of the subsequent instruction with the following values for hw1:
	  • `0b11xxxxxxxxxxxxxx`: All 32-bit instructions, and the 16-bit instructions B, UDF, SVC, LDM, and STM.
	  • `0b1011xxxxxxxxxxxx`: All instructions in 'Miscellaneous 16-bit instructions'.
	  • `0b10100xxxxxxxxxxx`: ADD Rd, PC, #imm
	  • `0b01001xxxxxxxxxxx`: LDR Rd, [PC, #imm]
	  • `0b0100x1xxx1111xxx`: ADD Rdn, PC; CMP Rn, PC; MOV Rd, PC; BX PC; BLX PC.
	  • `0b010001xx1xxxx111`: ADD PC, Rm; CMP PC, Rm; MOV PC, Rm. This pattern also covers unpredictable cases with BLX Rn.
	These instructions are always UNDEFINED, regardless of whether they would pass or fail the condition code check that applies to them as a result of being in an IT block.
	It is IMPLEMENTATION DEFINED whether the IT instruction is treated as:
	• A 16-bit instruction, that can only be followed by another 16-bit instruction.
	• The first half of a 32-bit instruction.
	This means that, for the situations that are UNDEFINED, either the second 16-bit instruction or the 32-bit instruction is UNDEFINED.
	An implementation might vary dynamically as to whether IT is treated as a 16-bit instruction or the first half of a 32-bit instruction.

If an instruction in an active IT block that would be disabled by this field sets this field to 1, then behavior is CONSTRAINED UNPREDICTABLE. For more information, see 'Changes to an ITD control by an instruction in an IT block'.

ITD is optional, but if it is implemented in the SCTLR_EL1, then it must also be implemented in the SCTLR_EL2, HSCTLR, and SCTLR.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement ITD, access to this field is RAZ/WI.

Otherwise:

Reserved, RES1.

nAA, bit [6]

When FEAT_LSE2 is implemented:

Non-aligned access. This bit controls generation of Alignment faults at EL1 and EL0 under certain conditions.

The following instructions generate an Alignment fault if all bytes being accessed are not within a single naturally aligned 16-byte quantity, for access:

- LDAPR, LDAPRH, LDAPUR, LDAPURH, LDAPURSH, LDAPURSW, LDAR, LDARH, LDLAR, LDLARH.
- STLLR, STLLRH, STLR, STLRH, STLUR, and STLURH.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- If FEAT_LRCPC3 is implemented, the post index versions of LDAPR and the pre index versions of STLR.
- If FEAT_LRCPC3 and Advanced SIMD and floating-point instructions are implemented, LDAPUR (SIMD&FP), LDAP1 (SIMD&FP), STLUR (SIMD&FP), and STL1 (SIMD&FP). If FEAT_LRCPC3 is implemented, the following instructions generate an Alignment fault if all bytes being accessed for a single register are not within a single naturally aligned 16-byte quantity, for access:
- LDIAPP, STILP.

nAA	Meaning
0b0	Unaligned accesses by the specified instructions generate an Alignment fault.
0b1	This control does not generate Alignment faults.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

For a load-acquire instruction that does not have acquire semantics as the result of the destination register, or registers, being ZR, it is IMPLEMENTATION SPECIFIC whether this field behaves as 1 or the programmed value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CP15BEN, bit [5]

When FEAT_AA32EL0 is implemented:

System instruction memory barrier enable. Enables accesses to the DMB, DSB, and ISB System instructions in the (coproc==0b1111) encoding space from EL0:

CP15BEN	Meaning
0b0	EL0 using AArch32: EL0 execution of the CP15DMB, CP15DSB, and CP15ISB instructions is UNDEFINED and generates an exception to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1. The exception is reported using EC syndrome value 0x00.
0b1	EL0 using AArch32: EL0 execution of the CP15DMB, CP15DSB, and CP15ISB instructions is enabled.

CP15BEN is optional, but if it is implemented in the SCTLR_EL1, then it must also be implemented in the SCTLR_EL2, HSCTLR, and SCTLR.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement CP15BEN, access to this field is RAO/WI.

Otherwise:

Reserved, RES0.

SA0, bit [4]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SP Alignment check enable for EL0. When set to 1, if a load or store instruction executed at EL0 uses the SP as the base address and the SP is not aligned to a 16-byte boundary, then an SP alignment fault exception is generated. For more information, see 'SP alignment checking'.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SA, bit [3]

SP Alignment check enable. When set to 1, if a load or store instruction executed at EL1 uses the SP as the base address and the SP is not aligned to a 16-byte boundary, then an SP alignment fault exception is generated. For more information, see 'SP alignment checking'.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [2]

Stage 1 Cacheability control, for data accesses.

C	Meaning
`0b0`	All data access to Stage 1 Normal memory from EL0 and EL1, and all Normal memory accesses from unified cache to the EL1&0 Stage 1 translation tables, are treated as Stage 1 Non-cacheable.
`0b1`	This control has no effect on the Stage 1 Cacheability of: • Data access to Normal memory from EL0 and EL1. • Normal memory accesses to the EL1&0 Stage 1 translation tables.

When the Effective value of the HCR_EL2.DC bit in the current Security state is 1, the PE ignores SCTLR_EL1.C. This means that EL0 and EL1 data accesses to Normal memory are Cacheable.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

A, bit [1]

Alignment check enable. This is the enable bit for Alignment fault checking at EL1 and EL0.

A	Meaning
`0b0`	Alignment fault checking is disabled when executing at EL1 or EL0. Alignment checks on some instructions are not disabled by this control. For more information, see 'Alignment of data accesses'.
`0b1`	Alignment fault checking is enabled when executing at EL1 or EL0. All instructions that load or store one or more registers have an alignment check that the address being accessed is aligned to the size of the data element(s) being accessed. If this check fails it causes an Alignment fault, which is taken as a Data Abort exception.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, the value of this bit is treated as 0 for all purposes other than reading the value of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M, bit [0]

MMU enable for EL1&0 stage 1 address translation.

M	Meaning
0b0	EL1&0 stage 1 address translation disabled. See the SCTLR_EL1.I field for the behavior of instruction accesses to Normal memory.
0b1	EL1&0 stage 1 address translation enabled.

If the Effective value of HCR_EL2.{DC, TGE} in the current Security state is not {0, 0}, then the PE behaves as if the value of the SCTLR_EL1.M field is 0 for all purposes other than returning the value of a direct read of the field.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Accessing SCTLR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name `SCTLR_EL1` or `SCTLR_EL12` are not guaranteed to be ordered with respect to accesses using the other accessor name.

If FEAT_SRMASK is implemented, accesses to SCTLR_EL1 are masked by SCTLRMASK_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SCTLR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.SCTLR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x110];
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        X[t, 64] = SCTLR_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = SCTLR_EL2;
    else
        X[t, 64] = SCTLR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCTLR_EL1;
```

MSR SCTLR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.SCTLR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x110] = X[t, 64];
    else
        if IsFeatureImplemented(FEAT_SRMASK) then
            SCTLR_EL1 = (X[t, 64] AND NOT EffectiveSCTLRMASK_EL1()) OR (SCTLR_EL1 AND
                 ↪EffectiveSCTLRMASK_EL1());
        else
            SCTLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_SRMASK) then
            SCTLR_EL2 = (X[t, 64] AND NOT EffectiveSCTLRMASK_EL2()) OR (SCTLR_EL2 AND
                 ↪EffectiveSCTLRMASK_EL2());
        else
            SCTLR_EL2 = X[t, 64];
    else
        SCTLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCTLR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, SCTLR_EL12

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        if EffectiveHCR_EL2_NVx() == '101' then
            X[t, 64] = NVMem[0x110];
        elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if ELIsInHost(EL2) then
            X[t, 64] = SCTLR_EL1;
        else
            UNDEFINED;
    elsif PSTATE.EL == EL3 then
        if ELIsInHost(EL2) then
            X[t, 64] = SCTLR_EL1;
        else
            UNDEFINED;
```

When FEAT_VHE is implemented MSR SCTLR_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x110] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        SCTLR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        SCTLR_EL1 = X[t, 64];
    else
        UNDEFINED;
```

When FEAT_SRMASK is implemented MRS <Xt>, SCTLRALIAS_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b110

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        if EL2Enabled() && HCR_EL2.TRVM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
             ↪'0') || HFGRTR2_EL2.nSCTLRALIAS_EL1 == '0') then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EffectiveHCR_EL2_NVx() IN {'111'} then
            X[t, 64] = NVMem[0x110];
        else
            X[t, 64] = SCTLR_EL1;
    elsif PSTATE.EL == EL2 then
        if ELIsInHost(EL2) then
            X[t, 64] = SCTLR_EL2;
        else
            X[t, 64] = SCTLR_EL1;
    elsif PSTATE.EL == EL3 then
        X[t, 64] = SCTLR_EL1;
```

When FEAT_SRMASK is implemented MSR SCTLRALIAS_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b110

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
         ↪'0') || HFGWTR2_EL2.nSCTLRALIAS_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x110] = X[t, 64];
    else
        if IsFeatureImplemented(FEAT_SRMASK) then
            SCTLR_EL1 = (X[t, 64] AND NOT EffectiveSCTLRMASK_EL1()) OR (SCTLR_EL1 AND
                 ↪EffectiveSCTLRMASK_EL1());
        else
            SCTLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_SRMASK) then
            SCTLR_EL2 = (X[t, 64] AND NOT EffectiveSCTLRMASK_EL2()) OR (SCTLR_EL2 AND
                 ↪EffectiveSCTLRMASK_EL2());
        else
            SCTLR_EL2 = X[t, 64];
    else
        SCTLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCTLR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.176 SCTLR_EL2, System Control Register (EL2)

The SCTLR_EL2 characteristics are:

Purpose

Provides top-level control of the system, including its memory system, at EL2.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, these controls apply also to execution at EL0.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

AArch64 System register SCTLR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HSCTLR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to SCTLR_EL2 are UNDEFINED.

Attributes

SCTLR_EL2 is a 64-bit register.

Field descriptions

TIDCP, bit [63]

When FEAT_TIDCP1 is implemented and ELIsInHost(EL2):

Trap IMPLEMENTATION DEFINED functionality. Traps EL0 accesses to the encodings reserved for IMPLEMENTATION DEFINED functionality to EL2.

TIDCP	Meaning
0b0	No instructions accessing the System register or System instruction spaces are trapped by this mechanism.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TIDCP	Meaning

0b1	If HCR_EL2.TGE==0, no instructions accessing the System register or System instruction spaces are trapped by this mechanism.
	If HCR_EL2.TGE==1, instructions accessing the following System register or System instruction spaces are trapped to EL2 by this mechanism:
	• In AArch64 state, EL0 access to the encodings in the following reserved encoding spaces are trapped:
	  • IMPLEMENTATION DEFINED System instructions, which are accessed using SYS and SYSL, with CRn == {11, 15}, and are reported using EC syndrome value 0x18.
	  • IMPLEMENTATION DEFINED System instructions, which are accessed using SYSP, with CRn == {11, 15}, and are reported using EC syndrome value 0x14.
	  • IMPLEMENTATION DEFINED System registers, which are accessed using MRS and MSR with the S3_<op1>_<Cn>_<Cm>_<op2> register name, and are reported using EC syndrome value 0x18.
	  • IMPLEMENTATION DEFINED System registers, which are accessed using MRRS and MSRR with the S3_<op1>_<Cn>_<Cm>_<op2> register name, and are reported using EC syndrome value 0x14.
	• In AArch32 state, EL0 MCR and MRC accesses to the following encodings are trapped and reported using EC syndrome value 0x03:
	  • All coproc==p15, CRn==c9, opc1 == {0-7}, CRm == {c0-c2, c5-c8}, opc2 == {0-7}.
	  • All coproc==p15, CRn==c10, opc1 ==={0-7}, CRm == {c0, c1, c4, c8}, opc2 == {0-7}.
	  • All coproc==p15, CRn==c11, opc1==={0-7}, CRm == {c0-c8, c15}, opc2 == {0-7}.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SPINTMASK, bit [62]

When FEAT_NMI is implemented:

SP Interrupt Mask enable. When SCTLR_EL2.NMI is 1, controls whether PSTATE.SP acts as an interrupt mask, and controls the value of PSTATE.ALLINT on taking an exception to EL2.

SPINTMASK	Meaning
0b0	Does not cause PSTATE.SP to mask interrupts. PSTATE.ALLINT is set to 1 on taking an exception to EL2.
0b1	When PSTATE.SP is 1 and execution is at EL2, an IRQ or FIQ interrupt that is targeted to EL2 is masked regardless of any indication of Superpriority. PSTATE.ALLINT is set to 0 on taking an exception to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

NMI, bit [61]

When FEAT_NMI is implemented:

Non-maskable Interrupt enable.

NMI	Meaning
0b0	This control does not affect interrupt masking behavior.
0b1	This control enables all of the following: • The use of the PSTATE.ALLINT interrupt mask. • IRQ and FIQ interrupts to have Superpriority as an additional attribute. • PSTATE.SP to be used as an interrupt mask.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnTP2, bit [60]

When FEAT_SME is implemented and ELIsInHost(EL2):

Traps instructions executed at EL0 that access TPIDR2_EL0 to EL2 when EL2 is implemented and enabled for the current Security state. The exception is reported using EC syndrome value 0x18.

EnTP2	Meaning
0b0	This control causes execution of these instructions at EL0 to be trapped.
0b1	This control does not cause execution of any instructions to be trapped.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCSO, bit [59]

When FEAT_MTE_STORE_ONLY is implemented:

Tag Checking Store Only.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TCSO	Meaning
0b0	This field has no effect on Tag checking.
0b1	Explicit Memory Read Effects generated by instructions executed in EL2 are Tag Unchecked.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCSO0, bit [58]

When FEAT_MTE_STORE_ONLY is implemented and ELIsInHost(EL2):

Tag Checking Store Only in EL0.

TCSO0	Meaning
0b0	This field has no effect on Tag checking.
0b1	Explicit Memory Read Effects generated by instructions executed in EL0 are Tag Unchecked.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EPAN, bit [57]

When FEAT_PAN3 is implemented and ELIsInHost(EL2):

Enhanced Privileged Access Never. When PSTATE.PAN is 1, determines whether an EL2 data access to a page with EL0 instruction access permission generates a Permission fault as a result of the Privileged Access Never mechanism.

EPAN	Meaning
0b0	No additional Permission faults are generated by this mechanism.
0b1	An EL2 data access to a page with stage 1 EL0 data access permission or stage 1 EL0 instruction access permission generates a Permission fault. Any speculative data accesses that would generate a Permission fault as a result of PSTATE.PAN = 1 if the accesses were not speculative, will not cause an allocation into a cache. When executing at EL2, and the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this does not prevent unprivileged speculative accesses generated from the EL0 hardware-defined context from causing allocation into a cache.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Note

The value of HCR_EL2.TGE does not change the effect of this field on privileged accesses.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnALS, bit [56]

When FEAT_LS64 is implemented and ELIsInHost(EL2):

Traps execution of an LD64B or ST64B instruction at EL0 to EL2.

EnALS	Meaning
0b0	Execution of an LD64B or ST64B instruction at EL0 is trapped to EL2.
0b1	This control does not cause any instructions to be trapped.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

A trap of an LD64B or ST64B instruction is reported using EC syndrome value 0x0A, with an ISS code of 0x0000002.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnAS0, bit [55]

When FEAT_LS64_ACCDATA is implemented and ELIsInHost(EL2):

Traps execution of an ST64BV0 instruction at EL0 to EL2.

EnAS0	Meaning
0b0	Execution of an ST64BV0 instruction at EL0 is trapped to EL2.
0b1	This control does not cause any instructions to be trapped.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

A trap of an ST64BV0 instruction is reported using EC syndrome value 0x0A, with an ISS code of 0x0000001.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

EnASR, bit [54]

When FEAT_LS64_V is implemented and ELIsInHost(EL2):

Traps execution of an ST64BV instruction at EL0 to EL2.

EnASR	Meaning
`0b0`	Execution of an ST64BV instruction at EL0 is trapped to EL2.
`0b1`	This control does not cause any instructions to be trapped.

If HCR_EL2.TGE == `0b0`, the field is IGNORED for all purposes other than direct reads and writes of the register.

A trap of an ST64BV instruction is reported using EC syndrome value `0x0A`, with an ISS code of `0x0000000`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [53:50]

Reserved, RES0.

TWEDEL, bits [49:46]

When FEAT_TWED is implemented and ELIsInHost(EL2):

TWE Delay. A 4-bit unsigned number that, when SCTLR_EL2.TWEDEn is 1, encodes the minimum delay in taking a trap of WFE* caused by SCTLR_EL2.nTWE as $2^{(TWEDEL + 8)}$ cycles.

If HCR_EL2.TGE == `0b0`, the field is IGNORED for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TWEDEn, bit [45]

When FEAT_TWED is implemented and ELIsInHost(EL2):

TWE Delay Enable. Enables a configurable delayed trap of the WFE* instruction caused by SCTLR_EL2.nTWE.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TWEDEn	Meaning
0b0	The delay for taking a WFE* trap is IMPLEMENTATION DEFINED.
0b1	The delay for taking a WFE* trap is at least the number of cycles defined in SCTLR_EL2.TWEDEL.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DSSBS, bit [44]

When FEAT_SSBS is implemented:

Default PSTATE.SSBS value on exception entry.

DSSBS	Meaning
0b0	PSTATE.SSBS is set to 0 on an exception to EL2.
0b1	PSTATE.SSBS is set to 1 on an exception to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RES0.

ATA, bit [43]

When FEAT_MTE2 is implemented:

Allocation Tag Access in EL2.

Controls use of Memory tagging in EL2.

ATA	Meaning
0b0	Use of Memory tagging is disabled at EL2.
0b1	Use of Memory tagging is enabled at EL2.

If SCR_EL3.ATA is 0, then the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ATA0, bit [42]

When FEAT_MTE2 is implemented and ELIsInHost(EL2):

Allocation Tag Access in EL0.

Controls use of Memory Tagging in EL0.

ATA0	Meaning
0b0	Use of Memory tagging is disabled at EL0.
0b1	Use of Memory tagging is enabled at EL2.

If SCR_EL3.ATA is 0, then the Effective value of this field is 0.

If HCR_EL2.TGE is 0, the field is IGNORED for all purposes other than direct reads and writes of the register.

Note

Software may change this control bit on a context switch.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCF, bits [41:40]

When FEAT_MTE2 is implemented:

Tag Check Fault in EL2. Controls the effect of Tag Check Faults due to Loads and Stores in EL2.

TCF	Meaning	Applies when
0b00	Tag Check Faults have no effect on the PE.	
0b01	Tag Check Faults cause a synchronous exception.	
0b10	Tag Check Faults are asynchronously accumulated.	
0b11	Tag Check Faults cause a synchronous exception on reads, and are asynchronously accumulated on writes.	FEAT_MTE_ASYM_FAULT is implemented

If FEAT_MTE3 is not implemented, the value 0b11 is reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCF0, bits [39:38]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_MTE2 is implemented and ELIsInHost(EL2):

Tag Check Fault in EL0. Controls the effect of Tag Check Faults due to Loads and Stores in EL0.

TCF0	Meaning	Applies when
0b00	Tag Check Faults have no effect on the PE.	
0b01	Tag Check Faults cause a synchronous exception.	
0b10	Tag Check Faults are asynchronously accumulated.	
0b11	Tag Check Faults cause a synchronous exception on reads, and are asynchronously accumulated on writes.	FEAT_MTE_ASYM_FAULT is implemented

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

If FEAT_MTE3 is not implemented, the value 0b11 is reserved.

Note

Software may change this control bit on a context switch.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ITFSB, bit [37]

When FEAT_MTE_ASYNC is implemented:

When synchronous exceptions are not being generated by Tag Check Faults, this field controls whether on exception entry into EL2, all Tag Check Faults due to instructions executed before exception entry, that are reported asynchronously, are synchronized into TFSRE0_EL1, TFSR_EL1, and TFSR_EL2 registers.

ITFSB	Meaning
0b0	Tag Check Faults are not synchronized on entry to EL2.
0b1	Tag Check Faults are synchronized on entry to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BT, bit [36]

When FEAT_BTI is implemented:

Indicates the Branch Type compatibility of the implicit BTI behavior for the following instructions at EL2:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- PACIASP.
- PACIBSP.
- If FEAT_PAuth_LR is implemented, PACIASPPC.
- If FEAT_PAuth_LR is implemented, PACIBSPPC.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit is named BT1.

BT	Meaning
0b0	When the PE is executing at EL2, when the specified instructions have an implicit BTI behavior, they are compatible with the same BTYPE values as BTI.jc.
0b1	When the PE is executing at EL2, when the specified instructions have an implicit BTI behavior, they are compatible with the same BTYPE values as BTI.c.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BT0, bit [35]

When FEAT_BTI is implemented and ELIsInHost(EL2):

Indicates the Branch Type compatibility of the implicit BTI behavior for the following instructions at EL0:

- PACIASP.
- PACIBSP.
- If FEAT_PAuth_LR is implemented, PACIASPPC.
- If FEAT_PAuth_LR is implemented, PACIBSPPC.

BT0	Meaning
0b0	When the PE is executing at EL0, when the specified instructions have an implicit BTI behavior, they are compatible with the same BTYPE values as BTI.jc.
0b1	When the PE is executing at EL0, when the specified instructions have an implicit BTI behavior, they are compatible with the same BTYPE values as BTI.c.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnFPM, bit [34]

When FEAT_FPMR is implemented and ELIsInHost(EL0):

Enables direct and indirect accesses to FPMR from EL0.

When accesses to FPMR are disabled by this control:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- Direct accesses to FPMR from EL0 are trapped to EL2 and reported using EC syndrome value `0x18`.
- Execution of FP8 data-processing instructions that indirectly access FPMR is UNDEFINED at EL0.

EnFPM	Meaning
`0b0`	Direct and indirect accesses to FPMR are disabled at EL0.
`0b1`	This control does not cause any instructions to be trapped.

Traps are not taken if there is a higher priority exception generated by the access.

If EL2 is not implemented or is disabled in the current Security state, the Effective value of this field is `0b1`.

When HCR_EL2.TGE is 0, this field is IGNORED for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MSCEn, bit [33]

When FEAT_MOPS is implemented and ELIsInHost(EL2):

Memory Copy and Memory Set instructions Enable. Enables execution of the Memory Copy and Memory Set instructions at EL0.

MSCEn	Meaning
`0b0`	Execution of the Memory Copy and Memory Set instructions is UNDEFINED at EL0.
`0b1`	This control does not cause any instructions to be UNDEFINED.

If HCR_EL2.TGE == `0b0`, the field is IGNORED for all purposes other than direct reads and writes of the register.

When FEAT_MOPS is implemented and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, the Effective value of this bit is `0b1`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CMOW, bit [32]

When FEAT_CMOW is implemented and ELIsInHost(EL2):

Controls the required permissions for the following cache maintenance instructions executed at EL0:

- Any DC instruction that operates by VA and performs a clean and invalidate operation.
- Any IC instruction that operates by VA.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CMOW	Meaning
0b0	This control does not cause any instruction to generate a stage 1 Permission fault.
0b1	For these instructions, when executed at EL0, the absence of stage 1 unprivileged write permission generates a stage 1 permission fault.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

For more information, see:

- Stage 1 permissions.
- Implications of enabling the dirty state management mechanism.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnIA, bit [31]

When FEAT_PAuth is implemented:

Controls enabling of pointer authentication of instruction addresses, using the APIAKey_EL1 key, at EL2 and at EL0 in the EL2&0 translation regime.

EnIA	Meaning
0b0	Pointer authentication of instruction addresses at EL2 and at EL0 in the EL2&0 translation regime, using the APIAKey_EL1 key, is not enabled.
0b1	Pointer authentication of instruction addresses at EL2 and at EL0 in the EL2&0 translation regime, using the APIAKey_EL1 key, is enabled.

Note

This field controls the behavior of the AddPACIA and AuthIA pseudocode functions. When pointer authentication is enabled, AddPACIA returns a copy of a pointer to which a pointer authentication code has been added, and AuthIA returns an authenticated copy of a pointer. When pointer authentication is disabled, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnIB, bit [30]

When FEAT_PAuth is implemented:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Controls enabling of pointer authentication of instruction addresses, using the APIBKey_EL1 key, at EL2 and at EL0 in the EL2&0 translation regime.

EnIB	Meaning
0b0	Pointer authentication of instruction addresses at EL2 and at EL0 in the EL2&0 translation regime, using the APIBKey_EL1 key, is not enabled.
0b1	Pointer authentication of instruction addresses at EL2 and at EL0 in the EL2&0 translation regime, using the APIBKey_EL1 key, is enabled.

Note

This field controls the behavior of the AddPACIB and AuthIB pseudocode functions. When pointer authentication is enabled, AddPACIB returns a copy of a pointer to which a pointer authentication code has been added, and AuthIB returns an authenticated copy of a pointer. When pointer authentication is disabled, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

LSMAOE, bit [29]

When FEAT_LSMAOC is implemented and ELIsInHost(EL2):

Load Multiple and Store Multiple Atomicity and Ordering Enable.

LSMAOE	Meaning
0b0	For all memory accesses at EL0, T32 and A32 Load Multiple and Store Multiple can have an interrupt taken during the sequence memory accesses, and the memory accesses are not required to be ordered.
0b1	The ordering and interrupt behavior of T32 and A32 Load Multiple and Store Multiple at EL0 is as defined for Armv8.0.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

nTLSMD, bit [28]

When FEAT_LSMAOC is implemented and ELIsInHost(EL2):

No Trap Load Multiple and Store Multiple to Device-nGRE/Device-nGnRE/Device-nGnRnE memory.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

nTLSMD	Meaning
0b0	All memory accesses by T32 and A32 Load Multiple and Store Multiple at EL0 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are trapped and generate a stage 1 Alignment fault.
0b1	All memory accesses by T32 and A32 Load Multiple and Store Multiple at EL0 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are not trapped.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

EnDA, bit [27]

When FEAT_PAuth is implemented:

Controls enabling of pointer authentication of data addresses, using the APDAKey_EL1 key, at EL2 and at EL0 in the EL2&0 translation regime.

EnDA	Meaning
0b0	Pointer authentication of data addresses at EL2 and at EL0 in the EL2&0 translation regime, using the APDAKey_EL1 key, is not enabled.
0b1	Pointer authentication of data addresses at EL2 and at EL0 in the EL2&0 translation regime, using the APDAKey_EL1 key, is enabled.

Note

This field controls the behavior of the AddPACDA and AuthDA pseudocode functions. When pointer authentication is enabled, AddPACDA returns a copy of a pointer to which a pointer authentication code has been added, and AuthDA returns an authenticated copy of a pointer. When pointer authentication is disabled, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UCI, bit [26]

When ELIsInHost(EL2):

Traps execution of cache maintenance instructions at EL0 to EL2, from AArch64 state only, reported using EC syndrome value 0x18, as follows:

- DC CVAU, DC CIVAC, DC CVAC, and IC IVAU.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- If FEAT_MTE is implemented, DC CIGVAC, DC CIGDVAC, DC CGVAC, and DC CGDVAC.
- If FEAT_DPB is implemented, DC CVAP.
- If FEAT_DPB and FEAT_MTE are implemented, DC CGVAP and DC CGDVAP.
- If FEAT_DPB2 is implemented, DC CVADP.
- If FEAT_DPB2 and FEAT_MTE are implemented, DC CGVADP and DC CGDVADP.
- If FEAT_OCCMO is implemented, DC CIVAOC, DC CIGDVAOC, DC CVAOC and DC CGDVAOC.

UCI	Meaning
`0b0`	For each of the specified instructions, if the execution of the instruction can be trapped, access at EL0 using AArch64 is trapped to EL2.
`0b1`	This control does not cause any instructions to be trapped.

If HCR_EL2.TGE == `0b0`, the field is IGNORED for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EE, bit [25]

When FEAT_MixedEnd is implemented:

Endianness of data accesses at EL2, stage 1 translation table walks in the EL2 or EL2&0 translation regime, and stage 2 translation table walks in the EL1&0 translation regime.

EE	Meaning
`0b0`	Explicit data accesses at EL2, stage 1 translation table walks in the EL2 or EL2&0 translation regime, and stage 2 translation table walks in the EL1&0 translation regime are little-endian.
`0b1`	Explicit data accesses at EL2, stage 1 translation table walks in the EL2 or EL2&0 translation regime, and stage 2 translation table walks in the EL1&0 translation regime are big-endian.

The EE bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

When FEAT_BigEnd is implemented:

Explicit data accesses at EL1, and stage 1 translation table walks in the EL1&0 translation regime are big-endian.

Reserved, RES1.

Otherwise:

Explicit data accesses at EL1, and stage 1 translation table walks in the EL1&0 translation regime are little-endian.

Reserved, RES0.

E0E, bit [24]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When ELIsInHost(EL2) and FEAT_MixedEndEL0 is implemented:

Endianness of data accesses at EL0.

E0E	Meaning
0b0	Explicit data accesses at EL0 are little-endian.
0b1	Explicit data accesses at EL0 are big-endian.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

This bit has no effect on the endianness of Explicit Memory Effects generated by unprivileged memory access instructions executed at EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_BigEndEL0 is implemented:

Explicit data accesses at EL0 are big-endian.

Reserved, RES1.

Otherwise:

Explicit data accesses at EL0 are little-endian.

Reserved, RES0.

SPAN, bit [23]

When ELIsInHost(EL2):

Set Privileged Access Never, on taking an exception to EL2.

SPAN	Meaning
0b0	PSTATE.PAN is set to 1 on taking an exception to EL2.
0b1	The value of PSTATE.PAN is left unchanged on taking an exception to EL2.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

EIS, bit [22]

When FEAT_ExS is implemented:

Exception entry is a context synchronization event.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EIS	Meaning
0b0	The taking of an exception to EL2 is not a context synchronization event.
0b1	The taking of an exception to EL2 is a context synchronization event.

If SCTLR_EL2.EIS is set to 0b0:

- Indirect writes to ESR_EL2, FAR_EL2, SPSR_EL2, ELR_EL2, and HPFAR_EL2 are synchronized on exception entry to EL2, so that a direct read of the register after exception entry sees the indirectly written value caused by the exception entry.
- Memory transactions, including instruction fetches, from an Exception level always use the translation resources associated with that translation regime.
- Exception Catch debug events are synchronous debug events.
- DCPS* and DRPS instructions are context synchronization events.
- Some exception entries reported are not considered IFBEs per the memory model.

The following are not affected by the value of SCTLR_EL2.EIS:

- Changes to the PSTATE information on entry to EL2.
- Behavior of accessing the banked copies of the stack pointer using the SP register name for loads, stores, and data processing instructions.
- The memory model requirement for exception entry to generate an Instruction Fetch Barrier Effect for some exception entries. See Basic definitions for the list of exception entries.
- Exit from Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

IESB, bit [21]

When FEAT_IESB is implemented:

Implicit Error Synchronization event enable.

IESB	Meaning
0b0	Disabled.
0b1	An implicit error synchronization event is added: • At each exception taken to EL2. • Before the operational pseudocode of each ERET instruction executed at EL2.

If FEAT_DoubleFault2 is implemented, the PE is in Non-debug state, and the Effective value of SCTLR2_EL2.NMEA is 1, then SCTLR_EL2.IESB is ignored and the PE behaves as if SCTLR_EL2.IESB is 1 for all purposes other than direct read of the register.

When the PE is in Debug state, the effect of this field is CONSTRAINED UNPREDICTABLE, and its Effective value might be 0 or 1 regardless of the value of the field. If the Effective value of the field is 1, then an implicit error synchronization event is added after each DCPSx instruction taken to EL2 and before each DRPS instruction executed at EL2, in addition to the other cases where it is added.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TSCXT, bit [20]

When (FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented) and ELIsInHost(EL2):

Trap EL0 Access to the SCXTNUM_EL0 register, when EL0 is using AArch64.

TSCXT	Meaning
`0b0`	EL0 access to SCXTNUM_EL0 is not disabled by this mechanism.
`0b1`	EL0 access to SCXTNUM_EL0 is disabled, causing an exception to EL2, and the SCXTNUM_EL0 value is treated as 0.

If HCR_EL2.TGE == `0b0`, the field is IGNORED for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_CSV2_2 is not implemented, FEAT_CSV2_1p2 is not implemented, and ELIsInHost(EL0):

Reserved, RES1.

Otherwise:

Reserved, RES0.

WXN, bit [19]

Write permission implies XN (Execute-never). For the EL2 or EL2&0 translation regime, this bit can restrict execute permissions on writeable pages.

WXN	Meaning
`0b0`	This control has no effect on memory access permissions.
`0b1`	In the EL2 or EL2&0 translation regime, any region of memory that is writeable at EL2 is XN at EL2. In the EL2&0 translation regime, any region of memory that is writeable at EL0 is XN at EL0.

This bit applies only when SCTLR_EL2.M bit is set.

The WXN bit is permitted to be cached in a TLB.

This field is RES0 if TCR2_EL2.PIE is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

nTWE, bit [18]

When ELIsInHost(EL2):

Traps execution of WFE instructions at EL0 to EL2, from both Execution states.

When FEAT_WFxT is implemented, this trap also applies to the WFET instruction.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

nTWE	Meaning
0b0	Any attempt to execute a WFE instruction at EL0 is trapped to EL2, if the instruction would otherwise have caused the PE to enter a low-power state.
0b1	This control does not cause any instructions to be trapped.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

In AArch32 state, the attempted execution of a conditional WFE instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

Bit [17]

Reserved, RES0.

nTWI, bit [16]

When ELIsInHost(EL2):

Traps execution of WFI instructions at EL0 to EL2, from both Execution states.

When FEAT_WFxT is implemented, this trap also applies to the WFIT instruction.

nTWI	Meaning
0b0	Any attempt to execute a WFI instruction at EL0 is trapped EL2, if the instruction would otherwise have caused the PE to enter a low-power state.
0b1	This control does not cause any instructions to be trapped.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

In AArch32 state, the attempted execution of a conditional WFI instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES1.

UCT, bit [15]

When ELIsInHost(EL2):

Traps EL0 accesses to the CTR_EL0 to EL2, from AArch64 state only.

UCT	Meaning
0b0	Accesses to the CTR_EL0 from EL0 using AArch64 are trapped to EL2.
0b1	This control does not cause any instructions to be trapped.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DZE, bit [14]

When ELIsInHost(EL2):

Traps execution of DC ZVA instructions at EL0 to EL2, from AArch64 state only.

If FEAT_MTE is implemented, this trap also applies to DC GVA and DC GZVA.

DZE	Meaning
0b0	Any attempt to execute an instruction that this trap applies to at EL0 using AArch64 is trapped to EL2. Reading DCZID_EL0.DZP from EL0 returns 1, indicating that the instructions that this trap applies to are not supported.
0b1	This control does not cause any instructions to be trapped.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnDB, bit [13]

When FEAT_PAuth is implemented:

Controls enabling of pointer authentication of data addresses, using the APDBKey_EL1 key, at EL2 and at EL0 in the EL2&0 translation regime.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EnDB	Meaning
0b0	Pointer authentication of data addresses at EL2 and at EL0 in the EL2&0 translation regime, using the APDBKey_EL1 key, is not enabled.
0b1	Pointer authentication of data addresses at EL2 and at EL0 in the EL2&0 translation regime, using the APDBKey_EL1 key, is enabled.

Note

This field controls the behavior of the AddPACDB and AuthDB pseudocode functions. When pointer authentication is enabled, AddPACDB returns a copy of a pointer to which a pointer authentication code has been added, and AuthDB returns an authenticated copy of a pointer. When pointer authentication is disabled, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

I, bit [12]

Instruction access Cacheability control, for accesses at EL2 and, when the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, EL0.

I	Meaning
0b0	All instruction accesses to Normal memory from EL2 are Non-cacheable for all levels of instruction and unified cache. When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, all instruction accesses to Normal memory from EL0 are Non-cacheable for all levels of instruction and unified cache. If SCTLR_EL2.M is 0, instruction accesses from stage 1 of the EL2 or EL2&0 translation regime are to Normal, Outer Shareable, Inner Non-cacheable, Outer Non-cacheable memory.
0b1	This control has no effect on the Cacheability of instruction access to Normal memory from EL2 and, when the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, instruction access to Normal memory from EL0. If the value of SCTLR_EL2.M is 0, instruction accesses from stage 1 of the EL2 or EL2&0 translation regime are to Normal, Outer Shareable, Inner Write-Through, Outer Write-Through memory.

This bit has no effect on the EL3 translation regime.

When EL2 is disabled in the current Security state or the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, this bit has no effect on the EL1&0 translation regime.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

EOS, bit [11]

When FEAT_ExS is implemented:

Exception exit is a context synchronization event.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EOS	Meaning
0b0	An exception return from EL2 is not a context synchronization event.
0b1	An exception return from EL2 is a context synchronization event.

If SCTLR_EL2.EOS is set to 0b0:

- Memory transactions, including instruction fetches, from an Exception level always use the translation resources associated with that translation regime.
- Exception Catch debug events are synchronous debug events.
- DCPS* and DRPS instructions are context synchronization events.

The following are not affected by the value of SCTLR_EL2.EOS:

- The indirect write of the PSTATE and PC values from SPSR_EL2 and ELR_EL2 on exception return is synchronized.
- Behavior of accessing the banked copies of the stack pointer using the SP register name for loads, stores, and data processing instructions.
- Exit from Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

EnRCTX, bit [10]

When FEAT_SPECRES is implemented and ELIsInHost(EL2):

Enable EL0 access to the following System instructions:

- CFPRCTX, DVPRCTX and CPPRCTX instructions.
- If FEAT_SPECRES2 is implemented, COSPRCTX.
- CFP RCTX, DVP RCTX and CPP RCTX instructions.
- If FEAT_SPECRES2 is implemented, COSP RCTX.

EnRCTX	Meaning
0b0	EL0 access to these instructions is disabled, and these instructions are trapped to EL2.
0b1	EL0 access to these instructions is enabled.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [9]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SED, bit [8]

When FEAT_AA32EL0 is implemented and ELIsInHost(EL2):

SETEND instruction disable. Disables SETEND instructions at EL0 using AArch32.

SED	Meaning
`0b0`	SETEND instruction execution is enabled at EL0 using AArch32.
`0b1`	SETEND instructions are UNDEFINED at EL0 using AArch32.

If HCR_EL2.TGE == `0b0`, the field is IGNORED for all purposes other than direct reads and writes of the register.

If the implementation does not support mixed-endian operation at any Exception level, this bit is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_AA32EL0 is not implemented and ELIsInHost(EL2):

If HCR_EL2.TGE == `0b0`, the field is IGNORED for all purposes other than direct reads and writes of the register.

Access to this field is RES1.

Otherwise:

Reserved, RES0.

ITD, bit [7]

When FEAT_AA32EL0 is implemented and ELIsInHost(EL2):

IT Disable. Disables some uses of IT instructions at EL0 using AArch32.

ITD	Meaning
`0b0`	All IT instruction functionality is enabled at EL0 using AArch32.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ITD	Meaning

`0b1`	Any attempt at EL0 using AArch32 to execute any of the following is UNDEFINED:
	• All encodings of the IT instruction with hw1[3:0]!=1000.
	• All encodings of the subsequent instruction with the following values for hw1:
	  • `0b11xxxxxxxxxxxxxx`: All 32-bit instructions, and the 16-bit instructions B, UDF, SVC, LDM, and STM.
	  • `0b1011xxxxxxxxxxxx`: All instructions in 'Miscellaneous 16-bit instructions'.
	  • `0b10100xxxxxxxxxxx`: ADD Rd, PC, #imm
	  • `0b01001xxxxxxxxxxx`: LDR Rd, [PC, #imm]
	  • `0b0100x1xxx1111xxx`: ADD Rdn, PC; CMP Rn, PC; MOV Rd, PC; BX PC; BLX PC.
	  • `0b010001xx1xxxx111`: ADD PC, Rm; CMP PC, Rm; MOV PC, Rm. This pattern also covers UNPREDICTABLE cases with BLX Rn.
	These instructions are always UNDEFINED, regardless of whether they would pass or fail the condition code check that applies to them as a result of being in an IT block.
	It is IMPLEMENTATION DEFINED whether the IT instruction is treated as:
	• A 16-bit instruction, that can only be followed by another 16-bit instruction.
	• The first half of a 32-bit instruction.
	This means that, for the situations that are UNDEFINED, either the second 16-bit instruction or the 32-bit instruction is UNDEFINED.
	An implementation might vary dynamically whether IT is treated as a 16-bit instruction or the first half of a 32-bit instruction.

If HCR_EL2.TGE == `0b0`, the field is IGNORED for all purposes other than direct reads and writes of the register.

If an instruction in an active IT block that would be disabled by this field sets this field to 1, then behavior is CONSTRAINED UNPREDICTABLE. For more information see 'Changes to an ITD control by an instruction in an IT block'.

ITD is optional, but if it is implemented in the SCTLR_EL2, then it must also be implemented in the SCTLR_EL1, HSCTLR, and SCTLR.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement ITD, access to this field is RAZ/WI.

When FEAT_AA32EL0 is not implemented and ELIsInHost(EL2):

Reserved, RES1.

Otherwise:

Reserved, RES0.

nAA, bit [6]

When FEAT_LSE2 is implemented:

Non-aligned access. This bit controls generation of Alignment faults under certain conditions at EL2, and, when the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, EL0.

The following instructions generate an Alignment fault if all bytes being accessed are not within a single naturally aligned 16-byte quantity, for access:

- LDAPR, LDAPRH, LDAPUR, LDAPURH, LDAPURSH, LDAPURSW, LDAR, LDARH, LDLAR, LDLARH.
- STLLR, STLLRH, STLR, STLRH, STLUR, and STLURH.
- If FEAT_LRCPC3 is implemented, the post index versions of LDAPR and the pre index versions of STLR.
- If FEAT_LRCPC3 and Advanced SIMD and floating-point instructions are implemented, LDAPUR (SIMD&FP), LDAP1 (SIMD&FP), STLUR (SIMD&FP), and STL1 (SIMD&FP).

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

If FEAT_LRCPC3 is implemented, the following instructions generate an Alignment fault if all bytes being accessed for a single register are not within a single naturally aligned 16-byte quantity, for access:

- LDIAPP, STILP.

nAA	Meaning
0b0	Unaligned accesses by the specified instructions generate an Alignment fault.
0b1	Unaligned accesses by the specified instructions do not generate an Alignment fault.

For a load-acquire instruction that does not have acquire semantics as the result of the destination register, or registers, being ZR, it is IMPLEMENTATION SPECIFIC whether this field behaves as 1 or the programmed value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CP15BEN, bit [5]

When FEAT_AA32EL0 is implemented and ELIsInHost(EL2):

System instruction memory barrier enable. Enables accesses to the DMB, DSB, and ISB System instructions in the (coproc==0b1111) encoding space from EL0:

CP15BEN	Meaning
0b0	EL0 using AArch32: EL0 execution of the CP15DMB, CP15DSB, and CP15ISB instructions is UNDEFINED.
0b1	EL0 using AArch32: EL0 execution of the CP15DMB, CP15DSB, and CP15ISB instructions is enabled.

If HCR_EL2.TGE == 0b0, the field is IGNORED for all purposes other than direct reads and writes of the register.

CP15BEN is optional, but if it is implemented in the SCTLR_EL2, then it must also be implemented in the SCTLR_EL1, HSCTLR, and SCTLR.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_AA32EL0 is not implemented and ELIsInHost(EL2):

Access to this field is RES0.

Otherwise:

Reserved, RES1.

SA0, bit [4]

When ELIsInHost(EL2):

SP Alignment check enable for EL0. When set to 1, if a load or store instruction executed at EL0 uses the SP as the base address and the SP is not aligned to a 16-byte boundary, then an SP alignment fault exception is generated. For more information, see 'SP alignment checking'.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

If HCR_EL2.TGE == `0b0`, the field is IGNORED for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

SA, bit [3]

SP Alignment check enable. When set to 1, if a load or store instruction executed at EL2 uses the SP as the base address and the SP is not aligned to a 16-byte boundary, then an SP alignment fault exception is generated. For more information, see 'SP alignment checking'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [2]

Data access Cacheability control, for accesses at EL2 and, when the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, EL0

C	Meaning

`0b0`	The following are Non-cacheable for all levels of data and unified cache:
	• Data accesses to Normal memory from EL2.
	• When the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, Normal memory accesses to the EL2 translation tables.
	• When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}:
	  • Data accesses to Normal memory from EL0.
	  • Normal memory accesses to the EL2&0 translation tables.

`0b1`	This control has no effect on the Cacheability of:
	• Data access to Normal memory from EL2.
	• When the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, Normal memory accesses to the EL2 translation tables.
	• When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}:
	  • Data accesses to Normal memory from EL0.
	  • Normal memory accesses to the EL2&0 translation tables.

This bit has no effect on the EL3 translation regime.

When EL2 is disabled in the current Security state or the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, this bit has no effect on the EL1&0 translation regime.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

A, bit [1]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Alignment check enable. This is the enable bit for Alignment fault checking at EL2 and, when the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, EL0.

A	Meaning
0b0	Alignment fault checking is disabled when executing at EL2. When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, alignment fault checking disabled when executing at EL0. Alignment checks on some instructions are not disabled by this control. For more information, see 'Alignment of data accesses'.
0b1	Alignment fault checking is enabled when executing at EL2. When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, alignment fault checking enabled when executing at EL0. All instructions that load or store one or more registers have an alignment check that the address being accessed is aligned to the size of the data element(s) being accessed. If this check fails it causes an Alignment fault, which is taken as a Data Abort exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M, bit [0]

MMU enable for EL2 or EL2&0 stage 1 address translation.

M	Meaning
0b0	When the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, EL2 stage 1 address translation disabled. When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, EL2&0 stage 1 address translation disabled. See the SCTLR_EL2.I field for the behavior of instruction accesses to Normal memory.
0b1	When the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, EL2 stage 1 address translation enabled. When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, EL2&0 stage 1 address translation enabled.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Accessing SCTLR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name SCTLR_EL2 or SCTLR_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

If FEAT_SRMASK is implemented, accesses to SCTLR_EL2 are masked by SCTLRMASK_EL2.

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

MRS <Xt>, SCTLR_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = SCTLR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCTLR_EL2;
```

MSR SCTLR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if IsFeatureImplemented(FEAT_SRMASK) then
        SCTLR_EL2 = (X[t, 64] AND NOT EffectiveSCTLRMASK_EL2()) OR (SCTLR_EL2 AND
            ↪EffectiveSCTLRMASK_EL2());
    else
        SCTLR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCTLR_EL2 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, SCTLR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0000	0b000

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
         ↪&& HFGRTR_EL2.SCTLR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x110];
    else
        X[t, 64] = SCTLR_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = SCTLR_EL2;
    else
        X[t, 64] = SCTLR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCTLR_EL1;
```

When FEAT_VHE is implemented MSR SCTLR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
         ↪&& HFGWTR_EL2.SCTLR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x110] = X[t, 64];
    else
        if IsFeatureImplemented(FEAT_SRMASK) then
            SCTLR_EL1 = (X[t, 64] AND NOT EffectiveSCTLRMASK_EL1()) OR (SCTLR_EL1 AND
                 ↪EffectiveSCTLRMASK_EL1());
        else
            SCTLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_SRMASK) then
            SCTLR_EL2 = (X[t, 64] AND NOT EffectiveSCTLRMASK_EL2()) OR (SCTLR_EL2 AND
                 ↪EffectiveSCTLRMASK_EL2());
        else
            SCTLR_EL2 = X[t, 64];
    else
        SCTLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCTLR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.177 SCTLR_EL3, System Control Register (EL3)

The SCTLR_EL3 characteristics are:

Purpose

Provides top-level control of the system, including its memory system, at EL3.

Configuration

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SCTLR_EL3 are UNDEFINED.

Attributes

SCTLR_EL3 is a 64-bit register.

Field descriptions

Bit [63]

Reserved, RES0.

SPINTMASK, bit [62]

When FEAT_NMI is implemented:

SP Interrupt Mask enable. When SCTLR_EL3.NMI is 1, controls whether PSTATE.SP acts as an interrupt mask, and controls the value of PSTATE.ALLINT on taking an exception to EL3.

SPINTMASK	Meaning
0b0	Does not cause PSTATE.SP to mask interrupts. PSTATE.ALLINT is set to 1 on taking an exception to EL3.
0b1	When PSTATE.SP is 1 and execution is at EL3, an IRQ or FIQ interrupt that is targeted to EL3 is masked regardless of any indication of Superpriority. PSTATE.ALLINT is set to 0 on taking an exception to EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

NMI, bit [61]

When FEAT_NMI is implemented:

Non-maskable Interrupt enable.

NMI	Meaning
0b0	This control does not affect interrupt masking behavior.
0b1	This control enables all of the following: • The use of the PSTATE.ALLINT interrupt mask. • IRQ and FIQ interrupts to have Superpriority as an additional attribute. • PSTATE.SP to be used as an interrupt mask.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3, this field resets to '0'.

Otherwise:

Reserved, RES0.

Bit [60]

Reserved, RES0.

TCSO, bit [59]

When FEAT_MTE_STORE_ONLY is implemented:

Tag Checking Store Only.

TCSO	Meaning
0b0	This field has no effect on Tag checking.
0b1	Explicit Memory Read Effects generated by instructions executed in EL3 are Tag Unchecked.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [58:45]

Reserved, RES0.

DSSBS, bit [44]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_SSBS is implemented:

Default PSTATE.SSBS value on Exception Entry.

DSSBS	Meaning
0b0	PSTATE.SSBS is set to 0 on an exception to EL3.
0b1	PSTATE.SSBS is set to 1 on an exception to EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RES0.

ATA, bit [43]

When FEAT_MTE2 is implemented:

Allocation Tag Access in EL3.

Controls use of Memory tagging in EL3.

ATA	Meaning
0b0	Use of Memory tagging is disabled at EL3.
0b1	Use of Memory tagging is enabled at EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [42]

Reserved, RES0.

TCF, bits [41:40]

When FEAT_MTE2 is implemented:

Tag Check Fault in EL3. Controls the effect of Tag Check Faults due to Loads and Stores in EL3.

TCF	Meaning	Applies when
0b00	Tag Check Faults have no effect on the PE.	
0b01	Tag Check Faults cause a synchronous exception.	

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TCF	Meaning	Applies when
0b10	Tag Check Faults are asynchronously accumulated.	
0b11	Tag Check Faults cause a synchronous exception on reads, and are asynchronously accumulated on writes.	FEAT_MTE_ASYM_FAULT is implemented

If FEAT_MTE3 is not implemented, the value 0b11 is reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [39:38]

Reserved, RES0.

ITFSB, bit [37]

When FEAT_MTE_ASYNC is implemented:

When synchronous exceptions are not being generated by Tag Check Faults, this field controls whether on exception entry into EL3, all Tag Check Faults due to instructions executed before exception entry, that are reported asynchronously, are synchronized into TFSRE0_EL1 and TFSR_ELx registers.

ITFSB	Meaning
0b0	Tag Check Faults are not synchronized on entry to EL3.
0b1	Tag Check Faults are synchronized on entry to EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BT, bit [36]

When FEAT_BTI is implemented:

Indicates the Branch Type compatibility of the implicit BTI behavior for the following instructions at EL3:

- PACIASP.
- PACIBSP.
- If FEAT_PAuth_LR is implemented, PACIASPPC.
- If FEAT_PAuth_LR is implemented, PACIBSPPC.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

BT	Meaning
0b0	When the PE is executing at EL3, when the specified instructions have an implicit BTI behavior, they are compatible with the same BTYPE values as BTI.jc.
0b1	When the PE is executing at EL3, when the specified instructions have an implicit BTI behavior, they are compatible with the same BTYPE values as BTI.c.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [35:32]

Reserved, RES0.

EnIA, bit [31]

When FEAT_PAuth is implemented:

Controls enabling of pointer authentication of instruction addresses, using the APIAKey_EL1 key, in the EL3 translation regime.

Possible values of this bit are:

EnIA	Meaning
0b0	Pointer authentication of instruction addresses, using the APIAKey_EL1 key, is not enabled.
0b1	Pointer authentication of instruction addresses, using the APIAKey_EL1 key, is enabled.

Note

This field controls the behavior of the AddPACIA and AuthIA pseudocode functions. Specifically, when the field is 1, AddPACIA returns a copy of a pointer to which a pointer authentication code has been added, and AuthIA returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnIB, bit [30]

When FEAT_PAuth is implemented:

Controls enabling of pointer authentication of instruction addresses, using the APIBKey_EL1 key, in the EL3 translation regime.

Possible values of this bit are:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EnIB	Meaning
0b0	Pointer authentication of instruction addresses, using the APIBKey_EL1 key, is not enabled.
0b1	Pointer authentication of instruction addresses, using the APIBKey_EL1 key, is enabled.

Note

This field controls the behavior of the AddPACIB and AuthIB pseudocode functions. Specifically, when the field is 1, AddPACIB returns a copy of a pointer to which a pointer authentication code has been added, and AuthIB returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [29:28]

Reserved, RES1.

EnDA, bit [27]

When FEAT_PAuth is implemented:

Controls enabling of pointer authentication of instruction addresses, using the APDAKey_EL1 key, in the EL3 translation regime.

EnDA	Meaning
0b0	Pointer authentication of data addresses, using the APDAKey_EL1 key, is not enabled.
0b1	Pointer authentication of data addresses, using the APDAKey_EL1 key, is enabled.

Note

This field controls the behavior of the AddPACDA and AuthDA pseudocode functions. Specifically, when the field is 1, AddPACDA returns a copy of a pointer to which a pointer authentication code has been added, and AuthDA returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [26]

Reserved, RES0.

EE, bit [25]

When FEAT_MixedEnd is implemented:

Endianness of data accesses at EL3, and stage 1 translation table walks in the EL3 translation regime.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EE	Meaning
0b0	Explicit data accesses at EL3, and stage 1 translation table walks in the EL3 translation regime are little-endian.
0b1	Explicit data accesses at EL3, and stage 1 translation table walks in the EL3 translation regime are big-endian.

The EE bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

When FEAT_BigEnd is implemented:

Explicit data accesses at EL1, and stage 1 translation table walks in the EL1&0 translation regime are big-endian.

Reserved, RES1.

Otherwise:

Explicit data accesses at EL1, and stage 1 translation table walks in the EL1&0 translation regime are little-endian.

Reserved, RES0.

Bit [24]

Reserved, RES0.

Bit [23]

Reserved, RES1.

EIS, bit [22]

When FEAT_ExS is implemented:

Exception Entry is a context synchronization event.

EIS	Meaning
0b0	The taking of an exception to EL3 is not a context synchronization event.
0b1	The taking of an exception to EL3 is a context synchronization event.

If SCTLR_EL3.EIS is set to 0b0:

- Indirect writes to ESR_EL3, FAR_EL3, SPSR_EL3, ELR_EL3 are synchronized on exception entry to EL3, so that a direct read of the register after exception entry sees the indirectly written value caused by the exception entry.
- Memory transactions, including instruction fetches, from an Exception level always use the translation resources associated with that translation regime.
- Exception Catch debug events are synchronous debug events.
- DCPS* and DRPS instructions are context synchronization events.
- Some exception entries reported are not considered IFBEs per the memory model.

The following are not affected by the value of SCTLR_EL3.EIS:

- Changes to the PSTATE information on entry to EL3.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- Behavior of accessing the banked copies of the stack pointer using the SP register name for loads, stores and data processing instructions.
- The memory model requirement for exception entry to generate an Instruction Fetch Barrier Effect for some exception entries. See Basic definitions for the list of exception entries.
- Exit from Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

IESB, bit [21]

When FEAT_IESB is implemented:

Implicit Error Synchronization event enable.

IESB	Meaning
0b0	Disabled.
0b1	An implicit error synchronization event is added: • At each exception taken to EL3. • Before the operational pseudocode of each ERET instruction executed at EL3.

When the PE is in Debug state, the effect of this field is CONSTRAINED UNPREDICTABLE, and its Effective value might be 0 or 1 regardless of the value of the field and, if implemented, SCR_EL3.NMEA. If the Effective value of the field is 1, then an implicit error synchronization event is added after each DCPSx instruction taken to EL3 and before each DRPS instruction executed at EL3, in addition to the other cases where it is added.

When FEAT_DoubleFault is implemented, the PE is in Non-debug state, and the Effective value of SCR_EL3.NMEA is 1, this field is ignored and its Effective value is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [20]

Reserved, RES0.

WXN, bit [19]

Write permission implies XN (Execute-never). For the EL3 translation regime, this bit can force all memory regions that are writable to be treated as XN.

WXN	Meaning
0b0	This control has no effect on memory access permissions.
0b1	Any region that is writable in the EL3 translation regime is forced to XN for accesses from software executing at EL3.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

This bit applies only when SCTLR_EL3.M bit is set.

The WXN bit is permitted to be cached in a TLB.

This field is RES0 if TCR_EL3.PIE is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [18]

Reserved, RES1.

Bit [17]

Reserved, RES0.

Bit [16]

Reserved, RES1.

Bits [15:14]

Reserved, RES0.

EnDB, bit [13]

When FEAT_PAuth is implemented:

Controls enabling of pointer authentication of instruction addresses, using the APDBKey_EL1 key, in the EL3 translation regime.

EnDB	Meaning
0b0	Pointer authentication of data addresses, using the APDBKey_EL1 key, is not enabled.
0b1	Pointer authentication of data addresses, using the APDBKey_EL1 key, is enabled.

Note

This field controls the behavior of the AddPACDB and AuthDB pseudocode functions. Specifically, when the field is 1, AddPACDB returns a copy of a pointer to which a pointer authentication code has been added, and AuthDB returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

I, bit [12]

Instruction access Cacheability control, for accesses at EL3:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

I	Meaning
0b0	All instruction access to Normal memory from EL3 are Non-cacheable for all levels of instruction and unified cache. If the value of SCTLR_EL3.M is 0, instruction accesses from stage 1 of the EL3 translation regime are to Normal, Outer Shareable, Inner Non-cacheable, Outer Non-cacheable memory.
0b1	This control has no effect on the Cacheability of instruction access to Normal memory from EL3. If the value of SCTLR_EL3.M is 0, instruction accesses from stage 1 of the EL3 translation regime are to Normal, Outer Shareable, Inner Write-Through, Outer Write-Through memory.

This bit has no effect on the EL1&0, EL2, or EL2&0 translation regimes.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3, this field resets to `'0'`.

EOS, bit [11]

When FEAT_ExS is implemented:

Exception Exit is a context synchronization event.

EOS	Meaning
0b0	An exception return from EL3 is not a context synchronization event
0b1	An exception return from EL3 is a context synchronization event

If SCTLR_EL3.EOS is set to `0b0`:

- Memory transactions, including instruction fetches, from an Exception level always use the translation resources associated with that translation regime.
- Exception Catch debug events are synchronous debug events.
- DCPS* and DRPS instructions are context synchronization events.

The following are not affected by the value of SCTLR_EL3.EOS:

- The indirect write of the PSTATE and PC values from SPSR_EL3 and ELR_EL3 on exception return is synchronized.
- If the PE enters Debug state before the first instruction after an Exception return from EL3 to Non-secure state, any pending Halting debug event completes execution.
- The GIC behavior that allocates interrupts to FIQ or IRQ changes simultaneously with leaving the EL3 Exception level.
- Behavior of accessing the banked copies of the stack pointer using the SP register name for loads, stores and data processing instructions.
- Exit from Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

Bits [10:7]

Reserved, RES0.

nAA, bit [6]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_LSE2 is implemented:

Non-aligned access. This bit controls generation of Alignment faults at EL3 under certain conditions.

The following instructions generate an Alignment fault if all bytes being accessed are not within a single naturally aligned 16-byte quantity, for access:

- LDAPR, LDAPRH, LDAPUR, LDAPURH, LDAPURSH, LDAPURSW, LDAR, LDARH, LDLAR, LDLARH.
- STLLR, STLLRH, STLR, STLRH, STLUR, and STLURH.
- If FEAT_LRCPC3 is implemented, the post index versions of LDAPR and the pre index versions of STLR.
- If FEAT_LRCPC3 and Advanced SIMD and floating-point instructions are implemented, LDAPUR (SIMD&FP), LDAP1 (SIMD&FP), STLUR (SIMD&FP), and STL1 (SIMD&FP).

If FEAT_LRCPC3 is implemented, the following instructions generate an Alignment fault if all bytes being accessed for a single register are not within a single naturally aligned 16-byte quantity, for access:

- LDIAPP, STILP.

nAA	Meaning
`0b0`	Unaligned accesses by the specified instructions generate an Alignment fault.
`0b1`	Unaligned accesses by the specified instructions do not generate an Alignment fault.

For a load-acquire instruction that does not have acquire semantics as the result of the destination register, or registers, being ZR, it is IMPLEMENTATION SPECIFIC whether this field behaves as 1 or the programmed value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [5:4]

Reserved, RES1.

SA, bit [3]

SP Alignment check enable. When set to 1, if a load or store instruction executed at EL3 uses the SP as the base address and the SP is not aligned to a 16-byte boundary, then a SP alignment fault exception is generated. For more information, see ‘SP alignment checking’.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [2]

Cacheability control, for data accesses.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

C	Meaning
0b0	All data access to Normal memory from EL3, and all Normal memory accesses to the EL3 translation tables, are Non-cacheable for all levels of data and unified cache.
0b1	This control has no effect on the Cacheability of: • Data access to Normal memory from EL3. • Normal memory accesses to the EL3 translation tables.

This bit has no effect on the EL1&0, EL2, or EL2&0 translation regimes.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3, this field resets to '0'.

A, bit [1]

Alignment check enable. This is the enable bit for Alignment fault checking at EL3.

A	Meaning
0b0	Alignment fault checking is disabled when executing at EL3. Alignment checks on some instructions are not disabled by this control. For more information, see 'Alignment of data accesses'.
0b1	Alignment fault checking is enabled when executing at EL3. All instructions that load or store one or more registers have an alignment check that the address being accessed is aligned to the size of the data element(s) being accessed. If this check fails it causes an Alignment fault, which is taken as a Data Abort exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M, bit [0]

MMU enable for EL3 stage 1 address translation. Possible values of this bit are:

M	Meaning
0b0	EL3 stage 1 address translation disabled. See the SCTLR_EL3.I field for the behavior of instruction accesses to Normal memory.
0b1	EL3 stage 1 address translation enabled.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3, this field resets to '0'.

Accessing SCTLR_EL3

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, SCTLR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0000	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCTLR_EL3;
```

```
MSR SCTLR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0000	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.SCTLR_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SCTLR_EL3 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.178 SCTLRMASK_EL1, System Control Masking Register (EL1)

The SCTLRMASK_EL1 characteristics are:

Purpose

Mask register to prevent updates of fields in SCTLR_EL1 on writes to SCTLR_EL1 or SCTLRALIAS_EL1.

Configuration

This register is present only when FEAT_SRMASK is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SCTLRMASK_EL1 are UNDEFINED.

Attributes

SCTLRMASK_EL1 is a 64-bit register.

Field descriptions

TIDCP, bit [63]

When FEAT_TIDCP1 is implemented:

Mask bit for TIDCP.

TIDCP	Meaning
0b0	SCTLR_EL1.TIDCP is writeable.
0b1	SCTLR_EL1.TIDCP is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

SPINTMASK, bit [62]

When FEAT_NMI is implemented:

Mask bit for SPINTMASK.

SPINTMASK	Meaning
0b0	SCTLR_EL1.SPINTMASK is writeable.
0b1	SCTLR_EL1.SPINTMASK is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NMI, bit [61]

When FEAT_NMI is implemented:

Mask bit for NMI.

NMI	Meaning
0b0	SCTLR_EL1.NMI is writeable.
0b1	SCTLR_EL1.NMI is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnTP2, bit [60]

When FEAT_SME is implemented:

Mask bit for EnTP2.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EnTP2	Meaning
0b0	SCTLR_EL1.EnTP2 is writeable.
0b1	SCTLR_EL1.EnTP2 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCSO, bit [59]

When FEAT_MTE_STORE_ONLY is implemented:

Mask bit for TCSO.

TCSO	Meaning
0b0	SCTLR_EL1.TCSO is writeable.
0b1	SCTLR_EL1.TCSO is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCSO0, bit [58]

When FEAT_MTE_STORE_ONLY is implemented:

Mask bit for TCSO0.

TCSO0	Meaning
0b0	SCTLR_EL1.TCSO0 is writeable.
0b1	SCTLR_EL1.TCSO0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EPAN, bit [57]

When FEAT_PAN3 is implemented:

Mask bit for EPAN.

EPAN	Meaning
0b0	SCTLR_EL1.EPAN is writeable.
0b1	SCTLR_EL1.EPAN is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnALS, bit [56]

When FEAT_LS64 is implemented:

Mask bit for EnALS.

EnALS	Meaning
0b0	SCTLR_EL1.EnALS is writeable.
0b1	SCTLR_EL1.EnALS is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnAS0, bit [55]

When FEAT_LS64_ACCDATA is implemented:

Mask bit for EnAS0.

EnAS0	Meaning
0b0	SCTLR_EL1.EnAS0 is writeable.
0b1	SCTLR_EL1.EnAS0 is not writeable.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnASR, bit [54]

When FEAT_LS64_V is implemented:

Mask bit for EnASR.

EnASR	Meaning
`0b0`	SCTLR_EL1.EnASR is writeable.
`0b1`	SCTLR_EL1.EnASR is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [53:47]

Reserved, RES0.

TWEDEL, bit [46]

When FEAT_TWED is implemented:

Mask bit for TWEDEL.

TWEDEL	Meaning
`0b0`	SCTLR_EL1.TWEDEL is writeable.
`0b1`	SCTLR_EL1.TWEDEL is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TWEDEn, bit [45]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_TWED is implemented:

Mask bit for TWEDEn.

TWEDEn	Meaning
0b0	SCTLR_EL1.TWEDEn is writeable.
0b1	SCTLR_EL1.TWEDEn is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DSSBS, bit [44]

When FEAT_SSBS is implemented:

Mask bit for DSSBS.

DSSBS	Meaning
0b0	SCTLR_EL1.DSSBS is writeable.
0b1	SCTLR_EL1.DSSBS is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ATA, bit [43]

When FEAT_MTE2 is implemented:

Mask bit for ATA.

ATA	Meaning
0b0	SCTLR_EL1.ATA is writeable.
0b1	SCTLR_EL1.ATA is not writeable.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ATA0, bit [42]

When FEAT_MTE2 is implemented:

Mask bit for ATA0.

ATA0	Meaning
`0b0`	SCTLR_EL1.ATA0 is writeable.
`0b1`	SCTLR_EL1.ATA0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [41]

Reserved, RES0.

TCF, bit [40]

When FEAT_MTE2 is implemented:

Mask bit for TCF.

TCF	Meaning
`0b0`	SCTLR_EL1.TCF is writeable.
`0b1`	SCTLR_EL1.TCF is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [39]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TCF0, bit [38]

When FEAT_MTE2 is implemented:

Mask bit for TCF0.

TCF0	Meaning
0b0	SCTLR_EL1.TCF0 is writeable.
0b1	SCTLR_EL1.TCF0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ITFSB, bit [37]

When FEAT_MTE_ASYNC is implemented:

Mask bit for ITFSB.

ITFSB	Meaning
0b0	SCTLR_EL1.ITFSB is writeable.
0b1	SCTLR_EL1.ITFSB is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BT1, bit [36]

When FEAT_BTI is implemented:

Mask bit for BT1.

BT1	Meaning
0b0	SCTLR_EL1.BT1 is writeable.
0b1	SCTLR_EL1.BT1 is not writeable.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BT0, bit [35]

When FEAT_BTI is implemented:

Mask bit for BT0.

BT0	Meaning
`0b0`	SCTLR_EL1.BT0 is writeable.
`0b1`	SCTLR_EL1.BT0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnFPM, bit [34]

When FEAT_FPMR is implemented:

Mask bit for EnFPM.

EnFPM	Meaning
`0b0`	SCTLR_EL1.EnFPM is writeable.
`0b1`	SCTLR_EL1.EnFPM is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MSCEn, bit [33]

When FEAT_MOPS is implemented:

Mask bit for MSCEn.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

MSCEn	Meaning
0b0	SCTLR_EL1.MSCEn is writeable.
0b1	SCTLR_EL1.MSCEn is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CMOW, bit [32]

When FEAT_CMOW is implemented:

Mask bit for CMOW.

CMOW	Meaning
0b0	SCTLR_EL1.CMOW is writeable.
0b1	SCTLR_EL1.CMOW is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnIA, bit [31]

When FEAT_PAuth is implemented:

Mask bit for EnIA.

EnIA	Meaning
0b0	SCTLR_EL1.EnIA is writeable.
0b1	SCTLR_EL1.EnIA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EnIB, bit [30]

When FEAT_PAuth is implemented:

Mask bit for EnIB.

EnIB	Meaning
0b0	SCTLR_EL1.EnIB is writeable.
0b1	SCTLR_EL1.EnIB is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

LSMAOE, bit [29]

When FEAT_LSMAOC is implemented:

Mask bit for LSMAOE.

LSMAOE	Meaning
0b0	SCTLR_EL1.LSMAOE is writeable.
0b1	SCTLR_EL1.LSMAOE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nTLSMD, bit [28]

When FEAT_LSMAOC is implemented:

Mask bit for nTLSMD.

nTLSMD	Meaning
0b0	SCTLR_EL1.nTLSMD is writeable.
0b1	SCTLR_EL1.nTLSMD is not writeable.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnDA, bit [27]

When FEAT_PAuth is implemented:

Mask bit for EnDA.

EnDA	Meaning
`0b0`	SCTLR_EL1.EnDA is writeable.
`0b1`	SCTLR_EL1.EnDA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UCI, bit [26]

Mask bit for UCI.

UCI	Meaning
`0b0`	SCTLR_EL1.UCI is writeable.
`0b1`	SCTLR_EL1.UCI is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

EE, bit [25]

When FEAT_MixedEnd is implemented:

Mask bit for EE.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EE	Meaning
0b0	SCTLR_EL1.EE is writeable.
0b1	SCTLR_EL1.EE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0E, bit [24]

When FEAT_MixedEndEL0 is implemented:

Mask bit for E0E.

E0E	Meaning
0b0	SCTLR_EL1.E0E is writeable.
0b1	SCTLR_EL1.E0E is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SPAN, bit [23]

When FEAT_PAN is implemented:

Mask bit for SPAN.

SPAN	Meaning
0b0	SCTLR_EL1.SPAN is writeable.
0b1	SCTLR_EL1.SPAN is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EIS, bit [22]

When FEAT_ExS is implemented:

Mask bit for EIS.

EIS	Meaning
0b0	SCTLR_EL1.EIS is writeable.
0b1	SCTLR_EL1.EIS is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IESB, bit [21]

When FEAT_IESB is implemented:

Mask bit for IESB.

IESB	Meaning
0b0	SCTLR_EL1.IESB is writeable.
0b1	SCTLR_EL1.IESB is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TSCXT, bit [20]

When FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented:

Mask bit for TSCXT.

TSCXT	Meaning
0b0	SCTLR_EL1.TSCXT is writeable.
0b1	SCTLR_EL1.TSCXT is not writeable.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

WXN, bit [19]

Mask bit for WXN.

WXN	Meaning
`0b0`	SCTLR_EL1.WXN is writeable.
`0b1`	SCTLR_EL1.WXN is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

nTWE, bit [18]

Mask bit for nTWE.

nTWE	Meaning
`0b0`	SCTLR_EL1.nTWE is writeable.
`0b1`	SCTLR_EL1.nTWE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [17]

Reserved, RES0.

nTWI, bit [16]

Mask bit for nTWI.

nTWI	Meaning
`0b0`	SCTLR_EL1.nTWI is writeable.
`0b1`	SCTLR_EL1.nTWI is not writeable.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

UCT, bit [15]

Mask bit for UCT.

UCT	Meaning
`0b0`	SCTLR_EL1.UCT is writeable.
`0b1`	SCTLR_EL1.UCT is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

DZE, bit [14]

Mask bit for DZE.

DZE	Meaning
`0b0`	SCTLR_EL1.DZE is writeable.
`0b1`	SCTLR_EL1.DZE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

EnDB, bit [13]

When FEAT_PAuth is implemented:

Mask bit for EnDB.

EnDB	Meaning
`0b0`	SCTLR_EL1.EnDB is writeable.
`0b1`	SCTLR_EL1.EnDB is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

I, bit [12]

Mask bit for I.

I	Meaning
0b0	SCTLR_EL1.I is writeable.
0b1	SCTLR_EL1.I is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

EOS, bit [11]

When FEAT_ExS is implemented:

Mask bit for EOS.

EOS	Meaning
0b0	SCTLR_EL1.EOS is writeable.
0b1	SCTLR_EL1.EOS is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnRCTX, bit [10]

When FEAT_SPECRES is implemented:

Mask bit for EnRCTX.

EnRCTX	Meaning
0b0	SCTLR_EL1.EnRCTX is writeable.
0b1	SCTLR_EL1.EnRCTX is not writeable.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UMA, bit [9]

Mask bit for UMA.

UMA	Meaning
`0b0`	SCTLR_EL1.UMA is writeable.
`0b1`	SCTLR_EL1.UMA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

SED, bit [8]

When FEAT_AA32EL0 is implemented:

Mask bit for SED.

SED	Meaning
`0b0`	SCTLR_EL1.SED is writeable.
`0b1`	SCTLR_EL1.SED is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ITD, bit [7]

When FEAT_AA32EL0 is implemented:

Mask bit for ITD.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ITD	Meaning
0b0	SCTLR_EL1.ITD is writeable.
0b1	SCTLR_EL1.ITD is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nAA, bit [6]

When FEAT_LSE2:

Mask bit for nAA.

nAA	Meaning
0b0	SCTLR_EL1.nAA is writeable.
0b1	SCTLR_EL1.nAA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CP15BEN, bit [5]

When FEAT_AA32EL0 is implemented:

Mask bit for CP15BEN.

CP15BEN	Meaning
0b0	SCTLR_EL1.CP15BEN is writeable.
0b1	SCTLR_EL1.CP15BEN is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SA0, bit [4]

Mask bit for SA0.

SA0	Meaning
0b0	SCTLR_EL1.SA0 is writeable.
0b1	SCTLR_EL1.SA0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

SA, bit [3]

Mask bit for SA.

SA	Meaning
0b0	SCTLR_EL1.SA is writeable.
0b1	SCTLR_EL1.SA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

C, bit [2]

Mask bit for C.

C	Meaning
0b0	SCTLR_EL1.C is writeable.
0b1	SCTLR_EL1.C is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

A, bit [1]

Mask bit for A.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

A	Meaning
0b0	SCTLR_EL1.A is writeable.
0b1	SCTLR_EL1.A is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

M, bit [0]

Mask bit for M.

M	Meaning
0b0	SCTLR_EL1.M is writeable.
0b1	SCTLR_EL1.M is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Accessing SCTLRMASK_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name `SCTLRMASK_EL1` or `SCTLRMASK_EL12` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SCTLRMASK_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HFGRTR2_EL2.nSCTLRMASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x318];
    else
        X[t, 64] = SCTLRMASK_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = SCTLRMASK_EL2;
    else
        X[t, 64] = SCTLRMASK_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCTLRMASK_EL1;
```

MSR SCTLRMASK_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HFGWTR2_EL2.nSCTLRMASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x318] = X[t, 64];
    elsif !IsZero(EffectiveSCTLRMASK_EL1()) then
        UNDEFINED;
    else
        SCTLRMASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        if !IsZero(EffectiveSCTLRMASK_EL2()) then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            SCTLRMASK_EL2 = X[t, 64];
    else
        SCTLRMASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCTLRMASK_EL1 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, SCTLRMASK_EL12

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x318];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = SCTLRMASK_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = SCTLRMASK_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented MSR SCTLRMASK_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x318] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            SCTLRMASK_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        SCTLRMASK_EL1 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.179 SCTLRMASK_EL2, System Control Masking Register (EL2)

The SCTLRMASK_EL2 characteristics are:

Purpose

Mask register to prevent updates of fields in SCTLR_EL2 on writes.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

This register is present only when FEAT_SRMASK is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SCTLRMASK_EL2 are UNDEFINED.

Attributes

SCTLRMASK_EL2 is a 64-bit register.

Field descriptions

TIDCP, bit [63]

When FEAT_TIDCP1 is implemented:

Mask bit for TIDCP.

TIDCP	Meaning
0b0	SCTLR_EL2.TIDCP is writeable.
0b1	SCTLR_EL2.TIDCP is not writeable.

The reset behavior of this field is:

- On a Warm reset:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- When the highest implemented Exception level is EL2, this field resets to `'0'`.
- Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SPINTMASK, bit [62]

When FEAT_NMI is implemented:

Mask bit for SPINTMASK.

SPINTMASK	Meaning
`0b0`	SCTLR_EL2.SPINTMASK is writeable.
`0b1`	SCTLR_EL2.SPINTMASK is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NMI, bit [61]

When FEAT_NMI is implemented:

Mask bit for NMI.

NMI	Meaning
`0b0`	SCTLR_EL2.NMI is writeable.
`0b1`	SCTLR_EL2.NMI is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnTP2, bit [60]

When FEAT_SME is implemented:

Mask bit for EnTP2.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EnTP2	Meaning
0b0	SCTLR_EL2.EnTP2 is writeable.
0b1	SCTLR_EL2.EnTP2 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCSO, bit [59]

When FEAT_MTE_STORE_ONLY is implemented:

Mask bit for TCSO.

TCSO	Meaning
0b0	SCTLR_EL2.TCSO is writeable.
0b1	SCTLR_EL2.TCSO is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCSO0, bit [58]

When FEAT_MTE_STORE_ONLY is implemented:

Mask bit for TCSO0.

TCSO0	Meaning
0b0	SCTLR_EL2.TCSO0 is writeable.
0b1	SCTLR_EL2.TCSO0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EPAN, bit [57]

When FEAT_PAN3 is implemented:

Mask bit for EPAN.

EPAN	Meaning
0b0	SCTLR_EL2.EPAN is writeable.
0b1	SCTLR_EL2.EPAN is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnALS, bit [56]

When FEAT_LS64 is implemented:

Mask bit for EnALS.

EnALS	Meaning
0b0	SCTLR_EL2.EnALS is writeable.
0b1	SCTLR_EL2.EnALS is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnAS0, bit [55]

When FEAT_LS64_ACCDATA is implemented:

Mask bit for EnAS0.

EnAS0	Meaning
0b0	SCTLR_EL2.EnAS0 is writeable.
0b1	SCTLR_EL2.EnAS0 is not writeable.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnASR, bit [54]

When FEAT_LS64_V is implemented:

Mask bit for EnASR.

EnASR	Meaning
`0b0`	SCTLR_EL2.EnASR is writeable.
`0b1`	SCTLR_EL2.EnASR is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [53:47]

Reserved, RES0.

TWEDEL, bit [46]

When FEAT_TWED is implemented:

Mask bit for TWEDEL.

TWEDEL	Meaning
`0b0`	SCTLR_EL2.TWEDEL is writeable.
`0b1`	SCTLR_EL2.TWEDEL is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TWEDEn, bit [45]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_TWED is implemented:

Mask bit for TWEDEn.

TWEDEn	Meaning
0b0	SCTLR_EL2.TWEDEn is writeable.
0b1	SCTLR_EL2.TWEDEn is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DSSBS, bit [44]

When FEAT_SSBS is implemented:

Mask bit for DSSBS.

DSSBS	Meaning
0b0	SCTLR_EL2.DSSBS is writeable.
0b1	SCTLR_EL2.DSSBS is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ATA, bit [43]

When FEAT_MTE2 is implemented:

Mask bit for ATA.

ATA	Meaning
0b0	SCTLR_EL2.ATA is writeable.
0b1	SCTLR_EL2.ATA is not writeable.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ATA0, bit [42]

When FEAT_MTE2 is implemented:

Mask bit for ATA0.

ATA0	Meaning
`0b0`	SCTLR_EL2.ATA0 is writeable.
`0b1`	SCTLR_EL2.ATA0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [41]

Reserved, RES0.

TCF, bit [40]

When FEAT_MTE2 is implemented:

Mask bit for TCF.

TCF	Meaning
`0b0`	SCTLR_EL2.TCF is writeable.
`0b1`	SCTLR_EL2.TCF is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [39]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TCF0, bit [38]

When FEAT_MTE2 is implemented:

Mask bit for TCF0.

TCF0	Meaning
0b0	SCTLR_EL2.TCF0 is writeable.
0b1	SCTLR_EL2.TCF0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ITFSB, bit [37]

When FEAT_MTE_ASYNC is implemented:

Mask bit for ITFSB.

ITFSB	Meaning
0b0	SCTLR_EL2.ITFSB is writeable.
0b1	SCTLR_EL2.ITFSB is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BT, bit [36]

When FEAT_BTI is implemented:

Mask bit for BT.

BT	Meaning
0b0	SCTLR_EL2.BT is writeable.
0b1	SCTLR_EL2.BT is not writeable.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BT0, bit [35]

When FEAT_BTI is implemented:

Mask bit for BT0.

BT0	Meaning
`0b0`	SCTLR_EL2.BT0 is writeable.
`0b1`	SCTLR_EL2.BT0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnFPM, bit [34]

When FEAT_FPMR is implemented:

Mask bit for EnFPM.

EnFPM	Meaning
`0b0`	SCTLR_EL2.EnFPM is writeable.
`0b1`	SCTLR_EL2.EnFPM is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MSCEn, bit [33]

When FEAT_MOPS is implemented:

Mask bit for MSCEn.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

MSCEn	Meaning
0b0	SCTLR_EL2.MSCEn is writeable.
0b1	SCTLR_EL2.MSCEn is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CMOW, bit [32]

When FEAT_CMOW is implemented:

Mask bit for CMOW.

CMOW	Meaning
0b0	SCTLR_EL2.CMOW is writeable.
0b1	SCTLR_EL2.CMOW is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnIA, bit [31]

When FEAT_PAuth is implemented:

Mask bit for EnIA.

EnIA	Meaning
0b0	SCTLR_EL2.EnIA is writeable.
0b1	SCTLR_EL2.EnIA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EnIB, bit [30]

When FEAT_PAuth is implemented:

Mask bit for EnIB.

EnIB	Meaning
0b0	SCTLR_EL2.EnIB is writeable.
0b1	SCTLR_EL2.EnIB is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

LSMAOE, bit [29]

When FEAT_LSMAOC is implemented:

Mask bit for LSMAOE.

LSMAOE	Meaning
0b0	SCTLR_EL2.LSMAOE is writeable.
0b1	SCTLR_EL2.LSMAOE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nTLSMD, bit [28]

When FEAT_LSMAOC is implemented:

Mask bit for nTLSMD.

nTLSMD	Meaning
0b0	SCTLR_EL2.nTLSMD is writeable.
0b1	SCTLR_EL2.nTLSMD is not writeable.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnDA, bit [27]

When FEAT_PAuth is implemented:

Mask bit for EnDA.

EnDA	Meaning
`0b0`	SCTLR_EL2.EnDA is writeable.
`0b1`	SCTLR_EL2.EnDA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UCI, bit [26]

Mask bit for UCI.

UCI	Meaning
`0b0`	SCTLR_EL2.UCI is writeable.
`0b1`	SCTLR_EL2.UCI is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

EE, bit [25]

When FEAT_MixedEnd is implemented:

Mask bit for EE.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EE	Meaning
0b0	SCTLR_EL2.EE is writeable.
0b1	SCTLR_EL2.EE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0E, bit [24]

When FEAT_MixedEndEL0 is implemented:

Mask bit for E0E.

E0E	Meaning
0b0	SCTLR_EL2.E0E is writeable.
0b1	SCTLR_EL2.E0E is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SPAN, bit [23]

Mask bit for SPAN.

SPAN	Meaning
0b0	SCTLR_EL2.SPAN is writeable.
0b1	SCTLR_EL2.SPAN is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

EIS, bit [22]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_ExS is implemented:

Mask bit for EIS.

EIS	Meaning
0b0	SCTLR_EL2.EIS is writeable.
0b1	SCTLR_EL2.EIS is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IESB, bit [21]

When FEAT_IESB is implemented:

Mask bit for IESB.

IESB	Meaning
0b0	SCTLR_EL2.IESB is writeable.
0b1	SCTLR_EL2.IESB is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TSCXT, bit [20]

When FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented:

Mask bit for TSCXT.

TSCXT	Meaning
0b0	SCTLR_EL2.TSCXT is writeable.
0b1	SCTLR_EL2.TSCXT is not writeable.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

WXN, bit [19]

Mask bit for WXN.

WXN	Meaning
`0b0`	SCTLR_EL2.WXN is writeable.
`0b1`	SCTLR_EL2.WXN is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

nTWE, bit [18]

Mask bit for nTWE.

nTWE	Meaning
`0b0`	SCTLR_EL2.nTWE is writeable.
`0b1`	SCTLR_EL2.nTWE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [17]

Reserved, RES0.

nTWI, bit [16]

Mask bit for nTWI.

nTWI	Meaning
`0b0`	SCTLR_EL2.nTWI is writeable.
`0b1`	SCTLR_EL2.nTWI is not writeable.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

UCT, bit [15]

Mask bit for UCT.

UCT	Meaning
`0b0`	SCTLR_EL2.UCT is writeable.
`0b1`	SCTLR_EL2.UCT is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

DZE, bit [14]

Mask bit for DZE.

DZE	Meaning
`0b0`	SCTLR_EL2.DZE is writeable.
`0b1`	SCTLR_EL2.DZE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

EnDB, bit [13]

When FEAT_PAuth is implemented:

Mask bit for EnDB.

EnDB	Meaning
`0b0`	SCTLR_EL2.EnDB is writeable.
`0b1`	SCTLR_EL2.EnDB is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

I, bit [12]

Mask bit for I.

I	Meaning
0b0	SCTLR_EL2.I is writeable.
0b1	SCTLR_EL2.I is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

EOS, bit [11]

When FEAT_ExS is implemented:

Mask bit for EOS.

EOS	Meaning
0b0	SCTLR_EL2.EOS is writeable.
0b1	SCTLR_EL2.EOS is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnRCTX, bit [10]

When FEAT_SPECRES is implemented:

Mask bit for EnRCTX.

EnRCTX	Meaning
0b0	SCTLR_EL2.EnRCTX is writeable.
0b1	SCTLR_EL2.EnRCTX is not writeable.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [9]

Reserved, RES0.

SED, bit [8]

When FEAT_AA32EL0 is implemented:

Mask bit for SED.

SED	Meaning
`0b0`	SCTLR_EL2.SED is writeable.
`0b1`	SCTLR_EL2.SED is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ITD, bit [7]

When FEAT_AA32EL0 is implemented:

Mask bit for ITD.

ITD	Meaning
`0b0`	SCTLR_EL2.ITD is writeable.
`0b1`	SCTLR_EL2.ITD is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

nAA, bit [6]

Mask bit for nAA.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

nAA	Meaning
0b0	SCTLR_EL2.nAA is writeable.
0b1	SCTLR_EL2.nAA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

CP15BEN, bit [5]

When FEAT_AA32EL0 is implemented:

Mask bit for CP15BEN.

CP15BEN	Meaning
0b0	SCTLR_EL2.CP15BEN is writeable.
0b1	SCTLR_EL2.CP15BEN is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SA0, bit [4]

Mask bit for SA0.

SA0	Meaning
0b0	SCTLR_EL2.SA0 is writeable.
0b1	SCTLR_EL2.SA0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

SA, bit [3]

Mask bit for SA.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SA	Meaning
0b0	SCTLR_EL2.SA is writeable.
0b1	SCTLR_EL2.SA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

C, bit [2]

Mask bit for C.

C	Meaning
0b0	SCTLR_EL2.C is writeable.
0b1	SCTLR_EL2.C is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

A, bit [1]

Mask bit for A.

A	Meaning
0b0	SCTLR_EL2.A is writeable.
0b1	SCTLR_EL2.A is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

M, bit [0]

Mask bit for M.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

M	Meaning
0b0	SCTLR_EL2.M is writeable.
0b1	SCTLR_EL2.M is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Accessing SCTLRMASK_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name SCTLRMASK_EL2 or SCTLRMASK_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SCTLRMASK_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SCTLRMASK_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCTLRMASK_EL2;
```

```
MSR SCTLRMASK_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0100	0b000

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsZero(EffectiveSCTLRMASK_EL2()) then
        UNDEFINED;
    else
        SCTLRMASK_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCTLRMASK_EL2 = X[t, 64];
```

MRS <Xt>, SCTLRMASK_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HFGRTR2_EL2.nSCTLRMASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x318];
    else
        X[t, 64] = SCTLRMASK_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        X[t, 64] = SCTLRMASK_EL2;
    else
        X[t, 64] = SCTLRMASK_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCTLRMASK_EL1;
```

MSR SCTLRMASK_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HFGWTR2_EL2.nSCTLRMASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x318] = X[t, 64];
    elsif !IsZero(EffectiveSCTLRMASK_EL1()) then
        UNDEFINED;
    else
        SCTLRMASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        if !IsZero(EffectiveSCTLRMASK_EL2()) then
            UNDEFINED;
        else
            SCTLRMASK_EL2 = X[t, 64];
    else
        SCTLRMASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCTLRMASK_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.180 SCXTNUM_EL0, EL0 Read/Write Software Context Number

The SCXTNUM_EL0 characteristics are:

Purpose

Provides a number that can be used to separate out different context numbers with the EL0 exception level, for the purpose of protecting against side-channels using branch prediction and similar resources.

Configuration

This register is present only when (FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented) and FEAT_AA64 is implemented. Otherwise, direct accesses to SCXTNUM_EL0 are UNDEFINED.

Attributes

SCXTNUM_EL0 is a 64-bit register.

Field descriptions

SCXTNUM, bits [63:0]

Software Context Number. A number to identify the context within the EL0 exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SCXTNUM_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SCXTNUM_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0000	0b111

```
if !((IsFeatureImplemented(FEAT_CSV2_2) || IsFeatureImplemented(FEAT_CSV2_1p2)) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL0) && SCTLR_EL1.TSCXT == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && HCR_EL2.EnSCXT == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HFGRTR_EL2.SCXTNUM_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.TSCXT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SCXTNUM_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.EnSCXT == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.SCXTNUM_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SCXTNUM_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SCXTNUM_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCXTNUM_EL0;
```

```
MSR SCXTNUM_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0000	0b111

```
if !((IsFeatureImplemented(FEAT_CSV2_2) || IsFeatureImplemented(FEAT_CSV2_1p2)) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL0) && SCTLR_EL1.TSCXT == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && HCR_EL2.EnSCXT == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HFGWTR_EL2.SCXTNUM_EL0 == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.TSCXT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SCXTNUM_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.EnSCXT == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.SCXTNUM_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SCXTNUM_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SCXTNUM_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCXTNUM_EL0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.181 SCXTNUM_EL1, EL1 Read/Write Software Context Number

The SCXTNUM_EL1 characteristics are:

Purpose

Provides a number that can be used to separate out different context numbers with the EL1 exception level, for the purpose of protecting against side-channels using branch prediction and similar resources.

Configuration

This register is present only when (FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented) and FEAT_AA64 is implemented. Otherwise, direct accesses to SCXTNUM_EL1 are UNDEFINED.

Attributes

SCXTNUM_EL1 is a 64-bit register.

Field descriptions

SCXTNUM, bits [63:0]

Software Context Number. A number to identify the context within the EL1 exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SCXTNUM_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name `SCXTNUM_EL1` or `SCXTNUM_EL12` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SCXTNUM_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b111

```
if !((IsFeatureImplemented(FEAT_CSV2_2) || IsFeatureImplemented(FEAT_CSV2_1p2)) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() == '011' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.EnSCXT == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.SCXTNUM_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x188];
    else
        X[t, 64] = SCXTNUM_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = SCXTNUM_EL2;
    else
        X[t, 64] = SCXTNUM_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCXTNUM_EL1;
```

```
MSR SCXTNUM_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b111

```
if !((IsFeatureImplemented(FEAT_CSV2_2) || IsFeatureImplemented(FEAT_CSV2_1p2)) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.EnSCXT == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.SCXTNUM_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x188] = X[t, 64];
    else
        SCXTNUM_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        SCXTNUM_EL2 = X[t, 64];
    else
        SCXTNUM_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCXTNUM_EL1 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, SCXTNUM_EL12

op0	op1	CRn	CRm	op2
0b11	0b101	0b1101	0b0000	0b111

```
if !((IsFeatureImplemented(FEAT_CSV2_2) || IsFeatureImplemented(FEAT_CSV2_1p2)) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x188];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = SCXTNUM_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = SCXTNUM_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented MSR SCXTNUM_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b1101	0b0000	0b111

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !((IsFeatureImplemented(FEAT_CSV2_2) || IsFeatureImplemented(FEAT_CSV2_1p2)) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x188] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            SCXTNUM_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        SCXTNUM_EL1 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.182 SCXTNUM_EL2, EL2 Read/Write Software Context Number

The SCXTNUM_EL2 characteristics are:

Purpose

Provides a number that can be used to separate out different context numbers with the EL2 exception level, for the purpose of protecting against side-channels using branch prediction and similar resources.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when (FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented) and FEAT_AA64 is implemented. Otherwise, direct accesses to SCXTNUM_EL2 are UNDEFINED.

Attributes

SCXTNUM_EL2 is a 64-bit register.

Field descriptions

SCXTNUM, bits [63:0]

Software Context Number. A number to identify the context within the EL2 exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SCXTNUM_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name `SCXTNUM_EL2` or `SCXTNUM_EL1` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SCXTNUM_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1101	0b0000	0b111

```
if !((IsFeatureImplemented(FEAT_CSV2_2) || IsFeatureImplemented(FEAT_CSV2_1p2)) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SCXTNUM_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCXTNUM_EL2;
```

MSR SCXTNUM_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1101	0b0000	0b111

```
if !((IsFeatureImplemented(FEAT_CSV2_2) || IsFeatureImplemented(FEAT_CSV2_1p2)) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SCXTNUM_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCXTNUM_EL2 = X[t, 64];
```

MRS <Xt>, SCXTNUM_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b111

```
if !((IsFeatureImplemented(FEAT_CSV2_2) || IsFeatureImplemented(FEAT_CSV2_1p2)) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.EnSCXT == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.SCXTNUM_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x188];
    else
        X[t, 64] = SCXTNUM_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = SCXTNUM_EL2;
    else
        X[t, 64] = SCXTNUM_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCXTNUM_EL1;
```

```
MSR SCXTNUM_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b111

```
if !((IsFeatureImplemented(FEAT_CSV2_2) || IsFeatureImplemented(FEAT_CSV2_1p2)) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.EnSCXT == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.SCXTNUM_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        NVMem[0x188] = X[t, 64];
    else
        SCXTNUM_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        SCXTNUM_EL2 = X[t, 64];
    else
        SCXTNUM_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SCXTNUM_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.183 SCXTNUM_EL3, EL3 Read/Write Software Context Number

The SCXTNUM_EL3 characteristics are:

Purpose

Provides a number that can be used to separate out different context numbers with the EL3 exception level, for the purpose of protecting against side-channels using branch prediction and similar resources.

Configuration

This register is present only when (HaveEL(EL3) && (IsFeatureImplemented(FEAT_CSV2_2) || IsFeatureImplemented(FEAT_CSV2_1p2))) && IsFeatureImplemented(FEAT_AA64). Otherwise, direct accesses to SCXTNUM_EL3 are UNDEFINED.

Attributes

SCXTNUM_EL3 is a 64-bit register.

Field descriptions

SCXTNUM, bits [63:0]

Software Context Number. A number to identify the context within the EL3 exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SCXTNUM_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SCXTNUM_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1101	0b0000	0b111

```
if !(HaveEL(EL3) && (IsFeatureImplemented(FEAT_CSV2_2) || IsFeatureImplemented(FEAT_CSV2_1p2)) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SCXTNUM_EL3;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MSR SCXTNUM_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1101	0b0000	0b111

```
if !(HaveEL(EL3) && (IsFeatureImplemented(FEAT_CSV2_2) || IsFeatureImplemented(FEAT_CSV2_1p2)) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    SCXTNUM_EL3 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.184 SMCR_EL1, SME Control Register (EL1)

The SMCR_EL1 characteristics are:

Purpose

This register controls aspects of Streaming SVE that are visible at Exception levels EL1 and EL0.

Configuration

This register has no effect if the PE is not in Streaming SVE mode.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this register has no effect on execution at EL0.

This register is present only when FEAT_SME is implemented. Otherwise, direct accesses to SMCR_EL1 are UNDEFINED.

Attributes

SMCR_EL1 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

FA64, bit [31]

When FEAT_SME_FA64 is implemented:

Controls whether execution of an A64 instruction at EL1 is considered legal when executed in Streaming SVE mode.

When the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, controls whether execution of an A64 instruction at EL0 is considered legal when executed in Streaming SVE mode.

FA64	Meaning
0b0	This control does not cause any instruction to be treated as legal when executed in Streaming SVE mode.
0b1	This control causes all implemented A64 instructions to be treated as legal when executed in Streaming SVE mode at EL1 and EL0, if they are treated as legal at more privileged Exception levels in the current Security state.

Arm recommends that portable SME software should not rely on this optional feature, and that operating systems should provide a means to test for compliance with this recommendation.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EZT0, bit [30]

When FEAT_SME2 is implemented:

Traps execution at EL1 and EL0 of the LDR, LUTI2, LUTI4, MOVT, STR, and ZERO instructions that access the ZT0 register to EL1, or to EL2 when EL2 is implemented and enabled in the current Security state and HCR_EL2.TGE is 1.

The exception is reported using ESR_EL1.EC or ESR_EL2.EC value `0x1D`, with an ISS code of `0x0000004`, at a lower priority than a trap due to PSTATE.SM or PSTATE.ZA.

EZT0	Meaning
`0b0`	This control causes execution of these instructions at EL1 and EL0 to be trapped.
`0b1`	This control does not cause execution of any instruction to be trapped.

Changes to this field only affect whether instructions that access ZT0 are trapped. They do not affect the contents of ZT0, which remain valid so long as PSTATE.ZA is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [29:9]

Reserved, RES0.

Bits [8:4]

Reserved, RAZ/WI.

LEN, bits [3:0]

Requests an Effective Streaming SVE vector length (SVL) at EL1 of (LEN+1)*128 bits. This field also defines the Effective Streaming SVE vector length at EL0 when EL2 is not implemented, or EL2 is not enabled in the current Security state, or the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.

The Streaming SVE vector length can be any power of two from 128 bits to 2048 bits inclusive. An implementation can support any subset of the architecturally permitted lengths.

When the PE is in Streaming SVE mode, the Effective SVE vector length (VL) is equal to SVL.

When FEAT_SVE is implemented, and the PE is not in Streaming SVE mode, VL is equal to the Effective Non-streaming SVE vector length. See ZCR_EL1.

For all purposes other than returning the result of a direct read of SMCR_EL1, the PE selects the Effective Streaming SVE vector length by performing checks in the following order:

1. If the requested length is less than the minimum implemented Streaming SVE vector length, then the Effective length is the minimum implemented Streaming SVE vector length.
2. If EL2 is implemented and enabled in the current Security state, and the requested length is greater than the Effective length at EL2, then the Effective length at EL2 is used.
3. If EL3 is implemented and the requested length is greater than the Effective length at EL3, then the Effective length at EL3 is used.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

4. Otherwise, the Effective length is the highest supported Streaming SVE vector length that is less than or equal to the requested length.

An indirect read of SMCR_EL1.LEN appears to occur in program order relative to a direct write of the same register, without the need for explicit synchronization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SMCR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name `SMCR_EL1` or `SMCR_EL12` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SMCR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b110

```
if !IsFeatureImplemented(FEAT_SME) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif CPACR_EL1.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL1, 0x1D);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif ELIsInHost(EL2) && CPTR_EL2.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x1F0];
    else
        X[t, 64] = SMCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL2) && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif ELIsInHost(EL2) && CPTR_EL2.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    elsif ELIsInHost(EL2) then
        X[t, 64] = SMCR_EL2;
    else
        X[t, 64] = SMCR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        X[t, 64] = SMCR_EL1;
```

MSR SMCR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b110

```
if !IsFeatureImplemented(FEAT_SME) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif CPACR_EL1.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL1, 0x1D);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif ELIsInHost(EL2) && CPTR_EL2.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x1F0] = X[t, 64];
    else
        SMCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL2) && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif ELIsInHost(EL2) && CPTR_EL2.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    elsif ELIsInHost(EL2) then
        SMCR_EL2 = X[t, 64];
    else
        SMCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        SMCR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, SMCR_EL12

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0010	0b110

```
if !IsFeatureImplemented(FEAT_SME) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x1F0];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
            UNDEFINED;
        elsif CPTR_EL2.SMEN IN {'x0'} then
            AArch64.SystemAccessTrap(EL2, 0x1D);
        elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x1D);
        else
            X[t, 64] = SMCR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        if CPTR_EL3.ESM == '0' then
            AArch64.SystemAccessTrap(EL3, 0x1D);
        else
            X[t, 64] = SMCR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented MSR SMCR_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0010	0b110

```
if !IsFeatureImplemented(FEAT_SME) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x1F0] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        elsif CPTR_EL2.SMEN IN {'x0'} then
            AArch64.SystemAccessTrap(EL2, 0x1D);
        elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x1D);
        else
            SMCR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        if CPTR_EL3.ESM == '0' then
            AArch64.SystemAccessTrap(EL3, 0x1D);
        else
            SMCR_EL1 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.185 SMCR_EL2, SME Control Register (EL2)

The SMCR_EL2 characteristics are:

Purpose

This register controls aspects of Streaming SVE that are visible at Exception levels EL2, EL1, and EL0.

Configuration

This register has no effect if the PE is not in Streaming SVE mode, or if EL2 is not enabled in the current Security state.

If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_SME is implemented. Otherwise, direct accesses to SMCR_EL2 are UNDEFINED.

Attributes

SMCR_EL2 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

FA64, bit [31]

When FEAT_SME_FA64 is implemented:

Controls whether execution of an A64 instruction at EL2, EL1, and EL0 when EL2 is implemented and enabled in the current Security state is considered legal when executed in Streaming SVE mode.

FA64	Meaning
0b0	This control does not cause any instruction to be treated as legal when executed in Streaming SVE mode.
0b1	This control causes all implemented A64 instructions to be treated as legal when executed in Streaming SVE mode at EL2, EL1, and EL0, if they are treated as legal at EL3.

Arm recommends that portable SME software should not rely on this optional feature, and that operating systems should provide a means to test for compliance with this recommendation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

EZT0, bit [30]

When FEAT_SME2 is implemented:

Traps execution at EL2, EL1, and EL0 of the LDR, LUTI2, LUTI4, MOVT, STR, and ZERO instructions that access the ZT0 register to EL2, when EL2 is enabled in the current Security state.

The exception is reported using ESR_EL2.EC value `0x1D`, with an ISS code of `0x0000004`, at a lower priority than a trap due to PSTATE.SM or PSTATE.ZA.

EZT0	Meaning
`0b0`	This control causes execution of these instructions at EL2, EL1, and EL0 to be trapped.
`0b1`	This control does not cause execution of any instruction to be trapped.

Changes to this field only affect whether instructions that access ZT0 are trapped. They do not affect the contents of ZT0, which remain valid so long as PSTATE.ZA is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [29:9]

Reserved, RES0.

Bits [8:4]

Reserved, RAZ/WI.

LEN, bits [3:0]

Requests an Effective Streaming SVE vector length (SVL) at EL2 of (LEN+1)*128 bits. This field also defines the Effective Streaming SVE vector length at EL0 when the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}.

The Streaming SVE vector length can be any power of two from 128 bits to 2048 bits inclusive. An implementation can support any subset of the architecturally permitted lengths.

When the PE is in Streaming SVE mode, the Effective SVE vector length (VL) is equal to SVL.

When FEAT_SVE is implemented, and the PE is not in Streaming SVE mode, VL is equal to the Effective Non-streaming SVE vector length. See ZCR_EL2.

For all purposes other than returning the result of a direct read of SMCR_EL2, the PE selects the Effective Streaming SVE vector length by performing checks in the following order:

1. If the requested length is less than the minimum implemented Streaming SVE vector length, then the Effective length is the minimum implemented Streaming SVE vector length.
2. If EL3 is implemented and the requested length is greater than the Effective length at EL3, then the Effective length at EL3 is used.
3. Otherwise, the Effective length is the highest supported Streaming SVE vector length that is less than or equal to the requested length.

An indirect read of SMCR_EL2.LEN appears to occur in program order relative to a direct write of the same register, without the need for explicit synchronization.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SMCR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name SMCR_EL2 or SMCR_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SMCR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0010	0b110

```
if !IsFeatureImplemented(FEAT_SME) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL2) && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif ELIsInHost(EL2) && CPTR_EL2.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        X[t, 64] = SMCR_EL2;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        X[t, 64] = SMCR_EL2;
```

```
MSR SMCR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0010	0b110

```
if !IsFeatureImplemented(FEAT_SME) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL2) && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif ELIsInHost(EL2) && CPTR_EL2.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        SMCR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        SMCR_EL2 = X[t, 64];
```

MRS <Xt>, SMCR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b110

```
if !IsFeatureImplemented(FEAT_SME) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif CPACR_EL1.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL1, 0x1D);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif ELIsInHost(EL2) && CPTR_EL2.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x1F0];
    else
        X[t, 64] = SMCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL2) && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif ELIsInHost(EL2) && CPTR_EL2.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    elsif ELIsInHost(EL2) then
        X[t, 64] = SMCR_EL2;
    else
        X[t, 64] = SMCR_EL1;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        X[t, 64] = SMCR_EL1;
```

MSR SMCR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b110

```
if !IsFeatureImplemented(FEAT_SME) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif CPACR_EL1.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL1, 0x1D);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif ELIsInHost(EL2) && CPTR_EL2.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x1F0] = X[t, 64];
    else
        SMCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL2) && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif ELIsInHost(EL2) && CPTR_EL2.SMEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    elsif ELIsInHost(EL2) then
        SMCR_EL2 = X[t, 64];
    else
        SMCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        SMCR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.186 SMCR_EL3, SME Control Register (EL3)

The SMCR_EL3 characteristics are:

Purpose

This register controls aspects of Streaming SVE that are visible at all Exception levels.

Configuration

This register has no effect if the PE is not in Streaming SVE mode.

This register is present only when FEAT_SME is implemented and EL3 is implemented. Otherwise, direct accesses to SMCR_EL3 are UNDEFINED.

Attributes

SMCR_EL3 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

FA64, bit [31]

When FEAT_SME_FA64 is implemented:

Controls whether execution of an A64 instruction is considered legal when executed in Streaming SVE mode.

FA64	Meaning
0b0	This control does not cause any instruction to be treated as legal when executed in Streaming SVE mode.
0b1	This control causes all implemented A64 instructions to be treated as legal when executed in Streaming SVE mode .

Arm recommends that portable SME software should not rely on this optional feature, and that operating systems should provide a means to test for compliance with this recommendation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EZT0, bit [30]

When FEAT_SME2 is implemented:

Traps execution at all Exception levels of the LDR, LUTI2, LUTI4, MOVT, STR, and ZERO instructions that access the ZT0 register to EL3.

The exception is reported using ESR_EL3.EC value `0x1D`, with an ISS code of `0x0000004`, at a lower priority than a trap due to PSTATE.SM or PSTATE.ZA.

EZT0	Meaning
`0b0`	This control causes execution of these instructions at all Exception levels to be trapped.
`0b1`	This control does not cause execution of any instruction to be trapped.

Changes to this field only affect whether instructions that access ZT0 are trapped. They do not affect the contents of ZT0, which remain valid so long as PSTATE.ZA is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [29:9]

Reserved, RES0.

Bits [8:4]

Reserved, RAZ/WI.

LEN, bits [3:0]

Requests an Effective Streaming SVE vector length (SVL) at EL3 of (LEN+1)*128 bits.

The Streaming SVE vector length can be any power of two from 128 bits to 2048 bits inclusive. An implementation can support any subset of the architecturally permitted lengths.

When the PE is in Streaming SVE mode, the Effective SVE vector length (VL) is equal to SVL.

When FEAT_SVE is implemented, and the PE is not in Streaming SVE mode, VL is equal to the Effective Non-streaming SVE vector length. See ZCR_EL3.

For all purposes other than returning the result of a direct read of SMCR_EL3, the PE selects the Effective Streaming SVE vector length by performing checks in the following order:

1. If the requested length is less than the minimum implemented Streaming SVE vector length, then the Effective length is the minimum implemented Streaming SVE vector length.
2. Otherwise, the Effective length is the highest supported Streaming SVE vector length that is less than or equal to the requested length.

An indirect read of SMCR_EL3.LEN appears to occur in program order relative to a direct write of the same register, without the need for explicit synchronization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SMCR_EL3

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, SMCR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0010	0b110

```
if !(IsFeatureImplemented(FEAT_SME) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        X[t, 64] = SMCR_EL3;
```

```
MSR SMCR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0010	0b110

```
if !(IsFeatureImplemented(FEAT_SME) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        SMCR_EL3 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.187 SMIDR_EL1, Streaming Mode Identification Register

The SMIDR_EL1 characteristics are:

Purpose

Provides additional identification mechanisms for scheduling purposes, for a PE that supports Streaming SVE mode.

Configuration

This register is present only when FEAT_SME is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SMIDR_EL1 are UNDEFINED.

Attributes

SMIDR_EL1 is a 64-bit register.

Field descriptions

Bits [63:60]

Reserved, RES0.

NSMC, bits [59:56]

If SMIDR_EL1.{SH,Affinity} indicates that the implementation of Streaming SVE mode is shared, then this field identifies the number of SMCUs, minus 1, associated with the concatenated SMIDR_EL1.{Affinity2,Affinity} 32-bit value.

If SMIDR_EL1.{SH,Affinity} indicates that the implementation of Streaming SVE mode is not shared, then this field is zero and should be ignored by software.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NSMC	Meaning
0b0000	The implementation of Streaming SVE mode associated with this PE is not shared or is a single SMCU.
0b0001..0b1110	The number of SMCUs in the group of SMCUs providing the implementation of Streaming SVE mode for this PE, minus 1.
0b1111	Reserved.

Access to this field is RO.

HIP, bits [55:52]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_SME2p2 is implemented and SMIDR_EL1.SMPS == '1':

Highest Implemented Priority. If Streaming SVE mode execution priority is supported, this field indicates the range of priority levels implemented by the PE, and the Highest Implemented Priority value.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HIP	Meaning
0b0000	All streaming execution priorities from 0 to 15 are implemented. The Highest Implemented Priority value is 15.
0b0001..0b1111	All streaming execution priorities less than or equal to this value are implemented. The Highest Implemented Priority value is the value of this field.

Access to this field is RO.

Otherwise:

Reserved, RES0.

Affinity2, bits [51:32]

The most significant 20 bits of the SMCU affinity for this PE, to be used in conjunction with SMIDR_EL1.Affinity.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Implementer, bits [31:24]

The Implementer code. This field must hold an implementer code that has been assigned by Arm.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Implementer	Meaning
0x00	Reserved for software use.
0x41	Arm Limited.
0x42	Broadcom Corporation.
0x43	Cavium Inc.
0x44	Digital Equipment Corporation.
0x46	Fujitsu Ltd.
0x49	Infineon Technologies AG.
0x4D	Motorola or Freescale Semiconductor Inc.
0x4E	NVIDIA Corporation.
0x50	Applied Micro Circuits Corporation.
0x51	Qualcomm Inc.
0x56	Marvell International Ltd.
0x69	Intel Corporation.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Implementer	Meaning
0xC0	Ampere Computing.

Arm can assign codes that are not published in this manual. All values not assigned by Arm are reserved and must not be used.

It is not required that this value is the same as the value of MIDR_EL1.Implementer.

Access to this field is RO.

Revision, bits [23:16]

Revision number for the Streaming Mode Compute Unit (SMCU).

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

SMPS, bit [15]

Indicates support for Streaming SVE mode execution priority.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SMPS	Meaning
0b0	Priority control not supported.
0b1	Priority control supported.

Access to this field is RO.

SH, bits [14:13]

Indicates whether the implementation of Streaming SVE mode in this PE is shared with other PEs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SH	Meaning
0b00	Refer to SMIDR_EL1.Affinity.
0b01	Reserved.
0b10	The implementation of Streaming SVE mode is not shared with other PEs.
0b11	The implementation of Streaming SVE mode is shared with other PEs.

Access to this field is RO.

Bit [12]

Reserved, RES0.

Affinity, bits [11:0]

The least significant 12 bits of the SMCU affinity for this PE.

This field has an IMPLEMENTATION DEFINED value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

If the implementation of Streaming SVE mode is shared, then the concatenated SMIDR_EL1.{Affinity2,Affinity} 32-bit value identifies which shared SMCUs are associated with this PE. Every PE that shares the same SMCUs has the same 32-bit affinity value. The 32-bit affinity value is unique within the system as a whole.

The SMIDR_EL1.SH field indicates whether the implementation of Streaming SVE mode is shared with other PEs. However, if SMIDR_EL1.SH is zero, then SMIDR_EL1.Affinity is used to indicate whether the implementation of Streaming SVE is shared, as follows:

- If SMIDR_EL1.Affinity is zero, then the implementation of Streaming SVE mode is not shared with other PEs.
- If SMIDR_EL1.Affinity is not zero, then the implementation of Streaming SVE mode is shared with other PEs.

Access to this field is RO.

Accessing SMIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SMIDR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b001	0b0000	0b0000	0b110

```
if !(IsFeatureImplemented(FEAT_SME) && IsFeatureImplemented(FEAT_AA64)) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = SMIDR_EL1;
elsif PSTATE.EL == EL2 then
    X[t, 64] = SMIDR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SMIDR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.188 SMPRI_EL1, Streaming Mode Priority Register

The SMPRI_EL1 characteristics are:

Purpose

Configures the streaming execution priority for instructions executed on a shared Streaming Mode Compute Unit (SMCU) when the PE is in Streaming SVE mode at any Exception level.

Configuration

When SMIDR_EL1.SMPS is ‘0’, this register is RES0.

This register is present only when FEAT_SME is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SMPRI_EL1 are UNDEFINED.

Attributes

SMPRI_EL1 is a 64-bit register.

Field descriptions

Bits [63:4]

Reserved, RES0.

Priority, bits [3:0]

Streaming execution priority value.

Either this value is used directly, or it is mapped into an effective priority value using SMPRIMAP_EL2.

This value is used directly when any of the following are true:

- The current Exception level is EL3 or EL2.
- The current Exception level is EL1 or EL0, if EL2 is implemented and enabled in the current Security state and HCRX_EL2.SMPME is ‘0’.
- The current Exception level is EL1 or EL0, if EL2 is either not implemented or not enabled in the current Security state.

The precise meaning and behavior of each streaming execution priority value is IMPLEMENTATION DEFINED.

In an implementation that shares execution resources between PEs, higher priority values are allocated more processing resource than other PEs configured with lower priority values in the same Priority domain.

If FEAT_SME2p2 is implemented, and this field is greater than the Highest Implemented Priority value indicated by SMIDR_EL1.HIP, then the PE uses the Highest Implemented Priority value as the priority.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SMPRI_EL1

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, SMPRI_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b100

```
if !(IsFeatureImplemented(FEAT_SME) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.nSMPRI_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SMPRI_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SMPRI_EL1;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SMPRI_EL1;
```

```
MSR SMPRI_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b100

```
if !(IsFeatureImplemented(FEAT_SME) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.nSMPRI_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SMPRI_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SMPRI_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SMPRI_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.189 SMPRIMAP_EL2, Streaming Mode Priority Mapping Register

The SMPRIMAP_EL2 characteristics are:

Purpose

Maps the value in SMPRI_EL1 to a streaming execution priority value for instructions executed at EL1 and EL0 in the same Security states as EL2.

Configuration

When SMIDR_EL1.SMPS is ‘0’, this register is RES0.

If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_SME is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SMPRIMAP_EL2 are UNDEFINED.

Attributes

SMPRIMAP_EL2 is a 64-bit register.

Field descriptions

When all of the following are true, the value in SMPRI_EL1 is mapped to a streaming execution priority using this register:

- The current Exception level is EL1 or EL0.
- EL2 is implemented and enabled in the current Security state.
- HCRX_EL2.SMPME is ‘1’.

Otherwise, SMPRI_EL1 holds the streaming execution priority value.

P15, bits [63:60]

Priority Mapping Entry 15. This entry is used when priority mapping is supported and enabled, and the SMPRI_EL1.Priority value is ‘15’.

This value is the highest streaming execution priority.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P14, bits [59:56]

Priority Mapping Entry 14. This entry is used when priority mapping is supported and enabled, and the SMPRI_EL1.Priority value is ‘14’.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

P13, bits [55:52]

Priority Mapping Entry 13. This entry is used when priority mapping is supported and enabled, and the SMPRI_EL1.Priority value is ‘13’.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P12, bits [51:48]

Priority Mapping Entry 12. This entry is used when priority mapping is supported and enabled, and the SMPRI_EL1.Priority value is ‘12’.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P11, bits [47:44]

Priority Mapping Entry 11. This entry is used when priority mapping is supported and enabled, and the SMPRI_EL1.Priority value is ‘11’.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P10, bits [43:40]

Priority Mapping Entry 10. This entry is used when priority mapping is supported and enabled, and the SMPRI_EL1.Priority value is ‘10’.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P9, bits [39:36]

Priority Mapping Entry 9. This entry is used when priority mapping is supported and enabled, and the SMPRI_EL1.Priority value is ‘9’.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P8, bits [35:32]

Priority Mapping Entry 8. This entry is used when priority mapping is supported and enabled, and the SMPRI_EL1.Priority value is ‘8’.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P7, bits [31:28]

Priority Mapping Entry 7. This entry is used when priority mapping is supported and enabled, and the SMPRI_EL1.Priority value is ‘7’.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P6, bits [27:24]

Priority Mapping Entry 6. This entry is used when priority mapping is supported and enabled, and the SMPRI_EL1.Priority value is ‘6’.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P5, bits [23:20]

Priority Mapping Entry 5. This entry is used when priority mapping is supported and enabled, and the SMPRI_EL1.Priority value is '5'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P4, bits [19:16]

Priority Mapping Entry 4. This entry is used when priority mapping is supported and enabled, and the SMPRI_EL1.Priority value is '4'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P3, bits [15:12]

Priority Mapping Entry 3. This entry is used when priority mapping is supported and enabled, and the SMPRI_EL1.Priority value is '3'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P2, bits [11:8]

Priority Mapping Entry 2. This entry is used when priority mapping is supported and enabled, and the SMPRI_EL1.Priority value is '2'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P1, bits [7:4]

Priority Mapping Entry 1. This entry is used when priority mapping is supported and enabled, and the SMPRI_EL1.Priority value is '1'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P0, bits [3:0]

Priority Mapping Entry 0. This entry is used when priority mapping is supported and enabled, and the SMPRI_EL1.Priority value is '0'.

This value is the lowest streaming execution priority.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SMPRIMAP_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SMPRIMAP_EL2
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0010	0b101

```
if !(IsFeatureImplemented(FEAT_SME) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x1F8];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SMPRIMAP_EL2;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SMPRIMAP_EL2;
```

MSR SMPRIMAP_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0010	0b101

```
if !(IsFeatureImplemented(FEAT_SME) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x1F8] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SMPRIMAP_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SMPRIMAP_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.190 TCR2_EL1, Extended Translation Control Register (EL1)

The TCR2_EL1 characteristics are:

Purpose

The control register for stage 1 of the EL1&0 translation regime.

Configuration

This register is present only when FEAT_TCR2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TCR2_EL1 are UNDEFINED.

Attributes

TCR2_EL1 is a 64-bit register.

Field descriptions

Unless stated otherwise, all the bits in TCR2_EL1 are permitted to be cached in a TLB.

Bits [63:22]

Reserved, RES0.

FNGNA1, bit [21]

When FEAT_THE is implemented:

Force non-global for unassured translations using TTBR1_EL1.

FNGNA1	Meaning
0b0	This bit has no effect on the interpretation of the nG bit.
0b1	Translations using TTBR1_EL1 are treated as non-global regardless of the value of the nG bit if all of the following is true: • The translation is for the EL1&0 translation regime. • Stage 1 and stage 2 translation are enabled. • Protection is enabled. • The final stage 1 translation using the descriptor does not have the Assured Translation property.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

This bit is permitted to be cached in a TLB.

This field is ignored by the PE and treated as zero when any of the following are true:

- All of the following are true:
  - EL2 is implemented and enabled in the current Security state.
  - The Effective value of HCRX_EL2.TCR2En is 0.
- EL3 is implemented and SCR_EL3.TCR2En == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FNGNA0, bit [20]

When FEAT_THE is implemented:

Force non-global for unassured translations using TTBR0_EL1.

FNGNA0	Meaning
0b0	This bit has no effect on the interpretation of the nG bit.
0b1	Translations using TTBR0_EL1 are treated as non-global regardless of the value of the nG bit if all of the following is true: • The translation is for the EL1&0 translation regime. • Stage 1 and stage 2 translation are enabled. • Protection is enabled. • The final stage 1 translation using the descriptor does not have the Assured Translation property.

This bit is permitted to be cached in a TLB.

This field is ignored by the PE and treated as zero when any of the following are true:

- All of the following are true:
  - EL2 is implemented and enabled in the current Security state.
  - The Effective value of HCRX_EL2.TCR2En is 0.
- EL3 is implemented and SCR_EL3.TCR2En == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [19]

Reserved, RES0.

FNG1, bit [18]

When FEAT_ASID2 is implemented:

Force non-global translations for TTBR1_EL1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FNG1	Meaning
0b0	This bit has no effect on the interpretation of the nG bit.
0b1	Translations using TTBR1_EL1 are treated as non-global regardless of the value of the nG bit.

This bit is permitted to be cached in a TLB.

This field is ignored by the PE and treated as zero when any of the following are true:

- All of the following are true:
  - EL2 is implemented and enabled in the current Security state.
  - The Effective value of HCRX_EL2.TCR2En is 0.
- EL3 is implemented and SCR_EL3.TCR2En == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FNG0, bit [17]

When FEAT_ASID2 is implemented:

Force non-global translations for TTBR0_EL1.

FNG0	Meaning
0b0	This bit has no effect on the interpretation of the nG bit.
0b1	Translations using TTBR0_EL1 are treated as non-global regardless of the value of the nG bit.

This bit is permitted to be cached in a TLB.

This field is ignored by the PE and treated as zero when any of the following are true:

- All of the following are true:
  - EL2 is implemented and enabled in the current Security state.
  - The Effective value of HCRX_EL2.TCR2En is 0.
- EL3 is implemented and SCR_EL3.TCR2En == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

A2, bit [16]

When FEAT_ASID2 is implemented:

Enable use of two ASIDs.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

A2	Meaning
0b0	Use of two ASIDs is disabled.
0b1	Use of two ASIDs is enabled.

This bit is permitted to be cached in a TLB.

This field is ignored by the PE and treated as zero when any of the following are true:

- All of the following are true:
  - EL2 is implemented and enabled in the current Security state.
  - The Effective value of HCRX_EL2.TCR2En is 0.
- EL3 is implemented and SCR_EL3.TCR2En == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DisCH1, bit [15]

When FEAT_D128 is implemented and TCR2_EL1.D128 == '1':

Disable the Contiguous bit for the Start Table for TTBR1_EL1.

DisCH1	Meaning
0b0	The Contiguous bit of Block or Page descriptors of the Start Table for TTBR1_EL1 is not affected by this field.
0b1	The Contiguous bit of Block or Page descriptors of the Start Table for TTBR1_EL1 is treated as 0.

This field is ignored by the PE and treated as zero when any of the following are true:

- All of the following are true:
  - EL2 is implemented and enabled in the current Security state.
  - The Effective value of HCRX_EL2.TCR2En is 0.
- EL3 is implemented and SCR_EL3.TCR2En == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DisCH0, bit [14]

When FEAT_D128 is implemented and TCR2_EL1.D128 == '1':

Disable the Contiguous bit for the Start Table for TTBR0_EL1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

DisCH0	Meaning
0b0	The Contiguous bit of Block or Page descriptors of the Start Table for TTBR0_EL1 is not affected by this field.
0b1	The Contiguous bit of Block or Page descriptors of the Start Table for TTBR0_EL1 is treated as 0.

This field is ignored by the PE and treated as zero when any of the following are true:

- All of the following are true:
  - EL2 is implemented and enabled in the current Security state.
  - The Effective value of HCRX_EL2.TCR2En is 0.
- EL3 is implemented and SCR_EL3.TCR2En == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [13:12]

Reserved, RES0.

HAFT, bit [11]

When FEAT_HAFT is implemented:

Hardware managed Access Flag for Table descriptors.

Enables the Hardware managed Access Flag for Table descriptors.

HAFT	Meaning
0b0	Hardware managed Access Flag for Table descriptors is disabled.
0b1	Hardware managed Access Flag for Table descriptors is enabled.

This field is ignored by the PE and treated as zero when any of the following are true:

- All of the following are true:
  - EL2 is implemented and enabled in the current Security state.
  - The Effective value of HCRX_EL2.TCR2En is 0.
- EL3 is implemented and SCR_EL3.TCR2En == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PTTWI, bit [10]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_THE is implemented:

Permit Translation table walk Incoherence.

Permits RCWS instructions to generate writes that have the Reduced Coherence property.

PTTWI	Meaning
0b0	Write accesses generated by RCWS at EL1&0 do not have the Reduced Coherence property.
0b1	Write accesses generated by RCWS at EL1&0 have the Reduced Coherence property if HCRX_EL2.PTTWI is 1.

This bit is permitted to be implemented as a read-only bit with a fixed value of 0.

This field is ignored by the PE and treated as zero when any of the following are true:

- All of the following are true:
  - EL2 is implemented and enabled in the current Security state.
  - The Effective value of HCRX_EL2.TCR2En is 0.
- EL3 is implemented and SCR_EL3.TCR2En == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [9:6]

Reserved, RES0.

D128, bit [5]

When FEAT_D128 is implemented:

Enables VMSAv9-128 translation system.

D128	Meaning
0b0	Translation system follows VMSAv8-64 translation process.
0b1	Translation system follows VMSAv9-128 translation process.

This field is ignored by the PE and treated as zero when any of the following are true:

- All of the following are true:
  - EL2 is implemented and enabled in the current Security state.
  - The Effective value of HCRX_EL2.TCR2En is 0.
- EL3 is implemented and SCR_EL3.TCR2En == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AIE, bit [4]

When FEAT_AIE is implemented:

Enable Attribute Indexing Extension.

AIE	Meaning
0b0	Attribute Indexing Extension Disabled.
0b1	Attribute Indexing Extension Enabled.

This field is RES1 when TCR2_EL1.D128 is 1.

This field is ignored by the PE and treated as zero when any of the following are true:

- All of the following are true:
  - EL2 is implemented and enabled in the current Security state.
  - The Effective value of HCRX_EL2.TCR2En is 0.
- EL3 is implemented and SCR_EL3.TCR2En == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

POE, bit [3]

When FEAT_S1POE is implemented:

Enables Permission Overlays for privileged accesses from EL1&0 translation regime.

POE	Meaning
0b0	Permission overlay disabled for EL1 access in stage 1 of EL1&0 translation regime.
0b1	Permission overlay enabled for EL1 access in stage 1 of EL1&0 translation regime.

This bit is not permitted to be cached in a TLB.

This field is ignored by the PE and treated as zero when any of the following are true:

- All of the following are true:
  - EL2 is implemented and enabled in the current Security state.
  - The Effective value of HCRX_EL2.TCR2En is 0.
- EL3 is implemented and SCR_EL3.TCR2En == 0.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0POE, bit [2]

When FEAT_S1POE is implemented:

Enables Permission Overlays for unprivileged accesses from EL1&0 translation regime.

E0POE	Meaning
`0b0`	Permission overlay disabled for EL0 access in stage 1 of EL1&0 translation regime.
`0b1`	Permission overlay enabled for EL0 access in stage 1 of EL1&0 translation regime.

This bit is not permitted to be cached in a TLB.

This field is ignored by the PE and treated as zero when any of the following are true:

- All of the following are true:
  - EL2 is implemented and enabled in the current Security state.
  - The Effective value of HCRX_EL2.TCR2En is 0.
- EL3 is implemented and SCR_EL3.TCR2En == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PIE, bit [1]

When FEAT_S1PIE is implemented:

Enables usage of Indirect Permission Scheme.

PIE	Meaning
`0b0`	Direct permission model.
`0b1`	Indirect permission model.

This field is RES1 when TCR2_EL1.D128 is 1.

This field is ignored by the PE and treated as zero when any of the following are true:

- All of the following are true:
  - EL2 is implemented and enabled in the current Security state.
  - The Effective value of HCRX_EL2.TCR2En is 0.
- EL3 is implemented and SCR_EL3.TCR2En == 0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PnCH, bit [0]

When FEAT_THE is implemented:

Protected attribute enable. Enables use of bit[52] of the stage 1 translation table entries as the Protected bit, for translations using TTBRn_EL1.

PnCH	Meaning
`0b0`	For translations using TTBRn_EL1, bit[52] of each stage 1 translation table entry is not the Protected bit.
`0b1`	For translations using TTBRn_EL1, bit[52] of each stage 1 translation table entry is the Protected bit.

If bit[52] is used as the Protected bit, it is not used as the Contiguous bit.

This field is RES0 when TCR2_EL1.D128 is 1.

This field is ignored by the PE and treated as zero when any of the following are true:

- All of the following are true:
  - EL2 is implemented and enabled in the current Security state.
  - The Effective value of HCRX_EL2.TCR2En is 0.
- EL3 is implemented and SCR_EL3.TCR2En == 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TCR2_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name `TCR2_EL1` or `TCR2_EL12` are not guaranteed to be ordered with respect to accesses using the other accessor name.

If FEAT_SRMASK is implemented, accesses to TCR2_EL1 are masked by TCR2MASK_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TCR2_EL1
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_TCR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TCR2En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.TCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.TCR2En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TCR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x270];
    else
        X[t, 64] = TCR2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TCR2En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TCR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = TCR2_EL2;
    else
        X[t, 64] = TCR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TCR2_EL1;
```

MSR TCR2_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_TCR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TCR2En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.TCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.TCR2En == '0') then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.TCR2En == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EffectiveHCR_EL2_NVx() IN {'111'} then
            NVMem[0x270] = X[t, 64];
        else
            if IsFeatureImplemented(FEAT_SRMASK) then
                TCR2_EL1 = (X[t, 64] AND NOT EffectiveTCR2MASK_EL1()) OR (TCR2_EL1 AND
                    ↪EffectiveTCR2MASK_EL1());
            else
                TCR2_EL1 = X[t, 64];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TCR2En == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.TCR2En == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif ELIsInHost(EL2) then
            if IsFeatureImplemented(FEAT_SRMASK) then
                TCR2_EL2 = (X[t, 64] AND NOT EffectiveTCR2MASK_EL2()) OR (TCR2_EL2 AND
                    ↪EffectiveTCR2MASK_EL2());
            else
                TCR2_EL2 = X[t, 64];
        else
            TCR2_EL1 = X[t, 64];
    elsif PSTATE.EL == EL3 then
        TCR2_EL1 = X[t, 64];
```

MRS <Xt>, TCR2_EL12

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_TCR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x270];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TCR2En == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.TCR2En == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            X[t, 64] = TCR2_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = TCR2_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented MSR TCR2_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_TCR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x270] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TCR2En == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.TCR2En == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TCR2_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        TCR2_EL1 = X[t, 64];
    else
        UNDEFINED;
```

When FEAT_SRMASK is implemented MRS <Xt>, TCR2ALIAS_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0111	0b111

```
if !(IsFeatureImplemented(FEAT_TCR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TCR2En == '0' then
            UNDEFINED;
        elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
             ↪'0') || HFGRTR2_EL2.nTCR2ALIAS_EL1 == '0') then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.TCR2En == '0') then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.TCR2En == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EffectiveHCR_EL2_NVx() IN {'111'} then
            X[t, 64] = NVMem[0x270];
        else
            X[t, 64] = TCR2_EL1;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TCR2En == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.TCR2En == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif ELIsInHost(EL2) then
            X[t, 64] = TCR2_EL2;
        else
            X[t, 64] = TCR2_EL1;
    elsif PSTATE.EL == EL3 then
        X[t, 64] = TCR2_EL1;
```

When FEAT_SRMASK is implemented MSR TCR2ALIAS_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0111	0b111

```
    if !(IsFeatureImplemented(FEAT_TCR2) && IsFeatureImplemented(FEAT_AA64)) then
        UNDEFINED;
    elsif PSTATE.EL == EL0 then
        UNDEFINED;
    elsif PSTATE.EL == EL1 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TCR2En == '0' then
            UNDEFINED;
        elsif EL2Enabled() && HCR_EL2.TVM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
             ↪'0') || HFGWTR2_EL2.nTCR2ALIAS_EL1 == '0') then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.TCR2En == '0') then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.TCR2En == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EffectiveHCR_EL2_NVx() IN {'111'} then
            NVMem[0x270] = X[t, 64];
        else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        if IsFeatureImplemented(FEAT_SRMASK) then
            TCR2_EL1 = (X[t, 64] AND NOT EffectiveTCR2MASK_EL1()) OR (TCR2_EL1 AND
                ↪EffectiveTCR2MASK_EL1());
        else
            TCR2_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TCR2En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TCR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_SRMASK) then
            TCR2_EL2 = (X[t, 64] AND NOT EffectiveTCR2MASK_EL2()) OR (TCR2_EL2 AND
                ↪EffectiveTCR2MASK_EL2());
        else
            TCR2_EL2 = X[t, 64];
    else
        TCR2_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TCR2_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.191 TCR2_EL2, Extended Translation Control Register (EL2)

The TCR2_EL2 characteristics are:

Purpose

The control register for stage 1 of the EL2&0 translation regime.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_TCR2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TCR2_EL2 are UNDEFINED.

Attributes

TCR2_EL2 is a 64-bit register.

Field descriptions

When !ELIsInHost(EL2):

Unless stated otherwise, all the bits in TCR2_EL2 are permitted to be cached in a TLB.

Bits [63:13]

Reserved, RES0.

AMEC0, bit [12]

When FEAT_MEC is implemented:

This field controls the enabling of the Alternate MECID translations for the EL2 translation regime.

TCR2_EL2.AMEC0 is provided to enable the safe update of MECID_A0_EL2, by disabling access and speculation to AMEC==1 Block or Page descriptors during the update.

AMEC0	Meaning
0b0	Use of a Block or Page descriptor containing AMEC == 1 generates a Translation fault.
0b1	Accesses translated by a Block or Page descriptor containing AMEC == 1 are associated with the MECID configured in MECID_A0_EL2.

This bit is permitted to be cached in a TLB only if it is 1.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

When SCTLR2_EL2.EMEC is 0, this field is ignored by the PE and the bit position of AMEC is RES0 in Block and Page descriptors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When IsCurrentSecurityState(SS_Secure), access to this field is RES0.
- When IsCurrentSecurityState(SS_NonSecure), access to this field is RES0.

Otherwise:

Reserved, RES0.

HAFT, bit [11]

When FEAT_HAFT is implemented:

Hardware managed Access Flag for Table descriptors.

Enables the Hardware managed Access Flag for Table descriptors.

HAFT	Meaning
`0b0`	Hardware managed Access Flag for Table descriptors is disabled.
`0b1`	Hardware managed Access Flag for Table descriptors is enabled.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PTTWI, bit [10]

When FEAT_THE is implemented:

Permit Translation table walk Incoherence.

Permits RCWS instructions to generate writes that have the Reduced Coherence property.

PTTWI	Meaning
`0b0`	Write accesses generated by RCWS at EL2 or EL2&0 do not have the Reduced Coherence property.
`0b1`	Write accesses generated by RCWS at EL2 or EL2&0 have the Reduced Coherence property.

This bit is permitted to be implemented as a read-only bit with a fixed value of 0.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [9:5]

Reserved, RES0.

AIE, bit [4]

When FEAT_AIE is implemented:

Enable Attribute Indexing Extension.

AIE	Meaning
`0b0`	Attribute Indexing Extension Disabled.
`0b1`	Attribute Indexing Extension Enabled.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

POE, bit [3]

When FEAT_S1POE is implemented:

Enables Permission Overlay for EL2 accesses.

POE	Meaning
`0b0`	Permission overlay disabled for EL2 access in stage 1 of EL2 translation regime.
`0b1`	Permission overlay enabled for EL2 access in stage 1 of EL2 translation regime.

This bit is not permitted to be cached in a TLB.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bit [2]

Reserved, RES0.

PIE, bit [1]

When FEAT_S1PIE is implemented:

Enables usage of Indirect Permission Scheme.

PIE	Meaning
0b0	Direct permission model.
0b1	Indirect permission model.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PnCH, bit [0]

When FEAT_THE is implemented:

Protected attribute enable. Enables use of bit[52] of stage 1 translation table entries as the Protected bit, for translations using TTBR0_EL2 when HCR_EL2.E2H is 0.

PnCH	Meaning
0b0	For translations using TTBRn_EL2, bit[52] of each stage 1 translation table entry is not the Protected bit.
0b1	For translations using TTBR0_EL2, bit[52] of each stage 1 translation table entry is the Protected bit.

If bit[52] is used as the Protected bit, it is not used as the Contiguous bit.

This field is RES0 when TCR2_EL2.D128 is 1.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When ELIsInHost(EL2):

Unless stated otherwise, all the bits in TCR2_EL2 are permitted to be cached in a TLB.

Bits [63:19]

Reserved, RES0.

FNG1, bit [18]

When FEAT_ASID2 is implemented:

Force non-global translations for TTBR1_EL2.

FNG1	Meaning
0b0	This bit has no effect on the interpretation of the nG bit.
0b1	Translations are treated as non-global regardless of the value of the nG bit.

This bit is permitted to be cached in a TLB.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FNG0, bit [17]

When FEAT_ASID2 is implemented:

Force non-global translations for TTBR0_EL2.

FNG0	Meaning
0b0	This bit has no effect on the interpretation of the nG bit.
0b1	Translations are treated as non-global regardless of the value of the nG bit.

This bit is permitted to be cached in a TLB.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

A2, bit [16]

When FEAT_ASID2 is implemented:

Enable use of two ASIDs.

A2	Meaning
0b0	Use of two ASIDs is disabled.
0b1	Use of two ASIDs is enabled.

This bit is permitted to be cached in a TLB.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DisCH1, bit [15]

When FEAT_D128 is implemented and TCR2_EL2.D128 == '1':

Disable the Contiguous bit for the Start Table for TTBR1_EL2.

DisCH1	Meaning
0b0	The Contiguous bit of Block or Page descriptors of the Start Table for TTBR1_EL2 is not affected by this field.
0b1	The Contiguous bit of Block or Page descriptors of the Start Table for TTBR1_EL2 is treated as 0.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DisCH0, bit [14]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_D128 is implemented and TCR2_EL2.D128 == '1':

Disable the Contiguous bit for the Start Table for TTBR0_EL2.

DisCH0	Meaning
0b0	The Contiguous bit of Block or Page descriptors of the Start Table for TTBR0_EL2 is not affected by this field.
0b1	The Contiguous bit of Block or Page descriptors of the Start Table for TTBR0_EL2 is treated as 0.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AMEC1, bit [13]

When FEAT_MEC is implemented:

This field controls the enabling of the Alternate MECID translations for accesses in the TTBR1_EL2 half of the VA range, for the EL2&0 translation regime.

TCR2_EL2.AMEC1 is provided to enable the safe update of MECID_A1_EL2, by disabling access and speculation to AMEC == 1 Block or Page descriptors during the update.

AMEC1	Meaning
0b0	Use of a Block or Page descriptor containing AMEC == 1 generates a Translation fault.
0b1	Accesses translated by a Block or Page descriptor containing AMEC == 1 are associated with the MECID configured in MECID_A1_EL2.

This bit is permitted to be cached in a TLB only if it is 1.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

When SCTLR2_EL2.EMEC is 0, this field is ignored by the PE and the bit position of AMEC is RES0 in Block and Page descriptors.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When IsCurrentSecurityState(SS_Secure), access to this field is RES0.
- When IsCurrentSecurityState(SS_NonSecure), access to this field is RES0.

Otherwise:

Reserved, RES0.

AMEC0, bit [12]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_MEC is implemented:

This field controls the enabling of the Alternate MECID translations for accesses in the TTBR0_EL2 half of the VA range, for the EL2&0 translation regime.

TCR2_EL2.AMEC0 is provided to enable the safe update of MECID_A0_EL2, by disabling access and speculation to AMEC==1 Block or Page descriptors during the update.

AMEC0	Meaning
0b0	Use of a Block or Page descriptor containing AMEC == 1 generates a Translation fault.
0b1	Accesses translated by a Block or Page descriptor containing AMEC == 1 are associated with the MECID configured in MECID_A0_EL2.

This bit is permitted to be cached in a TLB only if it is 1.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

When SCTLR2_EL2.EMEC is 0, this field is ignored by the PE and the bit position of AMEC is RES0 in Block and Page descriptors.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When IsCurrentSecurityState(SS_Secure), access to this field is RES0.
- When IsCurrentSecurityState(SS_NonSecure), access to this field is RES0.

Otherwise:

Reserved, RES0.

HAFT, bit [11]

When FEAT_HAFT is implemented:

Hardware managed Access Flag for Table descriptors.

Enables the Hardware managed Access Flag for Table descriptors.

HAFT	Meaning
0b0	Hardware managed Access Flag for Table descriptors is disabled.
0b1	Hardware managed Access Flag for Table descriptors is enabled.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PTTWI, bit [10]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_THE is implemented:

Permit Translation table walk Incoherence.

Permits RCWS instructions to generate writes that have the Reduced Coherence property.

PTTWI	Meaning
0b0	Write accesses generated by RCWS do not have the Reduced Coherence property.
0b1	Write accesses generated by RCWS have the Reduced Coherence property.

This bit is permitted to be implemented as a read-only bit with a fixed value of 0.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [9:6]

Reserved, RES0.

D128, bit [5]

When FEAT_D128 is implemented:

Enables VMSAv9-128 translation system.

D128	Meaning
0b0	Translation system follows VMSAv8-64 translation process.
0b1	Translation system follows VMSAv9-128 translation process.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AIE, bit [4]

When FEAT_AIE is implemented:

Enable Attribute Indexing Extension.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

AIE	Meaning
0b0	Attribute Indexing Extension Disabled.
0b1	Attribute Indexing Extension Enabled.

This field is RES1 when TCR2_EL2.D128 is 1.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

POE, bit [3]

When FEAT_S1POE is implemented:

Enables Permission Overlay for privileged accesses from EL2&0 translation regime.

POE	Meaning
0b0	Permission overlay disabled for EL2 access in stage 1 of EL2&0 translation regime.
0b1	Permission overlay enabled for EL2 access in stage 1 of EL2&0 translation regime.

This bit is not permitted to be cached in a TLB.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0POE, bit [2]

When FEAT_S1POE is implemented:

Enables Permission Overlay for unprivileged accesses from EL2&0 translation regime.

E0POE	Meaning
0b0	Permission overlay disabled for EL0 access in stage 1 of EL2&0 translation regime.
0b1	Permission overlay enabled for EL0 access in stage 1 of EL2&0 translation regime.

This bit is not permitted to be cached in a TLB.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PIE, bit [1]

When FEAT_S1PIE is implemented:

Enables usage of Indirect Permission Scheme.

PIE	Meaning
`0b0`	Direct permission model.
`0b1`	Indirect permission model.

This field is RES1 when TCR2_EL2.D128 is 1.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PnCH, bit [0]

When FEAT_THE is implemented:

Protected attribute enable. Enables use of bit[52] of stage 1 translation table entries as the Protected bit, for translations using TTBRn_EL2 when HCR_EL2.E2H is 1.

PnCH	Meaning
`0b0`	For translations using TTBRn_EL2, bit[52] of each stage 1 translation table entry is not the Protected bit.
`0b1`	For translations using TTBR0_EL2, bit[52] of each stage 1 translation table entry is the Protected bit.

If bit[52] is used as the Protected bit, it is not used as the Contiguous bit.

This field is RES0 when TCR2_EL2.D128 is 1.

When EL3 is implemented and SCR_EL3.TCR2En == 0, this field is ignored by the PE and treated as zero.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TCR2_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name `TCR2_EL2` or `TCR2_EL1` are not guaranteed to be ordered with respect to accesses using the other accessor name.

If FEAT_SRMASK is implemented, accesses to TCR2_EL2 are masked by TCR2MASK_EL2.

Accesses to this register use the following encodings in the System register encoding space:

```
MSR TCR2_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_TCR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TCR2En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TCR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if IsFeatureImplemented(FEAT_SRMASK) then
            TCR2_EL2 = (X[t, 64] AND NOT EffectiveTCR2MASK_EL2()) OR (TCR2_EL2 AND
                ↪EffectiveTCR2MASK_EL2());
        else
            TCR2_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TCR2_EL2 = X[t, 64];
```

```
MRS <Xt>, TCR2_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b011

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_TCR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TCR2En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.TCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.TCR2En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TCR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x270];
    else
        X[t, 64] = TCR2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TCR2En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TCR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = TCR2_EL2;
    else
        X[t, 64] = TCR2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TCR2_EL1;
```

```
MSR TCR2_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_TCR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TCR2En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.TCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.TCR2En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TCR2En == '0' then
        if EL3SDDUndef() then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EffectiveHCR_EL2_NVx() IN {'111'} then
            NVMem[0x270] = X[t, 64];
        else
            if IsFeatureImplemented(FEAT_SRMASK) then
                TCR2_EL1 = (X[t, 64] AND NOT EffectiveTCR2MASK_EL1()) OR (TCR2_EL1 AND
                    ↪EffectiveTCR2MASK_EL1());
            else
                TCR2_EL1 = X[t, 64];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TCR2En == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.TCR2En == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif ELIsInHost(EL2) then
            if IsFeatureImplemented(FEAT_SRMASK) then
                TCR2_EL2 = (X[t, 64] AND NOT EffectiveTCR2MASK_EL2()) OR (TCR2_EL2 AND
                    ↪EffectiveTCR2MASK_EL2());
            else
                TCR2_EL2 = X[t, 64];
        else
            TCR2_EL1 = X[t, 64];
    elsif PSTATE.EL == EL3 then
        TCR2_EL1 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, TCR2_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0000	0b011

```
if !(IsFeatureImplemented(FEAT_TCR2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TCR2En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TCR2En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = TCR2_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TCR2_EL2;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.192 TCR2MASK_EL1, Extended Translation Control Masking Register (EL1)

The TCR2MASK_EL1 characteristics are:

Purpose

Mask register to prevent updates of fields in TCR2_EL1 on writes to TCR2_EL1 or TCR2ALIAS_EL1.

Configuration

This register is present only when FEAT_SRMASK is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TCR2MASK_EL1 are UNDEFINED.

Attributes

TCR2MASK_EL1 is a 64-bit register.

Field descriptions

Bits [63:22]

Reserved, RES0.

FNGNA1, bit [21]

When FEAT_THE is implemented:

Mask bit for FNGNA1.

FNGNA1	Meaning
0b0	TCR2_EL1.FNGNA1 is writeable.
0b1	TCR2_EL1.FNGNA1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FNGNA0, bit [20]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_THE is implemented:

Mask bit for FNGNA0.

FNGNA0	Meaning
0b0	TCR2_EL1.FNGNA0 is writeable.
0b1	TCR2_EL1.FNGNA0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [19]

Reserved, RES0.

FNG1, bit [18]

When FEAT_ASID2 is implemented:

Mask bit for FNG1.

FNG1	Meaning
0b0	TCR2_EL1.FNG1 is writeable.
0b1	TCR2_EL1.FNG1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FNG0, bit [17]

When FEAT_ASID2 is implemented:

Mask bit for FNG0.

FNG0	Meaning
0b0	TCR2_EL1.FNG0 is writeable.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FNG0	Meaning
0b1	TCR2_EL1.FNG0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

A2, bit [16]

When FEAT_ASID2 is implemented:

Mask bit for A2.

A2	Meaning
0b0	TCR2_EL1.A2 is writeable.
0b1	TCR2_EL1.A2 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DisCH1, bit [15]

When FEAT_D128 is implemented:

Mask bit for DisCH1.

DisCH1	Meaning
0b0	TCR2_EL1.DisCH1 is writeable.
0b1	TCR2_EL1.DisCH1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

DisCH0, bit [14]

When FEAT_D128 is implemented:

Mask bit for DisCH0.

DisCH0	Meaning
`0b0`	TCR2_EL1.DisCH0 is writeable.
`0b1`	TCR2_EL1.DisCH0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [13:12]

Reserved, RES0.

HAFT, bit [11]

When FEAT_HAFT is implemented:

Mask bit for HAFT.

HAFT	Meaning
`0b0`	TCR2_EL1.HAFT is writeable.
`0b1`	TCR2_EL1.HAFT is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PTTWI, bit [10]

When FEAT_THE is implemented:

Mask bit for PTTWI.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

PTTWI	Meaning
0b0	TCR2_EL1.PTTWI is writeable.
0b1	TCR2_EL1.PTTWI is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [9:6]

Reserved, RES0.

D128, bit [5]

When FEAT_D128 is implemented:

Mask bit for D128.

D128	Meaning
0b0	TCR2_EL1.D128 is writeable.
0b1	TCR2_EL1.D128 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AIE, bit [4]

When FEAT_AIE is implemented:

Mask bit for AIE.

AIE	Meaning
0b0	TCR2_EL1.AIE is writeable.
0b1	TCR2_EL1.AIE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

POE, bit [3]

When FEAT_S1POE is implemented:

Mask bit for POE.

POE	Meaning
0b0	TCR2_EL1.POE is writeable.
0b1	TCR2_EL1.POE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0POE, bit [2]

When FEAT_S1POE is implemented:

Mask bit for E0POE.

E0POE	Meaning
0b0	TCR2_EL1.E0POE is writeable.
0b1	TCR2_EL1.E0POE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PIE, bit [1]

When FEAT_S1PIE is implemented:

Mask bit for PIE.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

PIE	Meaning
0b0	TCR2_EL1.PIE is writeable.
0b1	TCR2_EL1.PIE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PnCH, bit [0]

When FEAT_THE is implemented:

Mask bit for PnCH.

PnCH	Meaning
0b0	TCR2_EL1.PnCH is writeable.
0b1	TCR2_EL1.PnCH is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TCR2MASK_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name TCR2MASK_EL1 or TCR2MASK_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TCR2MASK_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0111	0b011

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HFGRTR2_EL2.nTCR2MASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x338];
    else
        X[t, 64] = TCR2MASK_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = TCR2MASK_EL2;
    else
        X[t, 64] = TCR2MASK_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TCR2MASK_EL1;
```

```
MSR TCR2MASK_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0111	0b011

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HFGWTR2_EL2.nTCR2MASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x338] = X[t, 64];
    elsif !IsZero(EffectiveTCR2MASK_EL1()) then
        UNDEFINED;
    else
        TCR2MASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        if !IsZero(EffectiveTCR2MASK_EL2()) then
            UNDEFINED;
        else
            TCR2MASK_EL2 = X[t, 64];
    else
        TCR2MASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TCR2MASK_EL1 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, TCR2MASK_EL12

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0111	0b011

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x338];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = TCR2MASK_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = TCR2MASK_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented MSR TCR2MASK_EL12, <Xt>

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0111	0b011

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x338] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TCR2MASK_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        TCR2MASK_EL1 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.193 TCR2MASK_EL2, Extended Translation Control Masking Register (EL2)

The TCR2MASK_EL2 characteristics are:

Purpose

Mask register to prevent updates of fields in TCR2_EL2 on writes.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

This register is present only when FEAT_SRMASK is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TCR2MASK_EL2 are UNDEFINED.

Attributes

TCR2MASK_EL2 is a 64-bit register.

Field descriptions

When !ELIsInHost(EL2):

Bits [63:13]

Reserved, RES0.

AMEC0, bit [12]

When FEAT_MEC is implemented:

Mask bit for AMEC0.

AMEC0	Meaning
0b0	TCR2_EL2.AMEC0 is writeable.
0b1	TCR2_EL2.AMEC0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

HAFT, bit [11]

When FEAT_HAFT is implemented:

Mask bit for HAFT.

HAFT	Meaning
0b0	TCR2_EL2.HAFT is writeable.
0b1	TCR2_EL2.HAFT is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PTTWI, bit [10]

When FEAT_THE is implemented:

Mask bit for PTTWI.

PTTWI	Meaning
0b0	TCR2_EL2.PTTWI is writeable.
0b1	TCR2_EL2.PTTWI is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [9:5]

Reserved, RES0.

AIE, bit [4]

When FEAT_AIE is implemented:

Mask bit for AIE.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

AIE	Meaning
0b0	TCR2_EL2.AIE is writeable.
0b1	TCR2_EL2.AIE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

POE, bit [3]

When FEAT_S1POE is implemented:

Mask bit for POE.

POE	Meaning
0b0	TCR2_EL2.POE is writeable.
0b1	TCR2_EL2.POE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [2]

Reserved, RES0.

PIE, bit [1]

When FEAT_S1PIE is implemented:

Mask bit for PIE.

PIE	Meaning
0b0	TCR2_EL2.PIE is writeable.
0b1	TCR2_EL2.PIE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PnCH, bit [0]

When FEAT_THE is implemented:

Mask bit for PnCH.

PnCH	Meaning
0b0	TCR2_EL2.PnCH is writeable.
0b1	TCR2_EL2.PnCH is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

When ELIsInHost(EL2):

Bits [63:19]

Reserved, RES0.

FNG1, bit [18]

When FEAT_ASID2 is implemented:

Mask bit for FNG1.

FNG1	Meaning
0b0	TCR2_EL2.FNG1 is writeable.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

0b1	TCR2_EL2.FNG1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FNG0, bit [17]

When FEAT_ASID2 is implemented:

Mask bit for FNG0.

FNG0	Meaning
0b0	TCR2_EL2.FNG0 is writeable.
0b1	TCR2_EL2.FNG0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

A2, bit [16]

When FEAT_ASID2 is implemented:

Mask bit for A2.

A2	Meaning
0b0	TCR2_EL2.A2 is writeable.
0b1	TCR2_EL2.A2 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DisCH1, bit [15]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_D128 is implemented:

Mask bit for DisCH1.

DisCH1	Meaning
0b0	TCR2_EL2.DisCH1 is writeable.
0b1	TCR2_EL2.DisCH1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DisCH0, bit [14]

When FEAT_D128 is implemented:

Mask bit for DisCH0.

DisCH0	Meaning
0b0	TCR2_EL2.DisCH0 is writeable.
0b1	TCR2_EL2.DisCH0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AMEC1, bit [13]

When FEAT_MEC is implemented:

Mask bit for AMEC1.

AMEC1	Meaning
0b0	TCR2_EL2.AMEC1 is writeable.
0b1	TCR2_EL2.AMEC1 is not writeable.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AMEC0, bit [12]

When FEAT_MEC is implemented:

Mask bit for AMEC0.

AMEC0	Meaning
`0b0`	TCR2_EL2.AMEC0 is writeable.
`0b1`	TCR2_EL2.AMEC0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HAFT, bit [11]

When FEAT_HAFT is implemented:

Mask bit for HAFT.

HAFT	Meaning
`0b0`	TCR2_EL2.HAFT is writeable.
`0b1`	TCR2_EL2.HAFT is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PTTWI, bit [10]

When FEAT_THE is implemented:

Mask bit for PTTWI.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

PTTWI	Meaning
0b0	TCR2_EL2.PTTWI is writeable.
0b1	TCR2_EL2.PTTWI is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [9:6]

Reserved, RES0.

D128, bit [5]

When FEAT_D128 is implemented:

Mask bit for D128.

D128	Meaning
0b0	TCR2_EL2.D128 is writeable.
0b1	TCR2_EL2.D128 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AIE, bit [4]

When FEAT_AIE is implemented:

Mask bit for AIE.

AIE	Meaning
0b0	TCR2_EL2.AIE is writeable.
0b1	TCR2_EL2.AIE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

POE, bit [3]

When FEAT_S1POE is implemented:

Mask bit for POE.

POE	Meaning
0b0	TCR2_EL2.POE is writeable.
0b1	TCR2_EL2.POE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0POE, bit [2]

When FEAT_S1POE is implemented:

Mask bit for E0POE.

E0POE	Meaning
0b0	TCR2_EL2.E0POE is writeable.
0b1	TCR2_EL2.E0POE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PIE, bit [1]

When FEAT_S1PIE is implemented:

Mask bit for PIE.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

PIE	Meaning
0b0	TCR2_EL2.PIE is writeable.
0b1	TCR2_EL2.PIE is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PnCH, bit [0]

When FEAT_THE is implemented:

Mask bit for PnCH.

PnCH	Meaning
0b0	TCR2_EL2.PnCH is writeable.
0b1	TCR2_EL2.PnCH is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TCR2MASK_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name TCR2MASK_EL2 or TCR2MASK_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TCR2MASK_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0111	0b011

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = TCR2MASK_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TCR2MASK_EL2;
```

MSR TCR2MASK_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0111	0b011

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsZero(EffectiveTCR2MASK_EL2()) then
        UNDEFINED;
    else
        TCR2MASK_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TCR2MASK_EL2 = X[t, 64];
```

MRS <Xt>, TCR2MASK_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0111	0b011

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HFGRTR2_EL2.nTCR2MASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x338];
    else
        X[t, 64] = TCR2MASK_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = TCR2MASK_EL2;
    else
        X[t, 64] = TCR2MASK_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TCR2MASK_EL1;
```

```
MSR TCR2MASK_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0111	0b011

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HFGWTR2_EL2.nTCR2MASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x338] = X[t, 64];
    elsif !IsZero(EffectiveTCR2MASK_EL1()) then
        UNDEFINED;
    else
        TCR2MASK_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        if !IsZero(EffectiveTCR2MASK_EL2()) then
            UNDEFINED;
        else
            TCR2MASK_EL2 = X[t, 64];
    else
        TCR2MASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TCR2MASK_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.194 TCR_EL1, Translation Control Register (EL1)

The TCR_EL1 characteristics are:

Purpose

The control register for stage 1 of the EL1&0 translation regime.

Configuration

AArch64 System register TCR_EL1 bits [31:0] are architecturally mapped to AArch32 System register TTBCR[31:0].

AArch64 System register TCR_EL1 bits [63:32] are architecturally mapped to AArch32 System register TTBCR2[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TCR_EL1 are UNDEFINED.

Attributes

TCR_EL1 is a 64-bit register.

Field descriptions

Any of the bits in TCR_EL1, other than the EPDx bits when they have the value 1, and the A1 bit are permitted to be cached in a TLB.

Bits [63:62]

Reserved, RES0.

MTX1, bit [61]

When FEAT_MTE_NO_ADDRESS_TAGS is implemented or FEAT_MTE_CANONICAL_TAGS is implemented:

Extended memory tag checking.

This field controls address generation and tag checking when EL0 and EL1 are using AArch64 where the data address would be translated by tables pointed to by TTBR1_EL1.

This control has an effect regardless of whether stage 1 of the EL1&0 translation regime is enabled or not.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

MTX1	Meaning
0b0	This control has no effect on the PE.
0b1	Bits[59:56] of a 64-bit VA hold a Logical Address Tag, and all of the following apply: • Bits[59:56] are treated as 0b1111 when checking if the address is out of range. • If FEAT_PAuth is implemented, bits[59:56] are not part of the PAC field. • A Canonical Tag Check operation is performed on Tag Checked memory accesses to a Canonically Tagged memory location.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MTX0, bit [60]

When FEAT_MTE_NO_ADDRESS_TAGS is implemented or FEAT_MTE_CANONICAL_TAGS is implemented:

Extended memory tag checking.

This field controls address generation and tag checking when EL0 and EL1 are using AArch64 where the data address would be translated by tables pointed to by TTBR0_EL1.

This control has an effect regardless of whether stage 1 of the EL1&0 translation regime is enabled or not.

MTX0	Meaning
0b0	This control has no effect on the PE.
0b1	Bits[59:56] of a 64-bit VA hold a Logical Address Tag, and all of the following apply: • Bits[59:56] are treated as 0b0000 when checking if the address is out of range. • If FEAT_PAuth is implemented, bits[59:56] are not part of the PAC field. • A Canonical Tag Check operation is performed on Tag Checked memory accesses to a Canonically Tagged memory location.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DS, bit [59]

When FEAT_LPA2 is implemented and (FEAT_D128 is not implemented or TCR2_EL1.D128 == '0'):

This field affects:

- Whether a 52-bit output address can be described by the translation tables of the 4KB or 16KB translation granules.
- The minimum value of TCR_EL1.{T0SZ,T1SZ}.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- How and where shareability for Block and Page descriptors are encoded.

DS	Meaning
0b0	Bits[49:48] of translation descriptors are RES0. Bits[9:8] in Block and Page descriptors encode shareability information in the SH[1:0] field. Bits[9:8] in Table descriptors are ignored by hardware. The minimum value of the TCR_EL1.{T0SZ, T1SZ} fields is 16. Any memory access using a smaller value generates a stage 1 level 0 translation table fault. Output address[51:48] is 0b0000.
0b1	Bits[49:48] of translation descriptors hold output address[49:48]. Bits[9:8] of Translation table descriptors hold output address[51:50]. The shareability information of Block and Page descriptors for cacheable locations is determined by: • TCR_EL1.SH0 if the VA is translated using tables pointed to by TTBR0_EL1. • TCR_EL1.SH1 if the VA is translated using tables pointed to by TTBR1_EL1. The minimum value of the TCR_EL1.{T0SZ, T1SZ} fields is 12. Any memory access using a smaller value generates a stage 1 level 0 translation table fault. All calculations of the stage 1 base address are modified for tables of fewer than 8 entries so that the table is aligned to 64 bytes. Bits[5:2] of TTBR0_EL1 or TTBR1_EL1 are used to hold bits[51:48] of the output address in all cases. **Note** As FEAT_LVA must be implemented if TCR_EL1.DS == 1, the minimum value of the TCR_EL1.{T0SZ, T1SZ} fields is 12, as determined by that extension. For the TLBI Range instructions affecting VA, the format of the argument is changed so that bits[36:0] hold BaseADDR[52:16]. For the 4KB translation granule, bits[15:12] of BaseADDR are treated as 0b0000. For the 16KB translation granule, bits[15:14] of BaseADDR are treated as 0b00. **Note** This forces alignment of the ranges used by the TLBI range instructions.

This field is RES0 for a 64KB translation granule.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

The Effective value of this bit is 0b0.

Access to this field is RES0.

TCMA1, bit [58]

When FEAT_MTE2 is implemented:

Controls the generation of Unchecked accesses at EL1, and at EL0 if the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, when address[59:55] = 0b11111.

TCMA1	Meaning
0b0	This control has no effect on the generation of Unchecked accesses at EL1 or EL0.
0b1	All accesses at EL1 and EL0 are Unchecked.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TCMA1	Meaning

Note

Software may change this control bit on a context switch.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCMA0, bit [57]

When FEAT_MTE2 is implemented:

Controls the generation of Unchecked accesses at EL1, and at EL0 if the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, when address[59:55] = `0b00000`.

TCMA0	Meaning
`0b0`	This control has no effect on the generation of Unchecked accesses at EL1 or EL0.
`0b1`	All accesses at EL1 and EL0 are Unchecked.

Note

Software may change this control bit on a context switch.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0PD1, bit [56]

When FEAT_E0PD is implemented:

Faulting control for unprivileged access to any address translated by TTBR1_EL1.

E0PD1	Meaning
`0b0`	Unprivileged access to any address translated by TTBR1_EL1 will not generate a fault by this mechanism.
`0b1`	Unprivileged access to any address translated by TTBR1_EL1 will generate a level 0 Translation fault.

Level 0 Translation faults generated as a result of this field are not counted as TLB misses for performance monitoring. The fault should take the same time to generate, whether the address is present in the TLB or not, to mitigate attacks that use fault timing.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0PD0, bit [55]

When FEAT_E0PD is implemented:

Faulting control for unprivileged access to any address translated by TTBR0_EL1.

E0PD0	Meaning
0b0	Unprivileged access to any address translated by TTBR0_EL1 will not generate a fault by this mechanism.
0b1	Unprivileged access to any address translated by TTBR0_EL1 will generate a level 0 Translation fault.

Level 0 Translation faults generated as a result of this field are not counted as TLB misses for performance monitoring. The fault should take the same time to generate, whether the address is present in the TLB or not, to mitigate attacks that use fault timing.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NFD1, bit [54]

When FEAT_SVE is implemented:

Non-Fault translation timing Disable when using TTBR1_EL1.

Controls how a TLB miss is reported in response to a non-fault unprivileged access for a virtual address that is translated using TTBR1_EL1.

If SVE is implemented, the affected access types include:

- All accesses due to an SVE non-fault contiguous load instruction.
- Accesses due to an SVE first-fault gather load instruction that are not for the First active element. Accesses due to an SVE first-fault contiguous load instruction are not affected.
- Accesses due to prefetch instructions might be affected, but the effect is not architecturally visible.

NFD1	Meaning
0b0	Does not affect the handling of a TLB miss on accesses translated using TTBR1_EL1.
0b1	A TLB miss on a virtual address that is translated using TTBR1_EL1 due to the specified access types causes the access to fail without taking an exception. The amount of time that the failure takes to be handled should not predictively leak whether it was caused by a TLB miss or a Permission fault, to mitigate attacks that use fault timing.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

NFD0, bit [53]

When FEAT_SVE is implemented:

Non-Fault translation timing Disable when using TTBR0_EL1.

Controls how a TLB miss is reported in response to a non-fault unprivileged access for a virtual address that is translated using TTBR0_EL1.

If SVE is implemented, the affected access types include:

- All accesses due to an SVE non-fault contiguous load instruction.
- Accesses due to an SVE first-fault gather load instruction that are not for the First active element. Accesses due to an SVE first-fault contiguous load instruction are not affected.
- Accesses due to prefetch instructions might be affected, but the effect is not architecturally visible.

NFD0	Meaning
0b0	Does not affect the handling of a TLB miss on accesses translated using TTBR0_EL1.
0b1	A TLB miss on a virtual address that is translated using TTBR0_EL1 due to the specified access types causes the access to fail without taking an exception. The amount of time that the failure takes to be handled should not predictively leak whether it was caused by a TLB miss or a Permission fault, to mitigate attacks that use fault timing.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBID1, bit [52]

When FEAT_PAuth is implemented:

Controls the use of the top byte of instruction addresses for address matching.

For the purpose of this field, all cache maintenance and address translation instructions that perform address translation are treated as data accesses.

For more information, see ‘Address tagging’.

TBID1	Meaning
0b0	TCR_EL1.TBI1 applies to Instruction and Data accesses.
0b1	TCR_EL1.TBI1 applies to Data accesses only.

This affects addresses where the address would be translated by tables pointed to by TTBR1_EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

TBID0, bit [51]

When FEAT_PAuth is implemented:

Controls the use of the top byte of instruction addresses for address matching.

For the purpose of this field, all cache maintenance and address translation instructions that perform address translation are treated as data accesses.

For more information, see 'Address tagging'.

TBID0	Meaning
0b0	TCR_EL1.TBI0 applies to Instruction and Data accesses.
0b1	TCR_EL1.TBI0 applies to Data accesses only.

This affects addresses where the address would be translated by tables pointed to by TTBR0_EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU162, bit [50]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry for translations using TTBR1_EL1.

HWU162	Meaning
0b0	For translations using TTBR1_EL1, bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR1_EL1, bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL1.HPD1 is 1.

The Effective value of this field is 0 if the value of TCR_EL1.HPD1 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU161, bit [49]

When FEAT_HPDS2 is implemented:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry for translations using TTBR1_EL1.

HWU161	Meaning
0b0	For translations using TTBR1_EL1, bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR1_EL1, bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL1.HPD1 is 1.

The Effective value of this field is 0 if the value of TCR_EL1.HPD1 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU160, bit [48]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry for translations using TTBR1_EL1.

HWU160	Meaning
0b0	For translations using TTBR1_EL1, bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR1_EL1, bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL1.HPD1 is 1.

The Effective value of this field is 0 if the value of TCR_EL1.HPD1 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU159, bit [47]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry for translations using TTBR1_EL1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

HWU159	Meaning
0b0	For translations using TTBR1_EL1, bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR1_EL1, bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL1.HPD1 is 1.

The Effective value of this field is 0 if the value of TCR_EL1.HPD1 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU062, bit [46]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry for translations using TTBR0_EL1.

HWU062	Meaning
0b0	For translations using TTBR0_EL1, bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR0_EL1, bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL1.HPD0 is 1.

The Effective value of this field is 0 if the value of TCR_EL1.HPD0 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU061, bit [45]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry for translations using TTBR0_EL1.

HWU061	Meaning
0b0	For translations using TTBR0_EL1, bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

HWU061	Meaning
0b1	For translations using TTBR0_EL1, bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL1.HPD0 is 1.

The Effective value of this field is 0 if the value of TCR_EL1.HPD0 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU060, bit [44]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry for translations using TTBR0_EL1.

HWU060	Meaning
0b0	For translations using TTBR0_EL1, bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR0_EL1, bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL1.HPD0 is 1.

The Effective value of this field is 0 if the value of TCR_EL1.HPD0 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU059, bit [43]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry for translations using TTBR0_EL1.

HWU059	Meaning
0b0	For translations using TTBR0_EL1, bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR0_EL1, bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL1.HPD0 is 1.

The Effective value of this field is 0 if the value of TCR_EL1.HPD0 is 0.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD1, bit [42]

When FEAT_HPDS is implemented:

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, PXNTable, and UXNTable, except NSTable, in the translation tables pointed to by TTBR1_EL1.

HPD1	Meaning
0b0	Hierarchical permissions are enabled.
0b1	Hierarchical permissions are disabled.

When disabled, the permissions are treated as if the bits are zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD0, bit [41]

When FEAT_HPDS is implemented:

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, PXNTable, and UXNTable, except NSTable, in the translation tables pointed to by TTBR0_EL1.

HPD0	Meaning
0b0	Hierarchical permissions are enabled.
0b1	Hierarchical permissions are disabled.

When disabled, the permissions are treated as if the bits are zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HD, bit [40]

When FEAT_HAFDBS is implemented:

Hardware management of dirty state in stage 1 translations from EL0 and EL1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

HD	Meaning
0b0	Stage 1 hardware management of dirty state disabled.
0b1	Stage 1 hardware management of dirty state enabled.

When the Effective value of TCR_EL1.HA is 0, this field behaves as 0 for all purposes other than a direct read of the value of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HA, bit [39]

When FEAT_HAFDBS is implemented:

Hardware Access flag update in stage 1 translations from EL0 and EL1.

HA	Meaning
0b0	Stage 1 Access flag update disabled.
0b1	Stage 1 Access flag update enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBI1, bit [38]

Top Byte ignored. Indicates whether the top byte of an address is used for address match for the TTBR1_EL1 region, or ignored and used for tagged addresses.

TBI1	Meaning
0b0	Top Byte used in the address calculation.
0b1	Top Byte ignored in the address calculation.

This affects addresses generated in EL0 and EL1 using AArch64 where the address would be translated by tables pointed to by TTBR1_EL1. It has an effect whether the EL1&0 translation regime is enabled or not.

If FEAT_PAuth is implemented and TCR_EL1.TBID1 is 1, then this field only applies to Data accesses.

Otherwise, if the value of TBI1 is 1 and bit [55] of the target address to be stored to the PC is 1, then bits[63:56] of that target address are also set to 1 before the address is stored in the PC, in the following cases:

- A branch or procedure return within EL0 or EL1.
- An exception taken to EL1.
- An exception return to EL0 or EL1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TBI0, bit [37]

Top Byte ignored. Indicates whether the top byte of an address is used for address match for the TTBR0_EL1 region, or ignored and used for tagged addresses.

TBI0	Meaning
0b0	Top Byte used in the address calculation.
0b1	Top Byte ignored in the address calculation.

This affects addresses generated in EL0 and EL1 using AArch64 where the address would be translated by tables pointed to by TTBR0_EL1. It has an effect whether the EL1&0 translation regime is enabled or not.

If FEAT_PAuth is implemented and TCR_EL1.TBID0 is 1, then this field only applies to Data accesses.

Otherwise, if the value of TBI0 is 1 and bit [55] of the target address to be stored to the PC is 0, then bits[63:56] of that target address are also set to 0 before the address is stored in the PC, in the following cases:

- A branch or procedure return within EL0 or EL1.
- An exception taken to EL1.
- An exception return to EL0 or EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

AS, bit [36]

ASID Size.

AS	Meaning
0b0	8 bit - the upper 8 bits of TTBR0_EL1 and TTBR1_EL1 are ignored by hardware for every purpose except reading back the register, and are treated as if they are all zeros for when used for allocation and matching entries in the TLB.
0b1	16 bit - the upper 16 bits of TTBR0_EL1 and TTBR1_EL1 are used for allocation and matching in the TLB.

If the implementation has only 8 bits of ASID, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [35]

Reserved, RES0.

IPS, bits [34:32]

Intermediate Physical Address Size.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

IPS	Meaning
0b000	32 bits, 4GB.
0b001	36 bits, 64GB.
0b010	40 bits, 1TB.
0b011	42 bits, 4TB.
0b100	44 bits, 16TB.
0b101	48 bits, 256TB.
0b110	52 bits, 4PB.
0b111	56 bits, 64PB.

The values 0b110 and 0b111 represent different output address sizes depending on implementation choices and translation configuration.

The following table captures the output address size represented by the value 0b110:

Descriptor Format	ID_AA64MMFR0_EL1.PARange	Translation Granule	DS[1] | Represented OA size |
Any	0b0101	Any	Any 48 bits, 256TB
VMSAv8-64	0b011x	4KB or 16KB	0 48 bits, 256TB
VMSAv8-64	0b011x	4KB or 16KB	1 52 bits, 4PB
VMSAv8-64	0b011x	64KB	N/A 52 bits, 4PB
VMSAv9-128	0b011x	Any	N/A 52 bits, 4PB

[1] This column represents the value of TCR_EL1.DS.

The following table captures the output address size represented by the value 0b111:

Descriptor Format	ID_AA64MMFR0_EL1.PARange	Translation Granule	Represented OA size
Any	0b0110	Any	OA size represented by 0b110
VMSAv8-64	0b0111	Any	OA size represented by 0b110
VMSAv9-128	0b0111	Any	56 bits, 64PB

If 52-bit PA is supported, and the translation table descriptors cannot express an OA larger than 48-bits, then bits[51:48] of every translation table base address are treated as 0b0000 for the stage of translation controlled by TCR_EL1.

If 56-bit PA is supported, and the translation table descriptors cannot express an OA larger than 52-bits, then bits[55:52] of every translation table base address are treated as 0b0000 for the stage of translation controlled by TCR_EL1.

If the output address size represented by this field is larger than the supported PA size expressed in ID_AA64MMFR0_EL1.PARange, then the output address size is treated as being the same as the supported PA size. Arm strongly recommends that software avoids configuring this field to a value representing an output address size larger than the supported PA size.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TG1, bits [31:30]

Granule size for the TTBR1_EL1.

TG1	Meaning
0b01	16KB.
0b10	4KB.
0b11	64KB.

Other values are reserved.

If the value is programmed to either a reserved value or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH1, bits [29:28]

Shareability attribute for memory associated with translation table walks using TTBR1_EL1.

SH1	Meaning
0b00	Non-shareable.
0b10	Outer Shareable.
0b11	Inner Shareable.

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ORGN1, bits [27:26]

Outer cacheability attribute for memory associated with translation table walks using TTBR1_EL1.

ORGN1	Meaning
0b00	Normal memory, Outer Non-cacheable.
0b01	Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRGN1, bits [25:24]

Inner cacheability attribute for memory associated with translation table walks using TTBR1_EL1.

IRGN1	Meaning
0b00	Normal memory, Inner Non-cacheable.
0b01	Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EPD1, bit [23]

Translation table walk disable for translations using TTBR1_EL1. This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using TTBR1_EL1. The encoding of this bit is:

EPD1	Meaning
0b0	Perform translation table walks using TTBR1_EL1.
0b1	A TLB miss on an address that is translated using TTBR1_EL1 generates a Translation fault. No translation table walk is performed.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A1, bit [22]

Selects whether TTBR0_EL1 or TTBR1_EL1 defines the ASID. The encoding of this bit is:

A1	Meaning
0b0	TTBR0_EL1.ASID defines the ASID.
0b1	TTBR1_EL1.ASID defines the ASID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T1SZ, bits [21:16]

The size offset of the memory region addressed by TTBR1_EL1. The region size is $2^{(64-T1SZ)}$ bytes.

The maximum and minimum possible values for T1SZ depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Note

For the 4KB translation granule, if FEAT_LPA2 is implemented, TCR_EL1.DS is 1, and this field is less than 16, the translation table walk begins with a level -1 initial lookup.
For the 16KB translation granule, if FEAT_LPA2 is implemented, TCR_EL1.DS is 1, and this field is less than 17, the translation table walk begins with a level 0 initial lookup.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TG0, bits [15:14]

Granule size for the TTBR0_EL1.

TG0	Meaning
0b00	4KB
0b01	64KB
0b10	16KB

Other values are reserved.

If the value is programmed to either a reserved value or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH0, bits [13:12]

Shareability attribute for memory associated with translation table walks using TTBR0_EL1.

SH0	Meaning
0b00	Non-shareable
0b10	Outer Shareable
0b11	Inner Shareable

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ORGN0, bits [11:10]

Outer cacheability attribute for memory associated with translation table walks using TTBR0_EL1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ORGN0	Meaning
0b00	Normal memory, Outer Non-cacheable.
0b01	Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRGN0, bits [9:8]

Inner cacheability attribute for memory associated with translation table walks using TTBR0_EL1.

IRGN0	Meaning
0b00	Normal memory, Inner Non-cacheable.
0b01	Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EPD0, bit [7]

Translation table walk disable for translations using TTBR0_EL1. This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using TTBR0_EL1. The encoding of this bit is:

EPD0	Meaning
0b0	Perform translation table walks using TTBR0_EL1.
0b1	A TLB miss on an address that is translated using TTBR0_EL1 generates a Translation fault. No translation table walk is performed.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [6]

Reserved, RES0.

T0SZ, bits [5:0]

The size offset of the memory region addressed by TTBR0_EL1. The region size is $2^{(64-T0SZ)}$ bytes.

The maximum and minimum possible values for T0SZ depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Note

For the 4KB translation granule, if FEAT_LPA2 is implemented, TCR_EL1.DS is 1, and this field is less than 16, the translation table walk begins with a level -1 initial lookup.
For the 16KB translation granule, if FEAT_LPA2 is implemented, TCR_EL1.DS is 1, and this field is less than 17, the translation table walk begins with a level 0 initial lookup.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TCR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name `TCR_EL1` or `TCR_EL12` are not guaranteed to be ordered with respect to accesses using the other accessor name.

If FEAT_SRMASK is implemented, accesses to TCR_EL1 are masked by TCRMASK_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TCR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.TCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x120];
    else
        X[t, 64] = TCR_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = TCR_EL2;
    else
        X[t, 64] = TCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TCR_EL1;
```

```
MSR TCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b010

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.TCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x120] = X[t, 64];
    else
        if IsFeatureImplemented(FEAT_SRMASK) then
            TCR_EL1 = (X[t, 64] AND NOT EffectiveTCRMASK_EL1()) OR (TCR_EL1 AND EffectiveTCRMASK_EL1());
        else
            TCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_SRMASK) then
            TCR_EL2 = (X[t, 64] AND NOT EffectiveTCRMASK_EL2()) OR (TCR_EL2 AND EffectiveTCRMASK_EL2());
        else
            TCR_EL2 = X[t, 64];
    else
        TCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TCR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, TCR_EL12

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x120];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = TCR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = TCR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented MSR TCR_EL12, <Xt>

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x120] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        TCR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        TCR_EL1 = X[t, 64];
    else
        UNDEFINED;
```

When FEAT_SRMASK is implemented MRS <Xt>, TCRALIAS_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0111	0b110

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
        ↪'0') || HFGRTR2_EL2.nTCRALIAS_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x120];
    else
        X[t, 64] = TCR_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = TCR_EL2;
    else
        X[t, 64] = TCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TCR_EL1;
```

When FEAT_SRMASK is implemented MSR TCRALIAS_EL1, <Xt>

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0111	0b110

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 ==
         ↪'0') || HFGWTR2_EL2.nTCRALIAS_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x120] = X[t, 64];
    else
        if IsFeatureImplemented(FEAT_SRMASK) then
            TCR_EL1 = (X[t, 64] AND NOT EffectiveTCRMASK_EL1()) OR (TCR_EL1 AND
                 ↪EffectiveTCRMASK_EL1());
        else
            TCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_SRMASK) then
            TCR_EL2 = (X[t, 64] AND NOT EffectiveTCRMASK_EL2()) OR (TCR_EL2 AND
                 ↪EffectiveTCRMASK_EL2());
        else
            TCR_EL2 = X[t, 64];
    else
        TCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TCR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.195 TCR_EL2, Translation Control Register (EL2)

The TCR_EL2 characteristics are:

Purpose

The control register for stage 1 of the EL2, or EL2&0, translation regime:

- When the Effective value of HCR_EL2.E2H is not 1, this register controls stage 1 of the EL2 translation regime, that supports a single VA range, translated using TTBR0_EL2.
- When the Effective value of HCR_EL2.E2H is 1, this register controls stage 1 of the EL2&0 translation regime, that supports both:
  - A lower VA range, translated using TTBR0_EL2.
  - A higher VA range, translated using TTBR1_EL2.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

AArch64 System register TCR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HTCR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TCR_EL2 are UNDEFINED.

Attributes

TCR_EL2 is a 64-bit register.

Field descriptions

When !ELIsInHost(EL2):

Any of the bits in TCR_EL2 are permitted to be cached in a TLB.

Bits [63:34]

Reserved, RES0.

MTX, bit [33]

When FEAT_MTE_NO_ADDRESS_TAGS is implemented or FEAT_MTE_CANONICAL_TAGS is implemented:

Extended memory tag checking.

This field controls address generation and tag checking when EL2 is using AArch64 where the data address would be translated by tables pointed to by TTBR0_EL2.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

This control has an effect regardless of whether stage 1 of the EL2 translation regime is enabled or not.

MTX	Meaning
`0b0`	This control has no effect on the PE.
`0b1`	Bits[59:56] of a 64-bit VA hold a Logical Address Tag, and all of the following apply: • Bits[59:56] are treated as `0b0000` when checking if the address is out of range. • If FEAT_PAuth is implemented, bits[59:56] are not part of the PAC field. • A Canonical Tag Check operation is performed on Tag Checked memory accesses to a Canonically Tagged memory location.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DS, bit [32]

When FEAT_LPA2 is implemented:

This field affects:

- Whether a 52-bit output address can be described by the translation tables of the 4KB or 16KB translation granules.
- The minimum value of TCR_EL2.T0SZ.
- How and where shareability for Block and Page descriptors are encoded.

DS	Meaning
`0b0`	Bits[49:48] of translation descriptors are RES0. Bits[9:8] in Block and Page descriptors encode shareability information in the SH[1:0] field. Bits[9:8] in table descriptors are ignored by hardware. The minimum value of TCR_EL2.T0SZ is 16. Any memory access using a smaller value generates a stage 1 level 0 translation table fault. Output address[51:48] is `0b0000`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

DS	Meaning
0b1	Bits[49:48] of translation descriptors hold output address[49:48]. Bits[9:8] of Translation table descriptors hold output address[51:50]. The shareability information of Block and Page descriptors for cacheable locations is determined by TCR_EL2.SH0. The minimum value of TCR_EL2.T0SZ is 12. Any memory access using a smaller value generates a stage 1 level 0 translation table fault. All calculations of the stage 1 base address are modified for tables of fewer than 8 entries so that the table is aligned to 64 bytes. Bits[5:2] of TTBR0_EL2 are used to hold bits[51:48] of the output address in all cases. **Note** As FEAT_LVA must be implemented if TCR_EL2.DS == 1, the minimum value of the TCR_EL2.T0SZ field is 12, as determined by that extension. For the TLBI Range instructions affecting VA, the format of the argument is changed so that bits[36:0] hold BaseADDR[52:16]. For the 4KB translation granule, bits[15:12] of BaseADDR are treated as 0b0000. For the 16KB translation granule, bits[15:14] of BaseADDR are treated as 0b00. **Note** This forces alignment of the ranges used by the TLBI range instructions.

This field is RES0 for a 64KB translation granule.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

The Effective value of this bit is 0b0.

Access to this field is RES0.

Bit [31]

Reserved, RES1.

TCMA, bit [30]

When FEAT_MTE2 is implemented:

Controls the generation of Unchecked accesses at EL2 when address [59:56] = 0b0000.

TCMA	Meaning
0b0	This control has no effect on the generation of Unchecked accesses.
0b1	All accesses are Unchecked.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBID, bit [29]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_PAuth is implemented:

Controls the use of the top byte of instruction addresses for address matching.

For the purpose of this field, all cache maintenance and address translation instructions that perform address translation are treated as data accesses.

For more information, see 'Address tagging'.

TBID	Meaning
0b0	TCR_EL2.TBI applies to Instruction and Data accesses.
0b1	TCR_EL2.TBI applies to Data accesses only.

This affects addresses where the address would be translated by tables pointed to by TTBR0_EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU62, bit [28]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry.

HWU62	Meaning
0b0	Bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU61, bit [27]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

HWU61	Meaning
0b0	Bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU60, bit [26]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry.

HWU60	Meaning
0b0	Bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU59, bit [25]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry.

HWU59	Meaning
0b0	Bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD is 0.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD, bit [24]

When FEAT_HPDS is implemented:

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, PXNTable, and UXNTable, except NSTable, in the translation tables pointed to by TTBR0_EL2.

HPD	Meaning
0b0	Hierarchical permissions are enabled.
0b1	Hierarchical permissions are disabled. **Note** In this case, bit[61] (APTable[0]) and bit[59] (PXNTable) of the next level descriptor attributes are required to be ignored by the PE and are no longer reserved, allowing them to be used by software.

When disabled, the permissions are treated as if the bits are zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [23]

Reserved, RES1.

HD, bit [22]

When FEAT_HAFDBS is implemented:

Hardware management of dirty state in stage 1 translations from EL2.

HD	Meaning
0b0	Stage 1 hardware management of dirty state disabled.
0b1	Stage 1 hardware management of dirty state enabled.

When the Effective value of TCR_EL2.HA is 0, this field behaves as 0 for all purposes other than a direct read of the value of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

HA, bit [21]

When FEAT_HAFDBS is implemented:

Hardware Access flag update in stage 1 translations from EL2.

HA	Meaning
0b0	Stage 1 Access flag update disabled.
0b1	Stage 1 Access flag update enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBI, bit [20]

Top Byte Ignored. Indicates whether the top byte of an address is used for address match for the TTBR0_EL2 region, or ignored and used for tagged addresses.

For more information, see 'Address tagging'.

TBI	Meaning
0b0	Top Byte used in the address calculation.
0b1	Top Byte ignored in the address calculation.

This affects addresses generated in EL2 using AArch64 where the address would be translated by tables pointed to by TTBR0_EL2. It has an effect whether the EL2 translation regime is enabled or not.

If FEAT_PAuth is implemented and TCR_EL2.TBID is 1, then this field only applies to Data accesses.

If the value of TBI is 1, then bits[63:56] of that target address are also set to 0 before the address is stored in the PC, in the following cases:

- A branch or procedure return within EL2.
- An exception taken to EL2.
- An exception return to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [19]

Reserved, RES0.

PS, bits [18:16]

Physical Address Size.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

PS	Meaning
0b000	32 bits, 4GB.
0b001	36 bits, 64GB.
0b010	40 bits, 1TB.
0b011	42 bits, 4TB.
0b100	44 bits, 16TB.
0b101	48 bits, 256TB.
0b110	52 bits, 4PB.

The following table captures the output address size represented by the value 0b110:

ID_AA64MMFR0_EL1.PARange	Translation Granule	DS[1] | Represented OA size |	
0b0101	Any	Any	48 bits, 256TB
0b011x	4KB or 16KB	0	48 bits, 256TB
0b011x	4KB or 16KB	1	52 bits, 4PB
0b011x	64KB	N/A	52 bits, 4PB

[1] This column represents the value of TCR_EL2.DS.

The output address size represented by the value 0b111 is the OA size represented by 0b110.

If 52-bit PA is supported, and the translation table descriptors cannot express an OA larger than 48-bits, then bits[51:48] of every translation table base address are treated as 0b0000 for the stage of translation controlled by TCR_EL2.

If 56-bit PA is supported, and the translation table descriptors cannot express an OA larger than 52-bits, then bits[55:52] of every translation table base address are treated as 0b0000 for the stage of translation controlled by TCR_EL2.

If the output address size represented by this field is larger than the supported PA size expressed in ID_AA64MMFR0_EL1.PARange, then the output address size is treated as being the same as the supported PA size. Arm strongly recommends that software avoids configuring this field to a value representing an output address size larger than the supported PA size.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TG0, bits [15:14]

Granule size for the TTBR0_EL2.

TG0	Meaning
0b00	4KB.
0b01	64KB.
0b10	16KB.

Other values are reserved.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

If the value is programmed to either a reserved value or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH0, bits [13:12]

Shareability attribute for memory associated with translation table walks using TTBR0_EL2.

SH0	Meaning
0b00	Non-shareable.
0b10	Outer Shareable.
0b11	Inner Shareable.

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ORGN0, bits [11:10]

Outer cacheability attribute for memory associated with translation table walks using TTBR0_EL2.

ORGN0	Meaning
0b00	Normal memory, Outer Non-cacheable.
0b01	Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRGN0, bits [9:8]

Inner cacheability attribute for memory associated with translation table walks using TTBR0_EL2.

IRGN0	Meaning
0b00	Normal memory, Inner Non-cacheable.
0b01	Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

IRGN0	Meaning
0b10	Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [7:6]

Reserved, RES0.

T0SZ, bits [5:0]

The size offset of the memory region addressed by TTBR0_EL2. The region size is $2^{(64-T0SZ)}$ bytes.

The maximum and minimum possible values for T0SZ depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

Note

For the 4KB translation granule, if FEAT_LPA2 is implemented, TCR_EL2.DS is 1, and this field is less than 16, the translation table walk begins with a level -1 initial lookup.
For the 16KB translation granule, if FEAT_LPA2 is implemented, TCR_EL2.DS is 1, and this field is less than 17, the translation table walk begins with a level 0 initial lookup.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When ELIsInHost(EL2):

Any of the bits in TCR_EL2, other than the EPDx bits when they have the value 1, and the A1 bit are permitted to be cached in a TLB.

Bits [63:62]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

MTX1, bit [61]

When FEAT_MTE_NO_ADDRESS_TAGS is implemented or FEAT_MTE_CANONICAL_TAGS is implemented:

Extended memory tag checking.

This field controls address generation and tag checking when EL0 and EL2 are using AArch64 where the data address would be translated by tables pointed to by TTBR1_EL2.

This control has an effect regardless of whether stage 1 of the EL2&0 translation regime is enabled or not.

MTX1	Meaning
`0b0`	This control has no effect on the PE.
`0b1`	Bits[59:56] of a 64-bit VA hold a Logical Address Tag, and all of the following apply: • Bits[59:56] are treated as `0b1111` when checking if the address is out of range. • If FEAT_PAuth is implemented, bits[59:56] are not part of the PAC field. • A Canonical Tag Check operation is performed on Tag Checked memory accesses to a Canonically Tagged memory location.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MTX0, bit [60]

When FEAT_MTE_NO_ADDRESS_TAGS is implemented or FEAT_MTE_CANONICAL_TAGS is implemented:

Extended memory tag checking.

This field controls address generation and tag checking when EL0 and EL2 are using AArch64 where the data address would be translated by tables pointed to by TTBR0_EL2.

This control has an effect regardless of whether stage 1 of the EL2&0 translation regime is enabled or not.

MTX0	Meaning
`0b0`	This control has no effect on the PE.
`0b1`	Bits[59:56] of a 64-bit VA hold a Logical Address Tag, and all of the following apply: • Bits[59:56] are treated as `0b0000` when checking if the address is out of range. • If FEAT_PAuth is implemented, bits[59:56] are not part of the PAC field. • A Canonical Tag Check operation is performed on Tag Checked memory accesses to a Canonically Tagged memory location.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

DS, bit [59]

When FEAT_LPA2 is implemented and (FEAT_D128 is not implemented or TCR2_EL2.D128 == '0'):

This field affects:

- Whether a 52-bit output address can be described by the translation tables of the 4KB or 16KB translation granules.
- The minimum value of TCR_EL2.{T0SZ,T1SZ}.
- How and where shareability for Block and Page descriptors are encoded.

DS	Meaning
`0b0`	Bits[49:48] of translation descriptors are RES0. Bits[9:8] in Block and Page descriptors encode shareability information in the SH[1:0] field. Bits[9:8] in table descriptors are ignored by hardware. The minimum value of the TCR_EL2.{T0SZ, T1SZ} fields is 16. Any memory access using a smaller value generates a stage 1 level 0 translation table fault. Output address[51:48] is `0b0000`.
`0b1`	Bits[49:48] of translation descriptors hold output address[49:48]. Bits[9:8] of Translation table descriptors hold output address[51:50]. The shareability information of Block and Page descriptors for cacheable locations is determined by: • TCR_EL2.SH0 if the VA is an address that is translated using tables pointed to by TTBR0_EL2. • TCR_EL2.SH1 if the VA is an address that is translated using tables pointed to by TTBR1_EL2. The minimum value of the TCR_EL2.{T0SZ, T1SZ} fields is 12. Any memory access using a smaller value generates a stage 1 level 0 translation table fault. All calculations of the stage 1 base address are modified for tables of fewer than 16 entries so that the table is aligned to 64 bytes. Bits[5:2] of TTBR0_EL2 or TTBR1_EL2 are used to hold bits[51:48] of the output address in all cases. **Note** As FEAT_LVA must be implemented if TCR_EL2.DS == 1, the minimum value of the TCR_EL2.{T0SZ, T1SZ} fields is 12, as determined by that extension. For the TLBI Range instructions affecting VA, the format of the argument is changed so that bits[36:0] hold BaseADDR[52:16]. For the 4KB translation granule, bits[15:12] of BaseADDR are treated as `0b0000`. For the 16KB translation granule, bits[15:14] of BaseADDR are treated as `0b00`. **Note** This forces alignment of the ranges used by the TLBI range instructions.

This field is RES0 for a 64KB translation granule.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0, and the Effective value of this bit is `0b0`.

TCMA1, bit [58]

When FEAT_MTE2 is implemented:

Controls the generation of Unchecked accesses at EL2, and at EL0 if HCR_EL2.TGE=1, when address[59:55] = `0b11111`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TCMA1	Meaning
0b0	This control has no effect on the generation of Unchecked accesses at EL2 or EL0.
0b1	All accesses are Unchecked.

Note

Software may change this control bit on a context switch.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCMA0, bit [57]

When FEAT_MTE2 is implemented:

Controls the generation of Unchecked accesses at EL2, and at EL0 if HCR_EL2.TGE=1, when address[59:55] = 0b00000.

TCMA0	Meaning
0b0	This control has no effect on the generation of Unchecked accesses at EL2 or EL0.
0b1	All accesses are Unchecked.

Note

Software may change this control bit on a context switch.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0PD1, bit [56]

When FEAT_E0PD is implemented:

Faulting control for unprivileged access to any address translated by TTBR1_EL2.

E0PD1	Meaning
0b0	Unprivileged access to any address translated by TTBR1_EL2 will not generate a fault by this mechanism.
0b1	Unprivileged access to any address translated by TTBR1_EL2 will generate a level 0 Translation fault.

Level 0 Translation faults generated as a result of this field are not counted as TLB misses for performance monitoring. The fault should take the same time to generate, whether the address is present in the TLB or not, to mitigate attacks that use fault timing.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0PD0, bit [55]

When FEAT_E0PD is implemented:

Faulting control for unprivileged access to any address translated by TTBR0_EL2.

E0PD0	Meaning
0b0	Unprivileged access to any address translated by TTBR0_EL2 will not generate a fault by this mechanism.
0b1	Unprivileged access to any address translated by TTBR0_EL2 will generate a level 0 Translation fault.

Level 0 Translation faults generated as a result of this field are not counted as TLB misses for performance monitoring. The fault should take the same time to generate, whether the address is present in the TLB or not, to mitigate attacks that use fault timing.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NFD1, bit [54]

When FEAT_SVE is implemented:

Non-Fault translation timing Disable when using TTBR1_EL2.

Controls how a TLB miss is reported in response to a non-fault unprivileged access for a virtual address that is translated using TTBR1_EL2.

If SVE is implemented, the affected access types include:

- All accesses due to an SVE non-fault contiguous load instruction.
- Accesses due to an SVE first-fault gather load instruction that are not for the First active element. Accesses due to an SVE first-fault contiguous load instruction are not affected.
- Accesses due to prefetch instructions might be affected, but the effect is not architecturally visible.

NFD1	Meaning
0b0	Does not affect the handling of a TLB miss on accesses translated using TTBR1_EL2.
0b1	A TLB miss on a virtual address that is translated using TTBR1_EL2 due to the specified access types causes the access to fail without taking an exception. The amount of time that the failure takes to be handled should not predictively leak whether it was caused by a TLB miss or a Permission fault, to mitigate attacks that use fault timing.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NFD0, bit [53]

When FEAT_SVE is implemented:

Non-Fault translation timing Disable when using TTBR0_EL2.

Controls how a TLB miss is reported in response to a non-fault unprivileged access for a virtual address that is translated using TTBR0_EL2.

If SVE is implemented, the affected access types include:

- All accesses due to an SVE non-fault contiguous load instruction.
- Accesses due to an SVE first-fault gather load instruction that are not for the First active element. Accesses due to an SVE first-fault contiguous load instruction are not affected.
- Accesses due to prefetch instructions might be affected, but the effect is not architecturally visible.

NFD0	Meaning
0b0	Does not affect the handling of a TLB miss on accesses translated using TTBR0_EL2.
0b1	A TLB miss on a virtual address that is translated using TTBR0_EL2 due to the specified access types causes the access to fail without taking an exception. The amount of time that the failure takes to be handled should not predictively leak whether it was caused by a TLB miss or a Permission fault, to mitigate attacks that use fault timing.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBID1, bit [52]

When FEAT_PAuth is implemented:

Controls the use of the top byte of instruction addresses for address matching.

For the purpose of this field, all cache maintenance and address translation instructions that perform address translation are treated as data accesses.

For more information, see ‘Address tagging’.

TBID1	Meaning
0b0	TCR_EL2.TBI1 applies to Instruction and Data accesses.
0b1	TCR_EL2.TBI1 applies to Data accesses only.

This affects addresses where the address would be translated by tables pointed to by TTBR1_EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

TBID0, bit [51]

When FEAT_PAuth is implemented:

Controls the use of the top byte of instruction addresses for address matching.

For more information, see 'Address tagging'.

TBID0	Meaning
0b0	TCR_EL2.TBI0 applies to Instruction and Data accesses.
0b1	TCR_EL2.TBI0 applies to Data accesses only.

This affects addresses where the address would be translated by tables pointed to by TTBR0_EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU162, bit [50]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry for translations using TTBR1_EL2.

HWU162	Meaning
0b0	For translations using TTBR1_EL2, bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR1_EL2, bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD1 is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD1 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU161, bit [49]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry for translations using TTBR1_EL2.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

HWU161	Meaning
0b0	For translations using TTBR1_EL2, bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR1_EL2, bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD1 is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD1 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU160, bit [48]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry for translations using TTBR1_EL2.

HWU160	Meaning
0b0	For translations using TTBR1_EL2, bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR1_EL2, bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD1 is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD1 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU159, bit [47]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry for translations using TTBR1_EL2.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

HWU159	Meaning
0b0	For translations using TTBR1_EL2, bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR1_EL2, bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD1 is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD1 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU062, bit [46]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry for translations using TTBR0_EL2.

HWU062	Meaning
0b0	For translations using TTBR0_EL2, bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR0_EL2, bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD0 is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD0 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU061, bit [45]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry for translations using TTBR0_EL2.

HWU061	Meaning
0b0	For translations using TTBR0_EL2, bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

HWU061	Meaning
0b1	For translations using TTBR0_EL2, bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD0 is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD0 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU060, bit [44]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry for translations using TTBR0_EL2.

HWU060	Meaning
0b0	For translations using TTBR0_EL2, bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR0_EL2, bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD0 is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD0 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU059, bit [43]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry for translations using TTBR0_EL2.

HWU059	Meaning
0b0	For translations using TTBR0_EL2, bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	For translations using TTBR0_EL2, bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD0 is 1.

The Effective value of this field is 0 if the value of TCR_EL2.HPD0 is 0.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD1, bit [42]

When FEAT_HPDS is implemented:

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, PXNTable, and UXNTable, except NSTable, in the translation tables pointed to by TTBR1_EL2.

HPD1	Meaning
0b0	Hierarchical permissions are enabled.
0b1	Hierarchical permissions are disabled.

When disabled, the permissions are treated as if the bits are zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD0, bit [41]

When FEAT_HPDS is implemented:

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, PXNTable, and UXNTable, except NSTable, in the translation tables pointed to by TTBR0_EL2.

HPD0	Meaning
0b0	Hierarchical permissions are enabled.
0b1	Hierarchical permissions are disabled.

When disabled, the permissions are treated as if the bits are zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HD, bit [40]

When FEAT_HAFDBS is implemented:

Hardware management of dirty state in stage 1 translations from EL2.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

HD	Meaning
0b0	Stage 1 hardware management of dirty state disabled.
0b1	Stage 1 hardware management of dirty state enabled.

When the Effective value of TCR_EL2.HA is 0, this field behaves as 0 for all purposes other than a direct read of the value of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HA, bit [39]

When FEAT_HAFDBS is implemented:

Hardware Access flag update in stage 1 translations from EL2.

HA	Meaning
0b0	Stage 1 Access flag update disabled.
0b1	Stage 1 Access flag update enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBI1, bit [38]

Top Byte Ignored. Indicates whether the top byte of an address is used for address match for the TTBR1_EL2 region, or ignored and used for tagged addresses.

For more information, see 'Address tagging'.

TBI1	Meaning
0b0	Top Byte used in the address calculation.
0b1	Top Byte ignored in the address calculation.

This affects addresses generated in EL0 and EL2 using AArch64 where the address would be translated by tables pointed to by TTBR1_EL2. It has an effect whether the EL2&0 translation regime is enabled or not.

If FEAT_PAuth is implemented and TCR_EL2.TBID1 is 1, then this field only applies to Data accesses.

If the value of TBI1 is 1 and bit [55] of the target address to be stored to the PC is 1, then bits[63:56] of that target address are also set to 1 before the address is stored in the PC, in the following cases:

- A branch or procedure return within EL2

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- If the Effective value of HCR_EL2.{E2H, TGE} is {1,1}, a branch or procedure return within EL0.
- An exception taken to EL2.
- An exception return to EL2.
- If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, then an exception return to EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TBI0, bit [37]

Top Byte Ignored. Indicates whether the top byte of an address is used for address match for the TTBR0_EL2 region, or ignored and used for tagged addresses.

For more information, see 'Address tagging'.

TBI0	Meaning
0b0	Top Byte used in the address calculation.
0b1	Top Byte ignored in the address calculation.

This affects addresses generated in EL0 and EL2 using AArch64 where the address would be translated by tables pointed to by TTBR0_EL2. It has an effect whether the EL2&0 translation regime is enabled or not.

If FEAT_PAuth is implemented and TCR_EL2.TBID0 is 1, then this field only applies to Data accesses.

If the value of TBI0 is 1 and bit [55] of the target address to be stored to the PC is 0, then bits[63:56] of that target address are also set to 0 before the address is stored in the PC, in the following cases:

- A branch or procedure return within EL2
- If the Effective value of HCR_EL2.{E2H, TGE} is {1,1}, a branch or procedure return within EL0.
- An exception taken to EL2.
- An exception return to EL2.
- If the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, then an exception return to EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

AS, bit [36]

ASID Size.

AS	Meaning
0b0	8 bit - the upper 8 bits of TTBR0_EL2 and TTBR1_EL2 are ignored by hardware for every purpose except reading back the register, and are treated as if they are all zeros for when used for allocation and matching entries in the TLB.
0b1	16 bit - the upper 16 bits of TTBR0_EL2 and TTBR1_EL2 are used for allocation and matching in the TLB.

If the implementation has only 8 bits of ASID, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bit [35]

Reserved, RES0.

IPS, bits [34:32]

Intermediate Physical Address Size.

IPS	Meaning
0b000	32 bits, 4GB.
0b001	36 bits, 64GB.
0b010	40 bits, 1TB.
0b011	42 bits, 4TB.
0b100	44 bits, 16TB.
0b101	48 bits, 256TB.
0b110	52 bits, 4PB.
0b111	56 bits, 64PB.

The values 0b110 and 0b111 represent different output address sizes depending on implementation choices and translation configuration.

The following table captures the output address size represented by the value 0b110:

Descriptor Format	ID_AA64MMFR0_EL1.PARange	Translation Granule	DS[1] | Represented OA size |	
Any	0b0101	Any	Any	48 bits, 256TB
VMSAv8-64	0b011x	4KB or 16KB	0	48 bits, 256TB
VMSAv8-64	0b011x	4KB or 16KB	1	52 bits, 4PB
VMSAv8-64	0b011x	64KB	N/A	52 bits, 4PB
VMSAv9-128	0b011x	Any	N/A	52 bits, 4PB

[1] This column represents the value of TCR_EL2.DS.

The following table captures the output address size represented by the value 0b111:

Descriptor Format	ID_AA64MMFR0_EL1.PARange	Translation Granule	Represented OA size
Any	0b0110	Any	OA size represented by 0b110
VMSAv8-64	0b0111	Any	OA size represented by 0b110
VMSAv9-128	0b0111	Any	56 bits, 64PB

If 52-bit PA is supported, and the translation table descriptors cannot express an OA larger than 48-bits, then bits[51:48] of every translation table base address are treated as 0b0000 for the stage of translation controlled by TCR_EL2.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

If 56-bit PA is supported, and the translation table descriptors cannot express an OA larger than 52-bits, then bits[55:52] of every translation table base address are treated as `0b0000` for the stage of translation controlled by TCR_EL2.

If the output address size represented by this field is larger than the supported PA size expressed in ID_AA64MMFR0_EL1.PARange, then the output address size is treated as being the same as the supported PA size. Arm strongly recommends that software avoids configuring this field to a value representing an output address size larger than the supported PA size.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TG1, bits [31:30]

Granule size for the TTBR1_EL2.

TG1	Meaning
`0b01`	16KB.
`0b10`	4KB.
`0b11`	64KB.

Other values are reserved.

If the value is programmed to either a reserved value, or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH1, bits [29:28]

Shareability attribute for memory associated with translation table walks using TTBR1_EL2.

SH1	Meaning
`0b00`	Non-shareable.
`0b10`	Outer Shareable.
`0b11`	Inner Shareable.

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ORGN1, bits [27:26]

Outer cacheability attribute for memory associated with translation table walks using TTBR1_EL2.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ORGN1	Meaning
0b00	Normal memory, Outer Non-cacheable.
0b01	Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRGN1, bits [25:24]

Inner cacheability attribute for memory associated with translation table walks using TTBR1_EL2.

IRGN1	Meaning
0b00	Normal memory, Inner Non-cacheable.
0b01	Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EPD1, bit [23]

Translation table walk disable for translations using TTBR1_EL2. This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using TTBR1_EL2. The encoding of this bit is:

EPD1	Meaning
0b0	Perform translation table walks using TTBR1_EL2.
0b1	A TLB miss on an address that is translated using TTBR1_EL2 generates a Translation fault. No translation table walk is performed.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A1, bit [22]

Selects whether TTBR0_EL2 or TTBR1_EL2 defines the ASID. The encoding of this bit is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

A1	Meaning
0b0	TTBR0_EL2.ASID defines the ASID.
0b1	TTBR1_EL2.ASID defines the ASID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T1SZ, bits [21:16]

The size offset of the memory region addressed by TTBR1_EL2. The region size is $2^{(64-T1SZ)}$ bytes.

The maximum and minimum possible values for T1SZ depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

Note

For the 4KB translation granule, if FEAT_LPA2 is implemented, TCR_EL2.DS is 1, and this field is less than 16, the translation table walk begins with a level -1 initial lookup.
For the 16KB translation granule, if FEAT_LPA2 is implemented, TCR_EL2.DS is 1, and this field is less than 17, the translation table walk begins with a level 0 initial lookup.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TG0, bits [15:14]

Granule size for the TTBR0_EL2.

TG0	Meaning
0b00	4KB.
0b01	64KB.
0b10	16KB.

Other values are reserved.

If the value is programmed to either a reserved value, or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH0, bits [13:12]

Shareability attribute for memory associated with translation table walks using TTBR0_EL2.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SH0	Meaning
0b00	Non-shareable.
0b10	Outer Shareable.
0b11	Inner Shareable.

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ORGN0, bits [11:10]

Outer cacheability attribute for memory associated with translation table walks using TTBR0_EL2.

ORGN0	Meaning
0b00	Normal memory, Outer Non-cacheable.
0b01	Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRGN0, bits [9:8]

Inner cacheability attribute for memory associated with translation table walks using TTBR0_EL2.

IRGN0	Meaning
0b00	Normal memory, Inner Non-cacheable.
0b01	Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EPD0, bit [7]

Translation table walk disable for translations using TTBR0_EL2. This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using TTBR0_EL2. The encoding of this bit is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EPD0	Meaning
0b0	Perform translation table walks using TTBR0_EL2.
0b1	A TLB miss on an address that is translated using TTBR0_EL2 generates a Translation fault. No translation table walk is performed.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [6]

Reserved, RES0.

T0SZ, bits [5:0]

The size offset of the memory region addressed by TTBR0_EL2. The region size is $2^{(64-T0SZ)}$ bytes.

The maximum and minimum possible values for T0SZ depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

Note

For the 4KB translation granule, if FEAT_LPA2 is implemented, TCR_EL2.DS is 1, and this field is less than 16, the translation table walk begins with a level -1 initial lookup.
For the 16KB translation granule, if FEAT_LPA2 is implemented, TCR_EL2.DS is 1, and this field is less than 17, the translation table walk begins with a level 0 initial lookup.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TCR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name TCR_EL2 or TCR_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

If FEAT_SRMASK is implemented, accesses to TCR_EL2 are masked by TCRMASK_EL2.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TCR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = TCR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TCR_EL2;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

MSR TCR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if IsFeatureImplemented(FEAT_SRMASK) then
        TCR_EL2 = (X[t, 64] AND NOT EffectiveTCRMASK_EL2()) OR (TCR_EL2 AND EffectiveTCRMASK_EL2());
    else
        TCR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TCR_EL2 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, TCR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
         ↪&& HFGRTR_EL2.TCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x120];
    else
        X[t, 64] = TCR_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = TCR_EL2;
    else
        X[t, 64] = TCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TCR_EL1;
```

When FEAT_VHE is implemented MSR TCR_EL1, <Xt>

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
         ↪&& HFGWTR_EL2.TCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x120] = X[t, 64];
    else
        if IsFeatureImplemented(FEAT_SRMASK) then
            TCR_EL1 = (X[t, 64] AND NOT EffectiveTCRMASK_EL1()) OR (TCR_EL1 AND
                 ↪EffectiveTCRMASK_EL1());
        else
            TCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_SRMASK) then
            TCR_EL2 = (X[t, 64] AND NOT EffectiveTCRMASK_EL2()) OR (TCR_EL2 AND
                 ↪EffectiveTCRMASK_EL2());
        else
            TCR_EL2 = X[t, 64];
    else
        TCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TCR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.196 TCR_EL3, Translation Control Register (EL3)

The TCR_EL3 characteristics are:

Purpose

The control register for stage 1 of the EL3 translation regime.

Configuration

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TCR_EL3 are UNDEFINED.

Attributes

TCR_EL3 is a 64-bit register.

Field descriptions

Unless stated otherwise, any of the bits in TCR_EL3 are permitted to be cached in a TLB.

Bits [63:44]

Reserved, RES0.

DisCH0, bit [43]

When FEAT_D128 is implemented and TCR_EL3.D128 == '1':

Disable the Contiguous bit for the Start Table.

DisCH0	Meaning
0b0	The Contiguous bit of Block or Page descriptors of the Start Table is not affected by this field.
0b1	The Contiguous bit of Block or Page descriptors of the Start Table is treated as 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

HAFT, bit [42]

When FEAT_HAFT is implemented:

Hardware managed Access Flag for Table descriptors.

Enables the Hardware managed Access Flag for Table descriptors.

HAFT	Meaning
0b0	Hardware managed Access Flag for Table descriptors is disabled.
0b1	Hardware managed Access Flag for Table descriptors is enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PTTWI, bit [41]

When FEAT_THE is implemented:

Permit Translation table walk Incoherence.

Permits RCWS instructions to generate writes that have the Reduced Coherence property.

PTTWI	Meaning
0b0	Write accesses generated by RCWS at EL3 do not have the Reduced Coherence property.
0b1	Write accesses generated by RCWS at EL3 have the Reduced Coherence property.

This bit is permitted to be implemented as a read-only bit with a fixed value of 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [40:39]

Reserved, RES0.

D128, bit [38]

When FEAT_D128 is implemented:

Enables VMSAv9-128 translation system.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D128	Meaning
0b0	Translation system follows VMSAv8-64 translation process.
0b1	Translation system follows VMSAv9-128 translation process.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AIE, bit [37]

When FEAT_AIE is implemented:

Enable Attribute Indexing Extension.

AIE	Meaning
0b0	Attribute Indexing Extension Disabled.
0b1	Attribute Indexing Extension Enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

POE, bit [36]

When FEAT_S1POE is implemented:

Enables Permission Overlay for EL3 accesses.

POE	Meaning
0b0	Permission overlay disabled for EL3 access in stage 1 of EL3 translation regime.
0b1	Permission overlay enabled for EL3 access in stage 1 of EL3 translation regime.

This bit is not permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

PIE, bit [35]

When FEAT_S1PIE is implemented:

Enables usage of Indirect Permission Scheme.

PIE	Meaning
0b0	Direct permission model.
0b1	Indirect permission model.

This field is RES1 when TCR_EL3.D128 is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PnCH, bit [34]

When FEAT_THE is implemented:

Protected attribute enable. Enables use of bit[52] of the stage 1 translation table entries as the Protected bit, for translations using TTBR0_EL3.

PnCH	Meaning
0b0	For translations using TTBR0_EL3, bit[52] of each stage 1 translation table entry is not the Protected bit.
0b1	For translations using TTBR0_EL3, bit[52] of each stage 1 translation table entry is the Protected bit.

If bit[52] is used as the Protected bit, it is not used as the Contiguous bit.

This field is RES0 when TCR_EL3.D128 is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MTX, bit [33]

When FEAT_MTE_NO_ADDRESS_TAGS is implemented or FEAT_MTE_CANONICAL_TAGS is implemented:

Extended memory tag checking.

This field controls address generation and tag checking when EL3 is using AArch64 where the data address would be translated by tables pointed to by TTBR0_EL3.

This control has an effect regardless of whether stage 1 of the EL3 translation regime is enabled or not.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

MTX	Meaning
0b0	This control has no effect on the PE.
0b1	Bits[59:56] of a 64-bit VA hold a Logical Address Tag, and all of the following apply: • Bits[59:56] are treated as 0b0000 when checking if the address is out of range. • If FEAT_PAuth is implemented, bits[59:56] are not part of the PAC field. • A Canonical Tag Check operation is performed on Tag Checked memory accesses to a Canonically Tagged memory location.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DS, bit [32]

When FEAT_LPA2 is implemented and (FEAT_D128 is not implemented or TCR_EL3.D128 == '0'):

This field affects:

- Whether a 52-bit output address can be described by the translation tables of the 4KB or 16KB translation granules.
- The minimum value of TCR_EL3.T0SZ.
- How and where shareability for Block and Page descriptors are encoded.

DS	Meaning
0b0	Bits[49:48] of translation descriptors are RES0. Bits[9:8] in Block and Page descriptors encode shareability information in the SH[1:0] field. Bits[9:8] in Table descriptors are ignored by hardware. The minimum value of TCR_EL3.T0SZ is 16. Any memory access using a smaller value generates a stage 1 level 0 translation table fault. Output address[51:48] is 0b0000.
0b1	Bits[49:48] of translation descriptors hold output address[49:48]. Bits[9:8] of table translation descriptors hold output address[51:50]. The shareability information of Block and Page descriptors for cacheable locations is determined by TCR_EL3.SH0. The minimum value of TCR_EL3.T0SZ is 12. Any memory access using a smaller value generates a stage 1 level 0 translation table fault. All calculations of the stage 1 base address are modified for tables of fewer than 8 entries so that the table is aligned to 64 bytes. Bits[5:2] of TTBR0_EL3 are used to hold bits[51:48] of the output address in all cases. **Note** As FEAT_LVA must be implemented if TCR_EL3.DS == 1, the minimum value of the TCR_EL3.T0SZ field is 12, as determined by that extension. For the TLBI Range instructions affecting VA, the format of the argument is changed so that bits[36:0] hold BaseADDR[52:16]. For the 4KB translation granule, bits[15:12] of BaseADDR are treated as 0b0000. For the 16KB translation granule, bits[15:14] of BaseADDR are treated as 0b00. **Note** This forces alignment of the ranges used by the TLBI range instructions.

This field is RES0 for a 64KB translation granule.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

The Effective value of this bit is `0b0`.

Access to this field is RES0.

Bit [31]

Reserved, RES1.

TCMA, bit [30]

When FEAT_MTE2 is implemented:

Controls the generation of Unchecked accesses at EL3 when address [59:56] = `0b0000`.

TCMA	Meaning
`0b0`	This control has no effect on the generation of Unchecked accesses.
`0b1`	All accesses are Unchecked.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBID, bit [29]

When FEAT_PAuth is implemented:

Controls the use of the top byte of instruction addresses for address matching.

TBID	Meaning
`0b0`	TCR_EL3.TBI applies to Instruction and Data accesses.
`0b1`	TCR_EL3.TBI applies to Data accesses only.

This affects addresses where the address would be translated by tables pointed to by TTBR0_EL3.

For the purpose of this field, all cache maintenance and address translation instructions that perform address translation are treated as data accesses.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU62, bit [28]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry.

HWU62	Meaning
`0b0`	Bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
`0b1`	Bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL3.HPD is 1.

The Effective value of this field is 0 if the value of TCR_EL3.HPD is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU61, bit [27]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry.

HWU61	Meaning
`0b0`	Bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
`0b1`	Bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL3.HPD is 1.

The Effective value of this field is 0 if the value of TCR_EL3.HPD is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU60, bit [26]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

HWU60	Meaning
0b0	Bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL3.HPD is 1.

The Effective value of this field is 0 if the value of TCR_EL3.HPD is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU59, bit [25]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry.

HWU59	Meaning
0b0	Bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL3.HPD is 1.

The Effective value of this field is 0 if the value of TCR_EL3.HPD is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD, bit [24]

When FEAT_HPDS is implemented:

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, PXNTable, and UXNTable, except NSTable, in the translation tables pointed to by TTBR0_EL3.

HPD	Meaning
0b0	Hierarchical permissions are enabled.
0b1	Hierarchical permissions are disabled. **Note** In this case, bit[61] (APTable[0]) and bit[59] (PXNTable) of the next level descriptor attributes are required to be ignored by the PE, and are no longer reserved, allowing them to be used by software.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When disabled, the permissions are treated as if the bits are zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [23]

Reserved, RES1.

HD, bit [22]

When FEAT_HAFDBS is implemented:

Hardware management of dirty state in stage 1 translations from EL3.

HD	Meaning
0b0	Stage 1 hardware management of dirty state disabled.
0b1	Stage 1 hardware management of dirty state enabled.

When the Effective value of TCR_EL3.HA is 0, this field behaves as 0 for all purposes other than a direct read of the value of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HA, bit [21]

When FEAT_HAFDBS is implemented:

Hardware Access flag update in stage 1 translations from EL3.

HA	Meaning
0b0	Stage 1 Access flag update disabled.
0b1	Stage 1 Access flag update enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBI, bit [20]

Top Byte Ignored. Indicates whether the top byte of an address is used for address match for the TTBR0_EL3 region, or ignored and used for tagged addresses.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TBI	Meaning
0b0	Top Byte used in the address calculation.
0b1	Top Byte ignored in the address calculation.

This affects addresses generated in EL3 using AArch64 where the address would be translated by tables pointed to by TTBR0_EL3. It has an effect whether the EL3 translation regime is enabled or not.

If FEAT_PAuth is implemented and TCR_EL3.TBID is 1, then this field only applies to Data accesses.

Otherwise, if the value of TBI is 1, then bits[63:56] of that target address are also set to 0 before the address is stored in the PC, in the following cases:

- A branch or procedure return within EL3.
- A exception taken to EL3.
- An exception return to EL3.

For more information, see 'Address tagging'.

Note

This control detrmines the scope of address tagging. It never causes an exception to be generated.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [19]

Reserved, RES0.

PS, bits [18:16]

Physical Address Size.

PS	Meaning
0b000	32 bits, 4GB.
0b001	36 bits, 64GB.
0b010	40 bits, 1TB.
0b011	42 bits, 4TB.
0b100	44 bits, 16TB.
0b101	48 bits, 256TB.
0b110	52 bits, 4PB.
0b111	56 bits, 64PB.

The values 0b110 and 0b111 represent different output address sizes depending on implementation choices and translation configuration.

The following table captures the output address size represented by the value 0b110:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Descriptor Format	ID_AA64MMFR0_EL1.PARange	Translation Granule	DS[1] | Represented OA size |	
Any	`0b0101`	Any	Any	48 bits, 256TB
VMSAv8-64	`0b011x`	4KB or 16KB	0	48 bits, 256TB
VMSAv8-64	`0b011x`	4KB or 16KB	1	52 bits, 4PB
VMSAv8-64	`0b011x`	64KB	N/A	52 bits, 4PB
VMSAv9-128	`0b011x`	Any	N/A	52 bits, 4PB

[1] This column represents the value of TCR_EL3.DS.

The following table captures the output address size represented by the value `0b111`:

Descriptor Format	ID_AA64MMFR0_EL1.PARange	Translation Granule	Represented OA size
Any	`0b0110`	Any	OA size represented by `0b110`
VMSAv8-64	`0b0111`	Any	OA size represented by `0b110`
VMSAv9-128	`0b0111`	Any	56 bits, 64PB

If 52-bit PA is supported, and the translation table descriptors cannot express an OA larger than 48-bits, then bits[51:48] of every translation table base address are treated as `0b0000` for the stage of translation controlled by TCR_EL3.

If 56-bit PA is supported, and the translation table descriptors cannot express an OA larger than 52-bits, then bits[55:52] of every translation table base address are treated as `0b0000` for the stage of translation controlled by TCR_EL3.

If the output address size represented by this field is larger than the supported PA size expressed in ID_AA64MMFR0_EL1.PARange, then the output address size is treated as being the same as the supported PA size. Arm strongly recommends that software avoids configuring this field to a value representing an output address size larger than the supported PA size.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TG0, bits [15:14]

Granule size for the TTBR0_EL3.

TG0	Meaning
`0b00`	4KB.
`0b01`	64KB.
`0b10`	16KB.

Other values are reserved.

If the value is programmed to either a reserved value or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH0, bits [13:12]

Shareability attribute for memory associated with translation table walks using TTBR0_EL3.

SH0	Meaning
0b00	Non-shareable.
0b10	Outer Shareable.
0b11	Inner Shareable.

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ORGN0, bits [11:10]

Outer cacheability attribute for memory associated with translation table walks using TTBR0_EL3.

ORGN0	Meaning
0b00	Normal memory, Outer Non-cacheable.
0b01	Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRGN0, bits [9:8]

Inner cacheability attribute for memory associated with translation table walks using TTBR0_EL3.

IRGN0	Meaning
0b00	Normal memory, Inner Non-cacheable.
0b01	Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bits [7:6]

Reserved, RES0.

T0SZ, bits [5:0]

The size offset of the memory region addressed by TTBR0_EL3. The region size is $2^{(64-T0SZ)}$ bytes.

The maximum and minimum possible values for T0SZ depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

Note

For the 4KB translation granule, if FEAT_LPA2 is implemented, TCR_EL3.DS is 1, and this field is less than 16, the translation table walk begins with a level -1 initial lookup.
For the 16KB translation granule, if FEAT_LPA2 is implemented, TCR_EL3.DS is 1, and this field is less than 17, the translation table walk begins with a level 0 initial lookup.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TCR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TCR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0010	0b0000	0b010

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TCR_EL3;
```

```
MSR TCR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0010	0b0000	0b010

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.TCR_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TCR_EL3 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.197 TCRMASK_EL1, Translation Control Masking Register (EL1)

The TCRMASK_EL1 characteristics are:

Purpose

Mask register to prevent updates of fields in TCR_EL1 on writes to TCR_EL1 or TCRALIAS_EL1.

Configuration

This register is present only when FEAT_SRMASK is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TCRMASK_EL1 are UNDEFINED.

Attributes

TCRMASK_EL1 is a 64-bit register.

Field descriptions

Bits [63:62]

Reserved, RES0.

MTX1, bit [61]

When FEAT_MTE_NO_ADDRESS_TAGS is implemented or FEAT_MTE_CANONICAL_TAGS is implemented:

Mask bit for MTX1.

MTX1	Meaning
0b0	TCR_EL1.MTX1 is writeable.
0b1	TCR_EL1.MTX1 is not writeable.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MTX0, bit [60]

When FEAT_MTE_NO_ADDRESS_TAGS is implemented or FEAT_MTE_CANONICAL_TAGS is implemented:

Mask bit for MTX0.

MTX0	Meaning
`0b0`	TCR_EL1.MTX0 is writeable.
`0b1`	TCR_EL1.MTX0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DS, bit [59]

When FEAT_LPA2 is implemented:

Mask bit for DS.

DS	Meaning
`0b0`	TCR_EL1.DS is writeable.
`0b1`	TCR_EL1.DS is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCMA1, bit [58]

When FEAT_MTE2 is implemented:

Mask bit for TCMA1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TCMA1	Meaning
0b0	TCR_EL1.TCMA1 is writeable.
0b1	TCR_EL1.TCMA1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCMA0, bit [57]

When FEAT_MTE2 is implemented:

Mask bit for TCMA0.

TCMA0	Meaning
0b0	TCR_EL1.TCMA0 is writeable.
0b1	TCR_EL1.TCMA0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0PD1, bit [56]

When FEAT_E0PD is implemented:

Mask bit for E0PD1.

E0PD1	Meaning
0b0	TCR_EL1.E0PD1 is writeable.
0b1	TCR_EL1.E0PD1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

E0PD0, bit [55]

When FEAT_E0PD is implemented:

Mask bit for E0PD0.

E0PD0	Meaning
0b0	TCR_EL1.E0PD0 is writeable.
0b1	TCR_EL1.E0PD0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NFD1, bit [54]

When FEAT_SVE is implemented:

Mask bit for NFD1.

NFD1	Meaning
0b0	TCR_EL1.NFD1 is writeable.
0b1	TCR_EL1.NFD1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NFD0, bit [53]

When FEAT_SVE is implemented:

Mask bit for NFD0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

NFD0	Meaning
0b0	TCR_EL1.NFD0 is writeable.
0b1	TCR_EL1.NFD0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBID1, bit [52]

When FEAT_PAuth is implemented:

Mask bit for TBID1.

TBID1	Meaning
0b0	TCR_EL1.TBID1 is writeable.
0b1	TCR_EL1.TBID1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBID0, bit [51]

When FEAT_PAuth is implemented:

Mask bit for TBID0.

TBID0	Meaning
0b0	TCR_EL1.TBID0 is writeable.
0b1	TCR_EL1.TBID0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

HWU162, bit [50]

When FEAT_HPDS2 is implemented:

Mask bit for HWU162.

HWU162	Meaning
0b0	TCR_EL1.HWU162 is writeable.
0b1	TCR_EL1.HWU162 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU161, bit [49]

When FEAT_HPDS2 is implemented:

Mask bit for HWU161.

HWU161	Meaning
0b0	TCR_EL1.HWU161 is writeable.
0b1	TCR_EL1.HWU161 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU160, bit [48]

When FEAT_HPDS2 is implemented:

Mask bit for HWU160.

HWU160	Meaning
0b0	TCR_EL1.HWU160 is writeable.
0b1	TCR_EL1.HWU160 is not writeable.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU159, bit [47]

When FEAT_HPDS2 is implemented:

Mask bit for HWU159.

HWU159	Meaning
`0b0`	TCR_EL1.HWU159 is writeable.
`0b1`	TCR_EL1.HWU159 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU062, bit [46]

When FEAT_HPDS2 is implemented:

Mask bit for HWU062.

HWU062	Meaning
`0b0`	TCR_EL1.HWU062 is writeable.
`0b1`	TCR_EL1.HWU062 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU061, bit [45]

When FEAT_HPDS2 is implemented:

Mask bit for HWU061.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

HWU061	Meaning
0b0	TCR_EL1.HWU061 is writeable.
0b1	TCR_EL1.HWU061 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU060, bit [44]

When FEAT_HPDS2 is implemented:

Mask bit for HWU060.

HWU060	Meaning
0b0	TCR_EL1.HWU060 is writeable.
0b1	TCR_EL1.HWU060 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU059, bit [43]

When FEAT_HPDS2 is implemented:

Mask bit for HWU059.

HWU059	Meaning
0b0	TCR_EL1.HWU059 is writeable.
0b1	TCR_EL1.HWU059 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

HPD1, bit [42]

When FEAT_HPDS is implemented:

Mask bit for HPD1.

HPD1	Meaning
0b0	TCR_EL1.HPD1 is writeable.
0b1	TCR_EL1.HPD1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD0, bit [41]

When FEAT_HPDS is implemented:

Mask bit for HPD0.

HPD0	Meaning
0b0	TCR_EL1.HPD0 is writeable.
0b1	TCR_EL1.HPD0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HD, bit [40]

When FEAT_HAFDBS is implemented:

Mask bit for HD.

HD	Meaning
0b0	TCR_EL1.HD is writeable.
0b1	TCR_EL1.HD is not writeable.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HA, bit [39]

When FEAT_HAFDBS is implemented:

Mask bit for HA.

HA	Meaning
`0b0`	TCR_EL1.HA is writeable.
`0b1`	TCR_EL1.HA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBI1, bit [38]

Mask bit for TBI1.

TBI1	Meaning
`0b0`	TCR_EL1.TBI1 is writeable.
`0b1`	TCR_EL1.TBI1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TBI0, bit [37]

Mask bit for TBI0.

TBI0	Meaning
`0b0`	TCR_EL1.TBI0 is writeable.
`0b1`	TCR_EL1.TBI0 is not writeable.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

AS, bit [36]

Mask bit for AS.

AS	Meaning
`0b0`	TCR_EL1.AS is writeable.
`0b1`	TCR_EL1.AS is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bits [35:33]

Reserved, RES0.

IPS, bit [32]

Mask bit for IPS.

IPS	Meaning
`0b0`	TCR_EL1.IPS is writeable.
`0b1`	TCR_EL1.IPS is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [31]

Reserved, RES0.

TG1, bit [30]

Mask bit for TG1.

TG1	Meaning
`0b0`	TCR_EL1.TG1 is writeable.
`0b1`	TCR_EL1.TG1 is not writeable.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [29]

Reserved, RES0.

SH1, bit [28]

Mask bit for SH1.

SH1	Meaning
`0b0`	TCR_EL1.SH1 is writeable.
`0b1`	TCR_EL1.SH1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [27]

Reserved, RES0.

ORGN1, bit [26]

Mask bit for ORGN1.

ORGN1	Meaning
`0b0`	TCR_EL1.ORGN1 is writeable.
`0b1`	TCR_EL1.ORGN1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [25]

Reserved, RES0.

IRGN1, bit [24]

Mask bit for IRGN1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

IRGN1	Meaning
0b0	TCR_EL1.IRGN1 is writeable.
0b1	TCR_EL1.IRGN1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

EPD1, bit [23]

Mask bit for EPD1.

EPD1	Meaning
0b0	TCR_EL1.EPD1 is writeable.
0b1	TCR_EL1.EPD1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

A1, bit [22]

Mask bit for A1.

A1	Meaning
0b0	TCR_EL1.A1 is writeable.
0b1	TCR_EL1.A1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bits [21:17]

Reserved, RES0.

T1SZ, bit [16]

Mask bit for T1SZ.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T1SZ	Meaning
0b0	TCR_EL1.T1SZ is writeable.
0b1	TCR_EL1.T1SZ is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [15]

Reserved, RES0.

TG0, bit [14]

Mask bit for TG0.

TG0	Meaning
0b0	TCR_EL1.TG0 is writeable.
0b1	TCR_EL1.TG0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [13]

Reserved, RES0.

SH0, bit [12]

Mask bit for SH0.

SH0	Meaning
0b0	TCR_EL1.SH0 is writeable.
0b1	TCR_EL1.SH0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [11]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ORGN0, bit [10]

Mask bit for ORGN0.

ORGN0	Meaning
0b0	TCR_EL1.ORGN0 is writeable.
0b1	TCR_EL1.ORGN0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [9]

Reserved, RES0.

IRGN0, bit [8]

Mask bit for IRGN0.

IRGN0	Meaning
0b0	TCR_EL1.IRGN0 is writeable.
0b1	TCR_EL1.IRGN0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

EPD0, bit [7]

Mask bit for EPD0.

EPD0	Meaning
0b0	TCR_EL1.EPD0 is writeable.
0b1	TCR_EL1.EPD0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bits [6:1]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T0SZ, bit [0]

Mask bit for T0SZ.

T0SZ	Meaning
0b0	TCR_EL1.T0SZ is writeable.
0b1	TCR_EL1.T0SZ is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Accessing TCRMASK_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name TCRMASK_EL1 or TCRMASK_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TCRMASK_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0111	0b010

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HFGRTR2_EL2.nTCRMASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x330];
    else
        X[t, 64] = TCRMASK_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = TCRMASK_EL2;
    else
        X[t, 64] = TCRMASK_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TCRMASK_EL1;
```

MSR TCRMASK_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0111	0b010

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HFGWTR2_EL2.nTCRMASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x330] = X[t, 64];
    elsif !IsZero(EffectiveTCRMASK_EL1()) then
        UNDEFINED;
    else
        TCRMASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        if !IsZero(EffectiveTCRMASK_EL2()) then
            UNDEFINED;
        else
            TCRMASK_EL2 = X[t, 64];
    else
        TCRMASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TCRMASK_EL1 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, TCRMASK_EL12

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0111	0b010

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x330];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = TCRMASK_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = TCRMASK_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented MSR TCRMASK_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0111	0b010

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x330] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            TCRMASK_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        TCRMASK_EL1 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.198 TCRMASK_EL2, Translation Control Masking Register (EL2)

The TCRMASK_EL2 characteristics are:

Purpose

Mask register to prevent updates of fields in TCR_EL2 on writes.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

This register is present only when FEAT_SRMASK is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TCRMASK_EL2 are UNDEFINED.

Attributes

TCRMASK_EL2 is a 64-bit register.

Field descriptions

When !ELIsInHost(EL2):

Bits [63:34]

Reserved, RES0.

MTX, bit [33]

When FEAT_MTE_NO_ADDRESS_TAGS is implemented or FEAT_MTE_CANONICAL_TAGS is implemented:

Mask bit for MTX.

MTX	Meaning
0b0	TCR_EL2.MTX is writeable.
0b1	TCR_EL2.MTX is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

DS, bit [32]

When FEAT_LPA2 is implemented:

Mask bit for DS.

DS	Meaning
0b0	TCR_EL2.DS is writeable.
0b1	TCR_EL2.DS is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [31]

Reserved, RES0.

TCMA, bit [30]

When FEAT_MTE2 is implemented:

Mask bit for TCMA.

TCMA	Meaning
0b0	TCR_EL2.TCMA is writeable.
0b1	TCR_EL2.TCMA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBID, bit [29]

When FEAT_PAuth is implemented:

Mask bit for TBID.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TBID	Meaning
0b0	TCR_EL2.TBID is writeable.
0b1	TCR_EL2.TBID is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU62, bit [28]

When FEAT_HPDS2 is implemented:

Mask bit for HWU62.

HWU62	Meaning
0b0	TCR_EL2.HWU62 is writeable.
0b1	TCR_EL2.HWU62 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU61, bit [27]

When FEAT_HPDS2 is implemented:

Mask bit for HWU61.

HWU61	Meaning
0b0	TCR_EL2.HWU61 is writeable.
0b1	TCR_EL2.HWU61 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

HWU60, bit [26]

When FEAT_HPDS2 is implemented:

Mask bit for HWU60.

HWU60	Meaning
0b0	TCR_EL2.HWU60 is writeable.
0b1	TCR_EL2.HWU60 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU59, bit [25]

When FEAT_HPDS2 is implemented:

Mask bit for HWU59.

HWU59	Meaning
0b0	TCR_EL2.HWU59 is writeable.
0b1	TCR_EL2.HWU59 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD, bit [24]

When FEAT_HPDS is implemented:

Mask bit for HPD.

HPD	Meaning
0b0	TCR_EL2.HPD is writeable.
0b1	TCR_EL2.HPD is not writeable.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [23]

Reserved, RES0.

HD, bit [22]

When FEAT_HAFDBS is implemented:

Mask bit for HD.

HD	Meaning
`0b0`	TCR_EL2.HD is writeable.
`0b1`	TCR_EL2.HD is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HA, bit [21]

When FEAT_HAFDBS is implemented:

Mask bit for HA.

HA	Meaning
`0b0`	TCR_EL2.HA is writeable.
`0b1`	TCR_EL2.HA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBI, bit [20]

Mask bit for TBI.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TBI	Meaning
0b0	TCR_EL2.TBI is writeable.
0b1	TCR_EL2.TBI is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bits [19:17]

Reserved, RES0.

PS, bit [16]

Mask bit for PS.

PS	Meaning
0b0	TCR_EL2.PS is writeable.
0b1	TCR_EL2.PS is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [15]

Reserved, RES0.

TG0, bit [14]

Mask bit for TG0.

TG0	Meaning
0b0	TCR_EL2.TG0 is writeable.
0b1	TCR_EL2.TG0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [13]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SH0, bit [12]

Mask bit for SH0.

SH0	Meaning
0b0	TCR_EL2.SH0 is writeable.
0b1	TCR_EL2.SH0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [11]

Reserved, RES0.

ORGN0, bit [10]

Mask bit for ORGN0.

ORGN0	Meaning
0b0	TCR_EL2.ORGN0 is writeable.
0b1	TCR_EL2.ORGN0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [9]

Reserved, RES0.

IRGN0, bit [8]

Mask bit for IRGN0.

IRGN0	Meaning
0b0	TCR_EL2.IRGN0 is writeable.
0b1	TCR_EL2.IRGN0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- Otherwise, this field resets to an architecturally UNKNOWN value.

Bits [7:1]

Reserved, RES0.

T0SZ, bit [0]

Mask bit for T0SZ.

T0SZ	Meaning
0b0	TCR_EL2.T0SZ is writeable.
0b1	TCR_EL2.T0SZ is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

When ELIsInHost(EL2):

Bits [63:62]

Reserved, RES0.

MTX1, bit [61]

When FEAT_MTE_NO_ADDRESS_TAGS is implemented or FEAT_MTE_CANONICAL_TAGS is implemented:

Mask bit for MTX1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

MTX1	Meaning
0b0	TCR_EL2.MTX1 is writeable.
0b1	TCR_EL2.MTX1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MTX0, bit [60]

When FEAT_MTE_NO_ADDRESS_TAGS is implemented or FEAT_MTE_CANONICAL_TAGS is implemented:

Mask bit for MTX0.

MTX0	Meaning
0b0	TCR_EL2.MTX0 is writeable.
0b1	TCR_EL2.MTX0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DS, bit [59]

When FEAT_LPA2 is implemented:

Mask bit for DS.

DS	Meaning
0b0	TCR_EL2.DS is writeable.
0b1	TCR_EL2.DS is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

TCMA1, bit [58]

When FEAT_MTE2 is implemented:

Mask bit for TCMA1.

TCMA1	Meaning
0b0	TCR_EL2.TCMA1 is writeable.
0b1	TCR_EL2.TCMA1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCMA0, bit [57]

When FEAT_MTE2 is implemented:

Mask bit for TCMA0.

TCMA0	Meaning
0b0	TCR_EL2.TCMA0 is writeable.
0b1	TCR_EL2.TCMA0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0PD1, bit [56]

When FEAT_E0PD is implemented:

Mask bit for E0PD1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E0PD1	Meaning
0b0	TCR_EL2.E0PD1 is writeable.
0b1	TCR_EL2.E0PD1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E0PD0, bit [55]

When FEAT_E0PD is implemented:

Mask bit for E0PD0.

E0PD0	Meaning
0b0	TCR_EL2.E0PD0 is writeable.
0b1	TCR_EL2.E0PD0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NFD1, bit [54]

When FEAT_SVE is implemented:

Mask bit for NFD1.

NFD1	Meaning
0b0	TCR_EL2.NFD1 is writeable.
0b1	TCR_EL2.NFD1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

NFD0, bit [53]

When FEAT_SVE is implemented:

Mask bit for NFD0.

NFD0	Meaning
0b0	TCR_EL2.NFD0 is writeable.
0b1	TCR_EL2.NFD0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBID1, bit [52]

When FEAT_PAuth is implemented:

Mask bit for TBID1.

TBID1	Meaning
0b0	TCR_EL2.TBID1 is writeable.
0b1	TCR_EL2.TBID1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBID0, bit [51]

When FEAT_PAuth is implemented:

Mask bit for TBID0.

TBID0	Meaning
0b0	TCR_EL2.TBID0 is writeable.
0b1	TCR_EL2.TBID0 is not writeable.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU162, bit [50]

When FEAT_HPDS2 is implemented:

Mask bit for HWU162.

HWU162	Meaning
`0b0`	TCR_EL2.HWU162 is writeable.
`0b1`	TCR_EL2.HWU162 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU161, bit [49]

When FEAT_HPDS2 is implemented:

Mask bit for HWU161.

HWU161	Meaning
`0b0`	TCR_EL2.HWU161 is writeable.
`0b1`	TCR_EL2.HWU161 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU160, bit [48]

When FEAT_HPDS2 is implemented:

Mask bit for HWU160.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

HWU160	Meaning
0b0	TCR_EL2.HWU160 is writeable.
0b1	TCR_EL2.HWU160 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU159, bit [47]

When FEAT_HPDS2 is implemented:

Mask bit for HWU159.

HWU159	Meaning
0b0	TCR_EL2.HWU159 is writeable.
0b1	TCR_EL2.HWU159 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU062, bit [46]

When FEAT_HPDS2 is implemented:

Mask bit for HWU062.

HWU062	Meaning
0b0	TCR_EL2.HWU062 is writeable.
0b1	TCR_EL2.HWU062 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

HWU061, bit [45]

When FEAT_HPDS2 is implemented:

Mask bit for HWU061.

HWU061	Meaning
0b0	TCR_EL2.HWU061 is writeable.
0b1	TCR_EL2.HWU061 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU060, bit [44]

When FEAT_HPDS2 is implemented:

Mask bit for HWU060.

HWU060	Meaning
0b0	TCR_EL2.HWU060 is writeable.
0b1	TCR_EL2.HWU060 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU059, bit [43]

When FEAT_HPDS2 is implemented:

Mask bit for HWU059.

HWU059	Meaning
0b0	TCR_EL2.HWU059 is writeable.
0b1	TCR_EL2.HWU059 is not writeable.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD1, bit [42]

When FEAT_HPDS is implemented:

Mask bit for HPD1.

HPD1	Meaning
`0b0`	TCR_EL2.HPD1 is writeable.
`0b1`	TCR_EL2.HPD1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD0, bit [41]

When FEAT_HPDS is implemented:

Mask bit for HPD0.

HPD0	Meaning
`0b0`	TCR_EL2.HPD0 is writeable.
`0b1`	TCR_EL2.HPD0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HD, bit [40]

When FEAT_HAFDBS is implemented:

Mask bit for HD.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

HD	Meaning
0b0	TCR_EL2.HD is writeable.
0b1	TCR_EL2.HD is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HA, bit [39]

When FEAT_HAFDBS is implemented:

Mask bit for HA.

HA	Meaning
0b0	TCR_EL2.HA is writeable.
0b1	TCR_EL2.HA is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBI1, bit [38]

Mask bit for TBI1.

TBI1	Meaning
0b0	TCR_EL2.TBI1 is writeable.
0b1	TCR_EL2.TBI1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

TBI0, bit [37]

Mask bit for TBI0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TBI0	Meaning
0b0	TCR_EL2.TBI0 is writeable.
0b1	TCR_EL2.TBI0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

AS, bit [36]

Mask bit for AS.

AS	Meaning
0b0	TCR_EL2.AS is writeable.
0b1	TCR_EL2.AS is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bits [35:33]

Reserved, RES0.

IPS, bit [32]

Mask bit for IPS.

IPS	Meaning
0b0	TCR_EL2.IPS is writeable.
0b1	TCR_EL2.IPS is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [31]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TG1, bit [30]

Mask bit for TG1.

TG1	Meaning
0b0	TCR_EL2.TG1 is writeable.
0b1	TCR_EL2.TG1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [29]

Reserved, RES0.

SH1, bit [28]

Mask bit for SH1.

SH1	Meaning
0b0	TCR_EL2.SH1 is writeable.
0b1	TCR_EL2.SH1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [27]

Reserved, RES0.

ORGN1, bit [26]

Mask bit for ORGN1.

ORGN1	Meaning
0b0	TCR_EL2.ORGN1 is writeable.
0b1	TCR_EL2.ORGN1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [25]

Reserved, RES0.

IRGN1, bit [24]

Mask bit for IRGN1.

IRGN1	Meaning
0b0	TCR_EL2.IRGN1 is writeable.
0b1	TCR_EL2.IRGN1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

EPD1, bit [23]

Mask bit for EPD1.

EPD1	Meaning
0b0	TCR_EL2.EPD1 is writeable.
0b1	TCR_EL2.EPD1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

A1, bit [22]

Mask bit for A1.

A1	Meaning
0b0	TCR_EL2.A1 is writeable.
0b1	TCR_EL2.A1 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bits [21:17]

Reserved, RES0.

T1SZ, bit [16]

Mask bit for T1SZ.

T1SZ	Meaning
0b0	TCR_EL2.T1SZ is writeable.
0b1	TCR_EL2.T1SZ is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [15]

Reserved, RES0.

TG0, bit [14]

Mask bit for TG0.

TG0	Meaning
0b0	TCR_EL2.TG0 is writeable.
0b1	TCR_EL2.TG0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [13]

Reserved, RES0.

SH0, bit [12]

Mask bit for SH0.

SH0	Meaning
0b0	TCR_EL2.SH0 is writeable.
0b1	TCR_EL2.SH0 is not writeable.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [11]

Reserved, RES0.

ORGN0, bit [10]

Mask bit for ORGN0.

ORGN0	Meaning
`0b0`	TCR_EL2.ORGN0 is writeable.
`0b1`	TCR_EL2.ORGN0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bit [9]

Reserved, RES0.

IRGN0, bit [8]

Mask bit for IRGN0.

IRGN0	Meaning
`0b0`	TCR_EL2.IRGN0 is writeable.
`0b1`	TCR_EL2.IRGN0 is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

EPD0, bit [7]

Mask bit for EPD0.

EPD0	Meaning
`0b0`	TCR_EL2.EPD0 is writeable.
`0b1`	TCR_EL2.EPD0 is not writeable.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bits [6:1]

Reserved, RES0.

T0SZ, bit [0]

Mask bit for T0SZ.

T0SZ	Meaning
0b0	TCR_EL2.T0SZ is writeable.
0b1	TCR_EL2.T0SZ is not writeable.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Accessing TCRMASK_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name `TCRMASK_EL2` or `TCRMASK_EL1` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TCRMASK_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0111	0b010

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = TCRMASK_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TCRMASK_EL2;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MSR TCRMASK_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0111	0b010

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsZero(EffectiveTCRMASK_EL2()) then
        UNDEFINED;
    else
        TCRMASK_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TCRMASK_EL2 = X[t, 64];
```

```
MRS <Xt>, TCRMASK_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0111	0b010

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HFGRTR2_EL2.nTCRMASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x330];
    else
        X[t, 64] = TCRMASK_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = TCRMASK_EL2;
    else
        X[t, 64] = TCRMASK_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TCRMASK_EL1;
```

MSR TCRMASK_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0111	0b010

```
if !(IsFeatureImplemented(FEAT_SRMASK) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HFGWTR2_EL2.nTCRMASK_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.SRMASKEn == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x330] = X[t, 64];
    elsif !IsZero(EffectiveTCRMASK_EL1()) then
        UNDEFINED;
    else
        TCRMASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.SRMASKEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.SRMASKEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        if !IsZero(EffectiveTCRMASK_EL2()) then
            UNDEFINED;
        else
            TCRMASK_EL2 = X[t, 64];
    else
        TCRMASK_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TCRMASK_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.199 TFSR_EL1, Tag Fault Status Register (EL1)

The TFSR_EL1 characteristics are:

Purpose

Holds accumulated Tag Check Faults occurring in EL1 that are not taken precisely.

Configuration

This register is present only when FEAT_MTE_ASYNC is implemented. Otherwise, direct accesses to TFSR_EL1 are UNDEFINED.

Attributes

TFSR_EL1 is a 64-bit register.

Field descriptions

Bits [63:2]

Reserved, RES0.

TF1, bit [1]

When FEAT_MTE_ASYNC is implemented:

Tag Check Fault. Asynchronously set to 1 when a Tag Check Fault using a virtual address with bit[55] == `0b1` occurs.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TF0, bit [0]

When FEAT_MTE_ASYNC is implemented:

Tag Check Fault. Asynchronously set to 1 when a Tag Check Fault using a virtual address with bit[55] == `0b0` occurs.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing TFSR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name `TFSR_EL1` or `TFSR_EL12` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TFSR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0110	0b000

```
if !IsFeatureImplemented(FEAT_MTE_ASYNC) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1')
        ↪then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && !(IsFeatureImplemented(FEAT_MTE2) && HCR_EL2.ATA == '1')
        ↪then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1') then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x190];
    else
        X[t, 64] = TFSR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1')
        ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1') then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = TFSR_EL2;
    else
        X[t, 64] = TFSR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TFSR_EL1;
```

```
MSR TFSR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0110	0b000

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_MTE_ASYNC) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1')
        ↪then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && !(IsFeatureImplemented(FEAT_MTE2) && HCR_EL2.ATA == '1')
        ↪then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1') then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x190] = X[t, 64];
    else
        TFSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1')
        ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1') then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        TFSR_EL2 = X[t, 64];
    else
        TFSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TFSR_EL1 = X[t, 64];
```

MRS <Xt>, TFSR_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b0101	0b0110	0b000

```
if !IsFeatureImplemented(FEAT_MTE_ASYNC) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        if HaveEL(EL3) && EL3SDDUndefPriority() && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA ==
            ↪'1') then
            UNDEFINED;
        elsif EL2Enabled() && !ELIsInHost(EL0) && !(IsFeatureImplemented(FEAT_MTE2) && HCR_EL2.ATA ==
            ↪'1') then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1') then
            if EL3SDDUndef() then
                UNDEFINED;
            else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                X[t, 64] = TFSR_EL1;
        elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1')
            ↪then
            UNDEFINED;
        elsif HaveEL(EL3) && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1') then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = TFSR_EL2;
    elsif PSTATE.EL == EL3 then
        X[t, 64] = TFSR_EL2;
```

MSR TFSR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0101	0b0110	0b000

```
if !IsFeatureImplemented(FEAT_MTE_ASYNC) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        if HaveEL(EL3) && EL3SDDUndefPriority() && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA ==
            ↪'1') then
            UNDEFINED;
        elsif EL2Enabled() && !ELIsInHost(EL0) && !(IsFeatureImplemented(FEAT_MTE2) && HCR_EL2.ATA ==
            ↪'1') then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1') then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TFSR_EL1 = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1')
        ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1') then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TFSR_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL3 then
    TFSR_EL2 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, TFSR_EL12

op0	op1	CRn	CRm	op2
0b11	0b101	0b0101	0b0110	0b000

```
if !IsFeatureImplemented(FEAT_MTE_ASYNC) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x190];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA
            ↪== '1') then
            UNDEFINED;
        elsif HaveEL(EL3) && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1') then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = TFSR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = TFSR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented MSR TFSR_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b0101	0b0110	0b000

```
if !IsFeatureImplemented(FEAT_MTE_ASYNC) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x190] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA
            ↪== '1') then
            UNDEFINED;
        elsif HaveEL(EL3) && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1') then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TFSR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        TFSR_EL1 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.200 TFSR_EL2, Tag Fault Status Register (EL2)

The TFSR_EL2 characteristics are:

Purpose

Holds accumulated Tag Check Faults occurring in EL2 that are not taken precisely.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_MTE_ASYNC is implemented. Otherwise, direct accesses to TFSR_EL2 are UNDEFINED.

Attributes

TFSR_EL2 is a 64-bit register.

Field descriptions

Bits [63:2]

Reserved, RES0.

TF1, bit [1]

When FEAT_MTE_ASYNC is implemented:

Tag Check Fault. Asynchronously set to 1 when a Tag Check Fault using a virtual address with bit[55] == `0b1` occurs.

When the Effective value of HCR_EL2.E2H is not 1, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TF0, bit [0]

When FEAT_MTE_ASYNC is implemented:

Tag Check Fault. Asynchronously set to 1 when a Tag Check Fault using a virtual address with bit[55] == `0b0` occurs.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing TFSR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name `TFSR_EL2` or `TFSR_EL1` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TFSR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0101	0b0110	0b000

```
if !IsFeatureImplemented(FEAT_MTE_ASYNC) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        if HaveEL(EL3) && EL3SDDUndefPriority() && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA ==
            ↪'1') then
            UNDEFINED;
        elsif EL2Enabled() && !ELIsInHost(EL0) && !(IsFeatureImplemented(FEAT_MTE2) && HCR_EL2.ATA ==
            ↪'1') then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1') then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = TFSR_EL1;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1')
        ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1') then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = TFSR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TFSR_EL2;
```

```
MSR TFSR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0101	0b0110	0b000

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_MTE_ASYNC) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        if HaveEL(EL3) && EL3SDDUndefPriority() && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA ==
            ↪'1') then
            UNDEFINED;
        elsif EL2Enabled() && !ELIsInHost(EL0) && !(IsFeatureImplemented(FEAT_MTE2) && HCR_EL2.ATA ==
            ↪'1') then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1') then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TFSR_EL1 = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1')
        ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1') then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TFSR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TFSR_EL2 = X[t, 64];
```

```
MRS <Xt>, TFSR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0110	0b000

```
if !IsFeatureImplemented(FEAT_MTE_ASYNC) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1')
        ↪then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && !(IsFeatureImplemented(FEAT_MTE2) && HCR_EL2.ATA == '1')
        ↪then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1') then
        if EL3SDDUndef() then
            UNDEFINED;
        else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EffectiveHCR_EL2_NVx() IN {'111'} then
            X[t, 64] = NVMem[0x190];
        else
            X[t, 64] = TFSR_EL1;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1')
            ↪then
            UNDEFINED;
        elsif HaveEL(EL3) && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1') then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif ELIsInHost(EL2) then
            X[t, 64] = TFSR_EL2;
        else
            X[t, 64] = TFSR_EL1;
    elsif PSTATE.EL == EL3 then
        X[t, 64] = TFSR_EL1;
```

MSR TFSR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0110	0b000

```
if !IsFeatureImplemented(FEAT_MTE_ASYNC) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1')
        ↪then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && !(IsFeatureImplemented(FEAT_MTE2) && HCR_EL2.ATA == '1')
        ↪then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1') then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x190] = X[t, 64];
    else
        TFSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1')
        ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1') then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        TFSR_EL2 = X[t, 64];
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        TFSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TFSR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.201 TFSR_EL3, Tag Fault Status Register (EL3)

The TFSR_EL3 characteristics are:

Purpose

Holds accumulated Tag Check Faults occurring in EL3 that are not taken precisely.

Configuration

This register is present only when FEAT_MTE_ASYNC is implemented. Otherwise, direct accesses to TFSR_EL3 are UNDEFINED.

Attributes

TFSR_EL3 is a 64-bit register.

Field descriptions

Bits [63:1]

Reserved, RES0.

TF0, bit [0]

When FEAT_MTE_ASYNC is implemented:

Tag Check Fault. Asynchronously set to 1 when a Tag Check Fault using a virtual address with bit[55] == `0b0` occurs.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TFSR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TFSR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0101	0b0110	0b000

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_MTE_ASYNC) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TFSR_EL3;
```

```
MSR TFSR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0101	0b0110	0b000

```
if !IsFeatureImplemented(FEAT_MTE_ASYNC) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TFSR_EL3 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.202 TFSRE0_EL1, Tag Fault Status Register (EL0).

The TFSRE0_EL1 characteristics are:

Purpose

Holds accumulated Tag Check Faults occurring in EL0 that are not taken precisely.

Configuration

This register is present only when FEAT_MTE_ASYNC is implemented. Otherwise, direct accesses to TFSRE0_EL1 are UNDEFINED.

Attributes

TFSRE0_EL1 is a 64-bit register.

Field descriptions

Bits [63:2]

Reserved, RES0.

TF1, bit [1]

When FEAT_MTE_ASYNC is implemented:

Tag Check Fault. Asynchronously set to 1 when a Tag Check Fault using a virtual address with bit[55] == `0b1` occurs.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TF0, bit [0]

When FEAT_MTE_ASYNC is implemented:

Tag Check Fault. Asynchronously set to 1 when a Tag Check Fault using a virtual address with bit[55] == `0b0` occurs.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TFSRE0_EL1

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, TFSRE0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0110	0b001

```
if !IsFeatureImplemented(FEAT_MTE_ASYNC) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1')
        ↪then
        UNDEFINED;
    elsif EL2Enabled() && !ELIsInHost(EL0) && !(IsFeatureImplemented(FEAT_MTE2) && HCR_EL2.ATA == '1')
        ↪then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1') then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = TFSRE0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1')
        ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1') then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = TFSRE0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TFSRE0_EL1;
```

```
MSR TFSRE0_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0110	0b001

```
if !IsFeatureImplemented(FEAT_MTE_ASYNC) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1')
        ↪then
        UNDEFINED;
    elsif EL2Enabled() && !ELIsInHost(EL0) && !(IsFeatureImplemented(FEAT_MTE2) && HCR_EL2.ATA == '1')
        ↪then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1') then
        if EL3SDDUndef() then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TFSRE0_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1')
     ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && !(IsFeatureImplemented(FEAT_MTE2) && SCR_EL3.ATA == '1') then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TFSRE0_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TFSRE0_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.203 TPIDR2_EL0, EL0 Read/Write Software Thread ID Register 2

The TPIDR2_EL0 characteristics are:

Purpose

Provides a location where SME-aware software executing at EL0 can store thread identifying information, for context management purposes.

The PE makes no use of this register.

Configuration

This register is present only when FEAT_SME is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TPIDR2_EL0 are UNDEFINED.

Attributes

TPIDR2_EL0 is a 64-bit register.

Field descriptions

ThreadID, bits [63:0]

Thread identifying information stored by software running at this Exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TPIDR2_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TPIDR2_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0000	0b101

```
if !(IsFeatureImplemented(FEAT_SME) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnTP2 == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL0) && SCTLR_EL1.EnTP2 == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.EnTP2 == '0' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HFGRTR_EL2.nTPIDR2_EL0 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnTP2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = TPIDR2_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnTP2 == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.nTPIDR2_EL0 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnTP2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = TPIDR2_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnTP2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.EnTP2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = TPIDR2_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TPIDR2_EL0;
```

```
MSR TPIDR2_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0000	0b101

```
if !(IsFeatureImplemented(FEAT_SME) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnTP2 == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL0) && SCTLR_EL1.EnTP2 == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && SCTLR_EL2.EnTP2 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HFGWTR_EL2.nTPIDR2_EL0 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnTP2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TPIDR2_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnTP2 == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.nTPIDR2_EL0 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnTP2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TPIDR2_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.EnTP2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.EnTP2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TPIDR2_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TPIDR2_EL0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.204 TPIDR_EL0, EL0 Read/Write Software Thread ID Register

The TPIDR_EL0 characteristics are:

Purpose

Provides a location where software executing at EL0 can store thread identifying information, for OS management purposes.

The PE makes no use of this register.

Configuration

AArch64 System register TPIDR_EL0 bits [31:0] are architecturally mapped to AArch32 System register TPIDRURW[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TPIDR_EL0 are UNDEFINED.

Attributes

TPIDR_EL0 is a 64-bit register.

Field descriptions

ThreadID, bits [63:0]

Thread ID. Thread identifying information stored by software running at this Exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TPIDR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TPIDR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
        ↪SCR_EL3.FGTEn == '1') && HFGRTR_EL2.TPIDR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = TPIDR_EL0;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        ↪HFGRTR_EL2.TPIDR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = TPIDR_EL0;
elsif PSTATE.EL == EL2 then
    X[t, 64] = TPIDR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TPIDR_EL0;
```

```
MSR TPIDR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
        ↪SCR_EL3.FGTEn == '1') && HFGWTR_EL2.TPIDR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TPIDR_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        ↪HFGWTR_EL2.TPIDR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TPIDR_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    TPIDR_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TPIDR_EL0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.205 TPIDR_EL1, EL1 Software Thread ID Register

The TPIDR_EL1 characteristics are:

Purpose

Provides a location where software executing at EL1 can store thread identifying information, for OS management purposes.

The PE makes no use of this register.

Configuration

AArch64 System register TPIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register TPIDRPRW[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TPIDR_EL1 are UNDEFINED.

Attributes

TPIDR_EL1 is a 64-bit register.

Field descriptions

ThreadID, bits [63:0]

Thread ID. Thread identifying information stored by software running at this Exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TPIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TPIDR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b100

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        ↪HFGRTR_EL2.TPIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    else
        X[t, 64] = TPIDR_EL1;
elsif PSTATE.EL == EL2 then
    X[t, 64] = TPIDR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TPIDR_EL1;
```

```
MSR TPIDR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1101	0b0000	0b100

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        ↪HFGWTR_EL2.TPIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TPIDR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    TPIDR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TPIDR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.206 TPIDR_EL2, EL2 Software Thread ID Register

The TPIDR_EL2 characteristics are:

Purpose

Provides a location where software executing at EL2 can store thread identifying information, for OS management purposes.

The PE makes no use of this register.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

AArch64 System register TPIDR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HTPIDR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TPIDR_EL2 are UNDEFINED.

Attributes

TPIDR_EL2 is a 64-bit register.

Field descriptions

ThreadID, bits [63:0]

Thread ID. Thread identifying information stored by software running at this Exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TPIDR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TPIDR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1101	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x090];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = TPIDR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TPIDR_EL2;
```

MSR TPIDR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1101	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x090] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TPIDR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TPIDR_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.207 TPIDR_EL3, EL3 Software Thread ID Register

The TPIDR_EL3 characteristics are:

Purpose

Provides a location where software executing at EL3 can store thread identifying information, for OS management purposes.

The PE makes no use of this register.

Configuration

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TPIDR_EL3 are UNDEFINED.

Attributes

TPIDR_EL3 is a 64-bit register.

Field descriptions

ThreadID, bits [63:0]

Thread ID. Thread identifying information stored by software running at this Exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TPIDR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TPIDR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1101	0b0000	0b010

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TPIDR_EL3;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MSR TPIDR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1101	0b0000	0b010

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.TPIDR_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TPIDR_EL3 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.208 TPIDRRO_EL0, EL0 Read-Only Software Thread ID Register

The TPIDRRO_EL0 characteristics are:

Purpose

Provides a location where software executing at EL1 or higher can store thread identifying information that is visible to software executing at EL0, for OS management purposes.

The PE makes no use of this register.

Configuration

AArch64 System register TPIDRRO_EL0 bits [31:0] are architecturally mapped to AArch32 System register TPIDRURO[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TPIDRRO_EL0 are UNDEFINED.

Attributes

TPIDRRO_EL0 is a 64-bit register.

Field descriptions

ThreadID, bits [63:0]

Thread ID. Thread identifying information stored by software running at this Exception level.

Accessing TPIDRRO_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TPIDRRO_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0000	0b011

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
        ↪SCR_EL3.FGTEn == '1') && HFGRTR_EL2.TPIDRRO_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = TPIDRRO_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        ↪HFGRTR_EL2.TPIDRRO_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    else
        X[t, 64] = TPIDRRO_EL0;
elsif PSTATE.EL == EL2 then
    X[t, 64] = TPIDRRO_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TPIDRRO_EL0;
```

```
MSR TPIDRRO_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0000	0b011

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        ↪HFGWTR_EL2.TPIDRRO_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TPIDRRO_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    TPIDRRO_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TPIDRRO_EL0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.209 TTBR0_EL1, Translation Table Base Register 0 (EL1)

The TTBR0_EL1 characteristics are:

Purpose

Holds the base address of the translation table for the initial lookup for stage 1 of the translation of an address from the lower VA range in the EL1&0 translation regime, and other information for this translation regime.

Configuration

TTBR0_EL1 is a 128-bit register that can also be accessed as a 64-bit value. If it is accessed as a 64-bit register, accesses read and write bits [63:0] and do not modify bits [127:64].

AArch64 System register TTBR0_EL1 bits [63:0] are architecturally mapped to AArch32 System register TTBR0[63:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TTBR0_EL1 are UNDEFINED.

Attributes

TTBR0_EL1 is a:

- 128-bit register when FEAT_D128 is implemented and TCR2_EL1.D128 == ‘1’.
- 64-bit register when FEAT_D128 is not implemented or TCR2_EL1.D128 == ‘0’.

Field descriptions

When FEAT_D128 is implemented and TCR2_EL1.D128 == '1':

Bits [127:88]

Reserved, RES0.

BADDR, bits [87:80, 47:5]

Translation table base address:

- Bits A[55:x] of the stage 1 translation table base address bits are in register bits[87:80, 47:x].
- Bits A[(x-1):0] of the stage 1 translation table base address are zero.

Address bit x is the minimum address bit required to align the translation table to the size of the table. x is calculated based on LOG2(StartTableSize), as described in VMSAv9-128. The smallest permitted value of x is 5.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bits [79:64]

Reserved, RES0.

ASID, bits [63:48]

An ASID for the translation table base address. The TCR_EL1.A1 field selects either TTBR0_EL1.ASID or TTBR1_EL1.ASID.

If the implementation has only 8 bits of ASID, then the upper 8 bits of this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [4:3]

Reserved, RES0.

SKL, bits [2:1]

Skip Level associated with translation table walks using TTBR0_EL1.

This determines the number of levels to be skipped from the regular start level of the stage 1 EL1&0 translation table walks using TTBR0_EL1.

SKL	Meaning
0b00	Skip 0 level from the regular start level.
0b01	Skip 1 level from the regular start level.
0b10	Skip 2 levels from the regular start level.
0b11	Skip 3 levels from the regular start level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When FEAT_TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by TTBR0_EL1 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR0_EL1.CnP is 1.

CnP	Meaning
0b0	The translation table entries pointed to by TTBR0_EL1, for the current translation regime and ASID, are permitted to differ from corresponding entries for TTBR0_EL1 for other PEs in the Inner Shareable domain. This is not affected by: • The value of TTBR0_EL1.CnP on those other PEs. • The value of the current ASID. • If EL2 is implemented and enabled in the current Security state, the value of the current VMID.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CnP	Meaning
0b1	The translation table entries pointed to by TTBR0_EL1 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR0_EL1.CnP is 1 and all of the following apply: • The translation table entries are pointed to by TTBR0_EL1. • The translation tables relate to the same translation regime. • The ASID is the same as the current ASID. • If EL2 is implemented and enabled in the current Security state, the value of the current VMID.

This bit is permitted to be cached in a TLB.

When a TLB combines entries from stage 1 translation and stage 2 translation into a single entry, that entry can only be shared between different PEs if the value of the CnP bit is 1 for both stage 1 and stage 2.

Note

If the value of the TTBR0_EL1.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR0_EL1s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

When FEAT_D128 is not implemented or TCR2_EL1.D128 == '0':

ASID, bits [63:48]

An ASID for the translation table base address. The TCR_EL1.A1 field selects either TTBR0_EL1.ASID or TTBR1_EL1.ASID.

If the implementation has only 8 bits of ASID, then the upper 8 bits of this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

BADDR[47:1], bits [47:1]

Translation table base address:

- Bits A[47:x] of the stage 1 translation table base address bits are in register bits[47:x].
- Bits A[(x-1):0] of the stage 1 translation table base address are zero.

Address bit x is the minimum address bit required to align the translation table to the size of the table. The AArch64 Virtual Memory System Architecture chapter describes how x is calculated based on the value of TCR_EL1.T0SZ, the translation stage, and the translation granule size.

The BADDR field represents a 52-bit address if any of the following apply:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- The value of TCR_EL1.IPS represents an OA size of 52 bits.
- The Effective value of TCR_EL1.DS is 1.

When TTBR0_EL1.BADDR represents a 52-bit addresses, all of the following apply:

- Bits A[51:48] of the stage 1 translation table base address bits are in register bits[5:2].
- Register bit[1] is RES0.
- The smallest permitted value of x is 6.
- When x>6, register bits[(x-1):6] are RES0.

Otherwise, all of the following apply:

- Register bits[(x-1):1] are RES0.
- If 52-bit PA is supported, then bits A[51:48] of the stage 1 translation table base address are treated as `0b0000`.

Note

If BADDR represents a 52-bit address, and the translation table has fewer than eight entries, the table must be aligned to 64 bytes. Otherwise the translation table must be aligned to the size of the table.
For the 64KB granule, if 52-bit PA is not supported, and the value of TCR_EL1.IPS is `0b110` or `0b111`, one of the following IMPLEMENTATION DEFINED behaviors occur:

- BADDR uses the extended format to represent a 52-bit base address.
- BADDR does not use the extended format.

When the value of ID_AA64MMFR0_EL1.PARange indicates that the implementation supports a 56 bit PA size, bits A[55:52] of the stage 1 translation table base address are zero.

If any register bit[47:1] that is defined as RES0 has the value 1 when a translation table walk is done using TTBR0_EL1, then the translation table base address might be misaligned, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Bits A[(x-1):0] of the stage 1 translation table base address are treated as if all the bits are zero. The value read back from the corresponding register bits is either the value written to the register or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When FEAT_TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by TTBR0_EL1 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR0_EL1.CnP is 1.

CnP	Meaning
`0b0`	The translation table entries pointed to by TTBR0_EL1, for the current translation regime and ASID, are permitted to differ from corresponding entries for TTBR0_EL1 for other PEs in the Inner Shareable domain. This is not affected by: • The value of TTBR0_EL1.CnP on those other PEs. • The value of the current ASID. • If EL2 is implemented and enabled in the current Security state, the value of the current VMID.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CnP	Meaning
0b1	The translation table entries pointed to by TTBR0_EL1 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR0_EL1.CnP is 1 and all of the following apply: • The translation table entries are pointed to by TTBR0_EL1. • The translation tables relate to the same translation regime. • The ASID is the same as the current ASID. • If EL2 is implemented and enabled in the current Security state, the value of the current VMID.

This bit is permitted to be cached in a TLB.

When a TLB combines entries from stage 1 translation and stage 2 translation into a single entry, that entry can only be shared between different PEs if the value of the CnP bit is 1 for both stage 1 and stage 2.

Note

If the value of the TTBR0_EL1.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR0_EL1s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TTBR0_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name `TTBR0_EL1` or `TTBR0_EL12` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TTBR0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.TTBR0_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x200];
    else
        X[t, 64] = TTBR0_EL1<63:0>;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        X[t, 64] = TTBR0_EL2<63:0>;
    else
        X[t, 64] = TTBR0_EL1<63:0>;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TTBR0_EL1<63:0>;
```

MSR TTBR0_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.TTBR0_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x200] = X[t, 64];
    else
        TTBR0_EL1<63:0> = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        TTBR0_EL2<63:0> = X[t, 64];
    else
        TTBR0_EL1<63:0> = X[t, 64];
elsif PSTATE.EL == EL3 then
    TTBR0_EL1<63:0> = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, TTBR0_EL12

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x200];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = TTBR0_EL1<63:0>;
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = TTBR0_EL1<63:0>;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented MSR TTBR0_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x200] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        TTBR0_EL1<63:0> = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        TTBR0_EL1<63:0> = X[t, 64];
    else
        UNDEFINED;
```

When FEAT_D128 is implemented MRRS <Xt>, <Xt+1>, TTBR0_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
         ↪&& HFGRTR_EL2.TTBR0_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.D128En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x14);
        elsif EffectiveHCR_EL2_NVx() IN {'111'} then
            X[t, t2, 128] = NVMem128[0x200];
        else
            X[t, t2, 128] = TTBR0_EL1;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x14);
        elsif ELIsInHost(EL2) then
            X[t, t2, 128] = TTBR0_EL2;
        else
            X[t, t2, 128] = TTBR0_EL1;
    elsif PSTATE.EL == EL3 then
        X[t, t2, 128] = TTBR0_EL1;
```

When FEAT_D128 is implemented MSRR TTBR0_EL1, <Xt>, <Xt+1>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
         ↪&& HFGWTR_EL2.TTBR0_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.D128En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem128[0x200] = X[t, t2, 128];
    else
        TTBR0_EL1<127:0> = X[t, t2, 128];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif ELIsInHost(EL2) then
        TTBR0_EL2<127:0> = X[t, t2, 128];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    else
        TTBR0_EL1<127:0> = X[t, t2, 128];
elsif PSTATE.EL == EL3 then
    TTBR0_EL1<127:0> = X[t, t2, 128];
```

When FEAT_D128 is implemented and FEAT_VHE is implemented MRRS <Xt>, <Xt+1>, TTBR0_EL12

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, t2, 128] = NVMem128[0x200];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x14);
        else
            X[t, t2, 128] = TTBR0_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, t2, 128] = TTBR0_EL1;
    else
        UNDEFINED;
```

When FEAT_D128 is implemented and FEAT_VHE is implemented MSRR TTBR0_EL12, <Xt>, <Xt+1>

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem128[0x200] = X[t, t2, 128];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x14);
        else
            TTBR0_EL1<127:0> = X[t, t2, 128];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        TTBR0_EL1<127:0> = X[t, t2, 128];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.210 TTBR0_EL2, Translation Table Base Register 0 (EL2)

The TTBR0_EL2 characteristics are:

Purpose

When the Effective value of HCR_EL2.E2H is not 1, holds the base address of the translation table for the initial lookup for stage 1 of an address translation in the EL2 translation regime, and other information for this translation regime.

When the Effective value of HCR_EL2.E2H is 1, holds the base address of the translation table for the initial lookup for stage 1 of the translation of an address from the lower VA range in the EL2&0 translation regime, and other information for this translation regime.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

TTBR0_EL2 is a 128-bit register that can also be accessed as a 64-bit value. If it is accessed as a 64-bit register, accesses read and write bits [63:0] and do not modify bits [127:64].

AArch64 System register TTBR0_EL2 bits [47:0] are architecturally mapped to AArch32 System register HTTBR[47:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TTBR0_EL2 are UNDEFINED.

Attributes

TTBR0_EL2 is a:

- 128-bit register when FEAT_D128 is implemented, TCR2_EL2.D128 == ‘1’, and ELIsInHost(EL2).
- 64-bit register when FEAT_D128 is not implemented or TCR2_EL2.D128 == ‘0’.

Field descriptions

When FEAT_D128 is implemented, TCR2_EL2.D128 == '1', and ELIsInHost(EL2):

Bits [127:88]

Reserved, RES0.

BADDR[55:5], bits [87:80, 47:5]

Translation table base address:

- Bits A[55:x] of the stage 1 translation table base address bits are in register bits[87:80, 47:x].
- Bits A[(x-1):0] of the stage 1 translation table base address are zero.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Address bit x is the minimum address bit required to align the translation table to the size of the table. x is calculated based on LOG2(StartTableSize), as described in VMSAv9-128. The smallest permitted value of x is 5.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [79:64]

Reserved, RES0.

ASID, bits [63:48]

When FEAT_VHE is implemented:

When the Effective value of HCR_EL2.E2H is 1, it holds an ASID for the translation table base address. The TCR_EL2.A1 field selects either TTBR0_EL2.ASID or TTBR1_EL2.ASID.

If the implementation has only 8 bits of ASID, then the upper 8 bits of this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [4:3]

Reserved, RES0.

SKL, bits [2:1]

Skip Level associated with translation table walks using TTBR0_EL2.

This determines the number of levels to be skipped from the regular start level of the stage 1 EL2&0 translation table walks using TTBR0_EL2.

SKL	Meaning
0b00	Skip 0 level from the regular start level.
0b01	Skip 1 level from the regular start level.
0b10	Skip 2 levels from the regular start level.
0b11	Skip 3 levels from the regular start level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When FEAT_TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by TTBR0_EL2 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR0_EL2.CnP is 1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CnP	Meaning
0b0	The translation table entries pointed to by TTBR0_EL2 for the current translation regime, and ASID if applicable, are permitted to differ from corresponding entries for TTBR0_EL2 for other PEs in the Inner Shareable domain. This is not affected by: • The value of TTBR0_EL2.CnP on those other PEs. • When the current translation regime is the EL2&0 regime, the value of the current ASID.
0b1	The translation table entries pointed to by TTBR0_EL2 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR0_EL2.CnP is 1 and all of the following apply: • The translation table entries are pointed to by TTBR0_EL2. • The translation tables relate to the same translation regime. • If that translation regime is the EL2&0 regime, the ASID is the same as the current ASID.

This bit is permitted to be cached in a TLB.

Note

If the value of the TTBR0_EL2.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR0_EL2s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

When FEAT_D128 is not implemented or TCR2_EL2.D128 == '0':

ASID, bits [63:48]

When FEAT_VHE is implemented:

When the Effective value of HCR_EL2.E2H is not 1, this field is RES0.

When the Effective value of HCR_EL2.E2H is 1, it holds an ASID for the translation table base address. The TCR_EL2.A1 field selects either TTBR0_EL2.ASID or TTBR1_EL2.ASID.

If the implementation has only 8 bits of ASID, then the upper 8 bits of this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

BADDR[47:1], bits [47:1]

Translation table base address:

- Bits A[47:x] of the stage 1 translation table base address bits are in register bits[47:x].
- Bits A[(x-1):0] of the stage 1 translation table base address are zero.

Address bit x is the minimum address bit required to align the translation table to the size of the table. The AArch64 Virtual Memory System Architecture chapter describes how x is calculated based on the value of TCR_EL2.T0SZ, the translation stage, and the translation granule size.

The BADDR field represents a 52-bit address if any of the following apply:

- The value of TCR_EL2.{I}PS represents an OA size of 52 bits.
- The Effective value of TCR_EL2.DS is 1.

When TTBR0_EL2.BADDR represents a 52-bit addresses, all of the following apply:

- Bits A[51:48] of the stage 1 translation table base address bits are in register bits[5:2].
- Register bit[1] is RES0.
- The smallest permitted value of x is 6.
- When x>6, register bits[(x-1):6] are RES0.

Otherwise, all of the following apply:

- Register bits[(x-1):1] are RES0.
- If 52-bit PA is supported, then bits A[51:48] of the stage 1 translation table base address are treated as `0b0000`.

Note

If BADDR represents a 52-bit address, and the translation table has fewer than eight entries, the table must be aligned to 64 bytes. Otherwise the translation table must be aligned to the size of the table.
The OA size specified by TCR_EL2.{I}PS is determined as follows:

- The value of TCR_EL2.PS when the Effective value of HCR_EL2.E2H is not 1.
- The value of TCR_EL2.IPS when the Effective value of HCR_EL2.E2H is 1.

For the 64KB granule, if 52-bit PA is not supported, and the value of TCR_EL2.{I}PS is `0b110` or `0b111`, one of the following IMPLEMENTATION DEFINED behaviors occur:

- BADDR uses the extended format to represent a 52-bit base address.
- BADDR does not use the extended format.

When the value of ID_AA64MMFR0_EL1.PARange indicates that the implementation supports a 56 bit PA size, bits A[55:52] of the stage 1 translation table base address are zero.

If any register bit[47:1] that is defined as RES0 has the value 1 when a translation table walk is done using TTBR0_EL2, then the translation table base address might be misaligned, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Bits A[(x-1):0] of the stage 1 translation table base address are treated as if all the bits are zero. The value read back from the corresponding register bits is either the value written to the register or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When FEAT_TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by TTBR0_EL2 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR0_EL2.CnP is 1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CnP	Meaning
0b0	The translation table entries pointed to by TTBR0_EL2 for the current translation regime, and ASID if applicable, are permitted to differ from corresponding entries for TTBR0_EL2 for other PEs in the Inner Shareable domain. This is not affected by: • The value of TTBR0_EL2.CnP on those other PEs. • When the current translation regime is the EL2&0 regime, the value of the current ASID.
0b1	The translation table entries pointed to by TTBR0_EL2 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR0_EL2.CnP is 1 and all of the following apply: • The translation table entries are pointed to by TTBR0_EL2. • The translation tables relate to the same translation regime. • If that translation regime is the EL2&0 regime, the ASID is the same as the current ASID.

This bit is permitted to be cached in a TLB.

Note

If the value of the TTBR0_EL2.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR0_EL2s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TTBR0_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name `TTBR0_EL2` or `TTBR0_EL1` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TTBR0_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL2 then
    X[t, 64] = TTBR0_EL2<63:0>;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TTBR0_EL2<63:0>;
```

MSR TTBR0_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TTBR0_EL2<63:0> = X[t, 64];
elsif PSTATE.EL == EL3 then
    TTBR0_EL2<63:0> = X[t, 64];
```

MRS <Xt>, TTBR0_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.TTBR0_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x200];
    else
        X[t, 64] = TTBR0_EL1<63:0>;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = TTBR0_EL2<63:0>;
    else
        X[t, 64] = TTBR0_EL1<63:0>;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TTBR0_EL1<63:0>;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MSR TTBR0_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.TTBR0_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x200] = X[t, 64];
    else
        TTBR0_EL1<63:0> = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        TTBR0_EL2<63:0> = X[t, 64];
    else
        TTBR0_EL1<63:0> = X[t, 64];
elsif PSTATE.EL == EL3 then
    TTBR0_EL1<63:0> = X[t, 64];
```

```
When FEAT_D128 is implemented MRRS <Xt>, <Xt+1>, TTBR0_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        X[t, t2, 128] = TTBR0_EL2;
elsif PSTATE.EL == EL3 then
    X[t, t2, 128] = TTBR0_EL2;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_D128 is implemented MSRR TTBR0_EL2, <Xt>, <Xt+1>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        TTBR0_EL2<127:0> = X[t, t2, 128];
elsif PSTATE.EL == EL3 then
    TTBR0_EL2<127:0> = X[t, t2, 128];
```

When FEAT_D128 is implemented MRRS <Xt>, <Xt+1>, TTBR0_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
         ↪&& HFGRTR_EL2.TTBR0_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.D128En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, t2, 128] = NVMem128[0x200];
    else
        X[t, t2, 128] = TTBR0_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif ELIsInHost(EL2) then
        X[t, t2, 128] = TTBR0_EL2;
    else
        X[t, t2, 128] = TTBR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, t2, 128] = TTBR0_EL1;
```

When FEAT_D128 is implemented MSRR TTBR0_EL1, <Xt>, <Xt+1>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
         ↪&& HFGWTR_EL2.TTBR0_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.D128En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem128[0x200] = X[t, t2, 128];
    else
        TTBR0_EL1<127:0> = X[t, t2, 128];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif ELIsInHost(EL2) then
        TTBR0_EL2<127:0> = X[t, t2, 128];
    else
        TTBR0_EL1<127:0> = X[t, t2, 128];
elsif PSTATE.EL == EL3 then
    TTBR0_EL1<127:0> = X[t, t2, 128];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.211 TTBR0_EL3, Translation Table Base Register 0 (EL3)

The TTBR0_EL3 characteristics are:

Purpose

Holds the base address of the translation table for the initial lookup for stage 1 of an address translation in the EL3 translation regime, and other information for this translation regime.

Configuration

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TTBR0_EL3 are UNDEFINED.

Attributes

TTBR0_EL3 is a 64-bit register.

Field descriptions

When FEAT_D128 is implemented and TCR_EL3.D128 == '1':

Bits [63:56]

Reserved, RES0.

BADDR, bits [55:5]

- Bits A[55:x] of the stage 1 translation table base address bits are in register bits[55:x].
- Bits A[(x-1):0] of the stage 1 translation table base address are zero.

Address bit x is the minimum address bit required to align the translation table to the size of the table. x is calculated based on LOG2(StartTableSize), as described in VMSAv9-128. The smallest permitted value of x is 5.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [4:3]

Reserved, RES0.

SKL, bits [2:1]

Skip Level associated with translation table walks using TTBR0_EL3.

This determines the number of levels to be skipped from the regular start level of the stage 1 EL3 translation table walks using TTBR0_EL3.

SKL	Meaning
0b00	Skip 0 level from the regular start level.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SKL	Meaning
0b01	Skip 1 level from the regular start level.
0b10	Skip 2 levels from the regular start level.
0b11	Skip 3 levels from the regular start level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

Common not Private. This bit indicates whether each entry that is pointed to by TTBR0_EL3 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR0_EL3.CnP is 1.

CnP	Meaning
0b0	The translation table entries pointed to by TTBR0_EL3, for the current translation regime, are permitted to differ from corresponding entries for TTBR0_EL3 for other PEs in the Inner Shareable domain. This is not affected by the value of TTBR0_EL3.CnP on those other PEs.
0b1	The translation table entries pointed to by TTBR0_EL3 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR0_EL3.CnP is 1 and the translation table entries are pointed to by TTBR0_EL3.

This bit is permitted to be cached in a TLB.

Note

If the value of the TTBR0_EL3.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR0_EL3s do not point to the same translation table entries the results of translations using TTBR0_EL3 are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_D128 is not implemented or TCR_EL3.D128 == '0':

Bits [63:48]

Reserved, RES0.

BADDR, bits [47:1]

- Bits A[47:x] of the stage 1 translation table base address bits are in register bits[47:x].

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- Bits A[(x-1):0] of the stage 1 translation table base address are zero.

Address bit x is the minimum address bit required to align the translation table to the size of the table. The AArch64 Virtual Memory System Architecture chapter describes how x is calculated based on the value of TCR_EL3.T0SZ, the translation stage, and the translation granule size.

The BADDR field represents a 52-bit address if any of the following apply:

- The value of TCR_EL3.PS represents an OA size of 52 bits.
- The Effective value of TCR_EL3.DS is 1.

When TTBR0_EL3.BADDR represents a 52-bit addresses, all of the following apply:

- Bits A[51:48] of the stage 1 translation table base address bits are in register bits[5:2].
- Register bit[1] is RES0.
- The smallest permitted value of x is 6.
- When x>6, register bits[(x-1):6] are RES0.

Otherwise, all of the following apply:

- Register bits[(x-1):1] are RES0.
- If 52-bit PA is supported, then bits A[51:48] of the stage 1 translation table base address are treated as `0b0000`.

Note

If BADDR represents a 52-bit address, and the translation table has fewer than eight entries, the table must be aligned to 64 bytes. Otherwise the translation table must be aligned to the size of the table.
For the 64KB granule, if 52-bit PA is not supported, and the value of TCR_EL3.PS is `0b110` or `0b111`, one of the following IMPLEMENTATION DEFINED behaviors occur:

- BADDR uses the extended format to represent a 52-bit base address.
- BADDR does not use the extended format.

When the value of ID_AA64MMFR0_EL1.PARange indicates that the implementation supports a 56 bit PA size, bits A[55:52] of the stage 1 translation table base address are zero.

If any register bit[47:1] that is defined as RES0 has the value 1 when a translation table walk is done using TTBR0_EL3, then the translation table base address might be misaligned, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Bits A[(x-1):0] of the stage 1 translation table base address are treated as if all the bits are zero. The value read back from the corresponding register bits is either the value written to the register or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When FEAT_TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by TTBR0_EL3 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR0_EL3.CnP is 1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CnP	Meaning
0b0	The translation table entries pointed to by TTBR0_EL3, for the current translation regime, are permitted to differ from corresponding entries for TTBR0_EL3 for other PEs in the Inner Shareable domain. This is not affected by the value of TTBR0_EL3.CnP on those other PEs.
0b1	The translation table entries pointed to by TTBR0_EL3 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR0_EL3.CnP is 1 and the translation table entries are pointed to by TTBR0_EL3.

This bit is permitted to be cached in a TLB.

Note

If the value of the TTBR0_EL3.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR0_EL3s do not point to the same translation table entries the results of translations using TTBR0_EL3 are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TTBR0_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TTBR0_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0010	0b0000	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TTBR0_EL3;
```

```
MSR TTBR0_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0010	0b0000	0b000

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.TTBR0_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TTBR0_EL3 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.212 TTBR1_EL1, Translation Table Base Register 1 (EL1)

The TTBR1_EL1 characteristics are:

Purpose

Holds the base address of the translation table for the initial lookup for stage 1 of the translation of an address from the higher VA range in the EL1&0 stage 1 translation regime, and other information for this translation regime.

Configuration

TTBR1_EL1 is a 128-bit register that can also be accessed as a 64-bit value. If it is accessed as a 64-bit register, accesses read and write bits [63:0] and do not modify bits [127:64].

AArch64 System register TTBR1_EL1 bits [63:0] are architecturally mapped to AArch32 System register TTBR1[63:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to TTBR1_EL1 are UNDEFINED.

Attributes

TTBR1_EL1 is a:

- 128-bit register when FEAT_D128 is implemented and TCR2_EL1.D128 == ‘1’.
- 64-bit register when FEAT_D128 is not implemented or TCR2_EL1.D128 == ‘0’.

Field descriptions

When FEAT_D128 is implemented and TCR2_EL1.D128 == '1':

Bits [127:88]

Reserved, RES0.

BADDR, bits [87:80, 47:5]

Translation table base address:

- Bits A[55:x] of the stage 1 translation table base address bits are in register bits[87:80, 47:x].
- Bits A[(x-1):0] of the stage 1 translation table base address are zero.

Address bit x is the minimum address bit required to align the translation table to the size of the table. x is calculated based on LOG2(StartTableSize), as described in VMSAv9-128. The smallest permitted value of x is 5.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bits [79:64]

Reserved, RES0.

ASID, bits [63:48]

An ASID for the translation table base address. The TCR_EL1.A1 field selects either TTBR0_EL1.ASID or TTBR1_EL1.ASID.

If the implementation has only 8 bits of ASID, then the upper 8 bits of this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [4:3]

Reserved, RES0.

SKL, bits [2:1]

Skip Level associated with translation table walks using TTBR1_EL1.

This determines the number of levels to be skipped from the regular start level of the stage 1 EL1&0 translation table walks using TTBR1_EL1.

SKL	Meaning
0b00	Skip 0 level from the regular start level.
0b01	Skip 1 level from the regular start level.
0b10	Skip 2 levels from the regular start level.
0b11	Skip 3 levels from the regular start level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When FEAT_TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by TBR1_EL1 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR1_EL1.CnP is 1.

CnP	Meaning
0b0	The translation table entries pointed to by TTBR1_EL1, for the current translation regime and ASID, are permitted to differ from corresponding entries for TTBR1_EL1 for other PEs in the Inner Shareable domain. This is not affected by: • The value of TTBR1_EL1.CnP on those other PEs. • The value of the current ASID. • If EL2 is implemented and enabled in the current Security state, the value of the current VMID.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CnP	Meaning
0b1	The translation table entries pointed to by TTBR1_EL1 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR1_EL1.CnP is 1 and all of the following apply: • The translation table entries are pointed to by TTBR1_EL1. • The translation tables relate to the same translation regime. • The ASID is the same as the current ASID. • If EL2 is implemented and enabled in the current Security state, the value of the current VMID.

This bit is permitted to be cached in a TLB.

When a TLB combines entries from stage 1 translation and stage 2 translation into a single entry, that entry can only be shared between different PEs if the value of the CnP bit is 1 for both stage 1 and stage 2.

Note

If the value of the TTBR1_EL1.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR1_EL1s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

When FEAT_D128 is not implemented or TCR2_EL1.D128 == '0':

ASID, bits [63:48]

An ASID for the translation table base address. The TCR_EL1.A1 field selects either TTBR0_EL1.ASID or TTBR1_EL1.ASID.

If the implementation has only 8 bits of ASID, then the upper 8 bits of this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

BADDR[47:1], bits [47:1]

Translation table base address:

- Bits A[47:x] of the stage 1 translation table base address bits are in register bits[47:x].
- Bits A[(x-1):0] of the stage 1 translation table base address are zero.

Address bit x is the minimum address bit required to align the translation table to the size of the table. The AArch64 Virtual Memory System Architecture chapter describes how x is calculated based on the value of TCR_EL1.T1SZ, the translation stage, and the translation granule size.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The BADDR field represents a 52-bit address if any of the following apply:

- The value of TCR_EL1.IPS represents an OA size of 52 bits.
- The Effective value of TCR_EL1.DS is 1.

When TTBR1_EL1.BADDR represents a 52-bit addresses, all of the following apply:

- Bits A[51:48] of the stage 1 translation table base address bits are in register bits[5:2].
- Register bit[1] is RES0.
- The smallest permitted value of x is 6.
- When x>6, register bits[(x-1):6] are RES0.

Otherwise, all of the following apply:

- Register bits[(x-1):1] are RES0.
- If 52-bit PA is supported, then bits A[51:48] of the stage 1 translation table base address are treated as `0b0000`.

Note

If BADDR represents a 52-bit address, and the translation table has fewer than eight entries, the table must be aligned to 64 bytes. Otherwise the translation table must be aligned to the size of the table.
For the 64KB granule, if 52-bit PA is not supported, and the value of TCR_EL1.IPS is `0b110` or `0b111`, one of the following IMPLEMENTATION DEFINED behaviors occur:

- BADDR uses the extended format to represent a 52-bit base address.
- BADDR does not use the extended format.

When the value of ID_AA64MMFR0_EL1.PARange indicates that the implementation supports a 56 bit PA size, bits A[55:52] of the stage 1 translation table base address are zero.

If any register bit[47:1] that is defined as RES0 has the value 1 when a translation table walk is done using TTBR1_EL1, then the translation table base address might be misaligned, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Bits A[(x-1):0] of the stage 1 translation table base address are treated as if all the bits are zero. The value read back from the corresponding register bits is either the value written to the register or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When FEAT_TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by TBR1_EL1 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR1_EL1.CnP is 1.

CnP	Meaning
`0b0`	The translation table entries pointed to by TTBR1_EL1, for the current translation regime and ASID, are permitted to differ from corresponding entries for TTBR1_EL1 for other PEs in the Inner Shareable domain. This is not affected by: • The value of TTBR1_EL1.CnP on those other PEs. • The value of the current ASID. • If EL2 is implemented and enabled in the current Security state, the value of the current VMID.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CnP	Meaning
0b1	The translation table entries pointed to by TTBR1_EL1 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR1_EL1.CnP is 1 and all of the following apply: • The translation table entries are pointed to by TTBR1_EL1. • The translation tables relate to the same translation regime. • The ASID is the same as the current ASID. • If EL2 is implemented and enabled in the current Security state, the value of the current VMID.

This bit is permitted to be cached in a TLB.

When a TLB combines entries from stage 1 translation and stage 2 translation into a single entry, that entry can only be shared between different PEs if the value of the CnP bit is 1 for both stage 1 and stage 2.

Note

If the value of the TTBR1_EL1.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR1_EL1s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TTBR1_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name `TTBR1_EL1` or `TTBR1_EL12` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TTBR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.TTBR1_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x210];
    else
        X[t, 64] = TTBR1_EL1<63:0>;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = TTBR1_EL2<63:0>;
    else
        X[t, 64] = TTBR1_EL1<63:0>;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TTBR1_EL1<63:0>;
```

MSR TTBR1_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.TTBR1_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x210] = X[t, 64];
    else
        TTBR1_EL1<63:0> = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        TTBR1_EL2<63:0> = X[t, 64];
    else
        TTBR1_EL1<63:0> = X[t, 64];
elsif PSTATE.EL == EL3 then
    TTBR1_EL1<63:0> = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, TTBR1_EL12

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x210];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = TTBR1_EL1<63:0>;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            UNDEFINED;
    elsif PSTATE.EL == EL3 then
        if ELIsInHost(EL2) then
            X[t, 64] = TTBR1_EL1<63:0>;
        else
            UNDEFINED;
```

When FEAT_VHE is implemented MSR TTBR1_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x210] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        TTBR1_EL1<63:0> = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        TTBR1_EL1<63:0> = X[t, 64];
    else
        UNDEFINED;
```

When FEAT_D128 is implemented MRRS <Xt>, <Xt+1>, TTBR1_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
         ↪&& HFGRTR_EL2.TTBR1_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.D128En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x14);
        elsif EffectiveHCR_EL2_NVx() IN {'111'} then
            X[t, t2, 128] = NVMem128[0x210];
        else
            X[t, t2, 128] = TTBR1_EL1;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x14);
        elsif ELIsInHost(EL2) then
            X[t, t2, 128] = TTBR1_EL2;
        else
            X[t, t2, 128] = TTBR1_EL1;
    elsif PSTATE.EL == EL3 then
        X[t, t2, 128] = TTBR1_EL1;
```

When FEAT_D128 is implemented MSRR TTBR1_EL1, <Xt>, <Xt+1>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
         ↪&& HFGWTR_EL2.TTBR1_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.D128En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem128[0x210] = X[t, t2, 128];
    else
        TTBR1_EL1<127:0> = X[t, t2, 128];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        elsif ELIsInHost(EL2) then
            TTBR1_EL2<127:0> = X[t, t2, 128];
        else
            TTBR1_EL1<127:0> = X[t, t2, 128];
    elsif PSTATE.EL == EL3 then
        TTBR1_EL1<127:0> = X[t, t2, 128];
```

When FEAT_D128 is implemented and FEAT_VHE is implemented MRRS <Xt>, <Xt+1>, TTBR1_EL12

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, t2, 128] = NVMem128[0x210];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x14);
        else
            X[t, t2, 128] = TTBR1_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, t2, 128] = TTBR1_EL1;
    else
        UNDEFINED;
```

When FEAT_D128 is implemented and FEAT_VHE is implemented MSRR TTBR1_EL12, <Xt>, <Xt+1>

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem128[0x210] = X[t, t2, 128];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x14);
        else
            TTBR1_EL1<127:0> = X[t, t2, 128];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        TTBR1_EL1<127:0> = X[t, t2, 128];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.213 TTBR1_EL2, Translation Table Base Register 1 (EL2)

The TTBR1_EL2 characteristics are:

Purpose

When the Effective value of HCR_EL2.E2H is 1, holds the base address of the translation table for the initial lookup for stage 1 of the translation of an address from the higher VA range in the EL2&0 translation regime, and other information for this translation regime.

Note

When the Effective value of HCR_EL2.E2H is not 1, the contents of this register are ignored by the PE, except for a direct read or write of the register.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

TTBR1_EL2 is a 128-bit register that can also be accessed as a 64-bit value. If it is accessed as a 64-bit register, accesses read and write bits [63:0] and do not modify bits [127:64].

This register is present only when FEAT_VHE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TTBR1_EL2 are UNDEFINED.

Attributes

TTBR1_EL2 is a:

- 128-bit register when FEAT_D128 is implemented, TCR2_EL2.D128 == ‘1’, and ELIsInHost(EL2).
- 64-bit register when FEAT_D128 is not implemented or TCR2_EL2.D128 == ‘0’.

Field descriptions

When FEAT_D128 is implemented, TCR2_EL2.D128 == '1', and ELIsInHost(EL2):

Bits [127:88]

Reserved, RES0.

BADDR, bits [87:80, 47:5]

Translation table base address:

- Bits A[55:x] of the stage 1 translation table base address bits are in register bits[87:80, 47:x].
- Bits A[(x-1):0] of the stage 1 translation table base address are zero.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Address bit x is the minimum address bit required to align the translation table to the size of the table. x is calculated based on LOG2(StartTableSize), as described in VMSAv9-128. The smallest permitted value of x is 5.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [79:64]

Reserved, RES0.

ASID, bits [63:48]

An ASID for the translation table base address. The TCR_EL2.A1 field selects either TTBR0_EL2.ASID or TTBR1_EL2.ASID.

If the implementation has only 8 bits of ASID, then the upper 8 bits of this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [4:3]

Reserved, RES0.

SKL, bits [2:1]

Skip Level associated with translation table walks using TTBR1_EL2.

This determines the number of levels to be skipped from the regular start level of the stage 1 EL2&0 translation table walks using TTBR1_EL2.

SKL	Meaning
`0b00`	Skip 0 level from the regular start level.
`0b01`	Skip 1 level from the regular start level.
`0b10`	Skip 2 levels from the regular start level.
`0b11`	Skip 3 levels from the regular start level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When FEAT_TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by TBR1_EL2 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR1_EL2.CnP is 1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CnP	Meaning
0b0	The translation table entries pointed to by TTBR1_EL2 for the current ASID are permitted to differ from corresponding entries for TTBR1_EL2 for other PEs in the Inner Shareable domain. This is not affected by: • The value of TTBR1_EL2.CnP on those other PEs. • The value of the current ASID.
0b1	The translation table entries pointed to by TTBR1_EL2 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR1_EL2.CnP is 1 and all of the following apply: • The translation table entries are pointed to by TTBR1_EL2. • The ASID is the same as the current ASID.

This bit is permitted to be cached in a TLB.

Note

- TTBR1_EL2 is accessible only when the Effective value of HCR_EL2.E2H is 1, meaning the current translation regime is the EL2&0 regime.
- If the value of the TTBR1_EL2.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR1_EL2s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

When FEAT_D128 is not implemented or TCR2_EL2.D128 == '0':

ASID, bits [63:48]

An ASID for the translation table base address. The TCR_EL2.A1 field selects either TTBR0_EL2.ASID or TTBR1_EL2.ASID.

If the implementation has only 8 bits of ASID, then the upper 8 bits of this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

BADDR[47:1], bits [47:1]

Translation table base address:

- Bits A[47:x] of the stage 1 translation table base address bits are in register bits[47:x].
- Bits A[(x-1):0] of the stage 1 translation table base address are zero.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Address bit x is the minimum address bit required to align the translation table to the size of the table. The AArch64 Virtual Memory System Architecture chapter describes how x is calculated based on the value of TCR_EL2.T1SZ, the translation stage, and the translation granule size.

The BADDR field represents a 52-bit address if any of the following apply:

- The value of TCR_EL2.{I}PS represents an OA size of 52 bits.
- The Effective value of TCR_EL2.DS is 1.

When TTBR1_EL2.BADDR represents a 52-bit addresses, all of the following apply:

- Bits A[51:48] of the stage 1 translation table base address bits are in register bits[5:2].
- Register bit[1] is RES0.
- The smallest permitted value of x is 6.
- When x>6, register bits[(x-1):6] are RES0.

Otherwise, all of the following apply:

- Register bits[(x-1):1] are RES0.
- If 52-bit PA is supported, then bits A[51:48] of the stage 1 translation table base address are treated as `0b0000`.

Note

If BADDR represents a 52-bit address, and the translation table has fewer than eight entries, the table must be aligned to 64 bytes. Otherwise the translation table must be aligned to the size of the table.
The OA size specified by TCR_EL2.{I}PS is determined as follows:

- The value of TCR_EL2.PS when the Effective value of HCR_EL2.E2H is not 1.
- The value of TCR_EL2.IPS when the Effective value of HCR_EL2.E2H is 1.

For the 64KB granule, if 52-bit PA is not supported, and the value of TCR_EL2.{I}PS is `0b110` or `0b111`, one of the following IMPLEMENTATION DEFINED behaviors occur:

- BADDR uses the extended format to represent a 52-bit base address.
- BADDR does not use the extended format.

When the value of ID_AA64MMFR0_EL1.PARange indicates that the implementation supports a 56 bit PA size, bits A[55:52] of the stage 1 translation table base address are zero.

If any register bit[47:1] that is defined as RES0 has the value 1 when a translation table walk is done using TTBR1_EL2, then the translation table base address might be misaligned, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Bits A[(x-1):0] of the stage 1 translation table base address are treated as if all the bits are zero. The value read back from the corresponding register bits is either the value written to the register or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When FEAT_TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by TBR1_EL2 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR1_EL2.CnP is 1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CnP	Meaning
0b0	The translation table entries pointed to by TTBR1_EL2 for the current ASID are permitted to differ from corresponding entries for TTBR1_EL2 for other PEs in the Inner Shareable domain. This is not affected by: • The value of TTBR1_EL2.CnP on those other PEs. • The value of the current ASID.
0b1	The translation table entries pointed to by TTBR1_EL2 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR1_EL2.CnP is 1 and all of the following apply: • The translation table entries are pointed to by TTBR1_EL2. • The ASID is the same as the current ASID.

This bit is permitted to be cached in a TLB.

Note

- TTBR1_EL2 is accessible only when the Effective value of HCR_EL2.E2H is 1, meaning the current translation regime is the EL2&0 regime.
- If the value of the TTBR1_EL2.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR1_EL2s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TTBR1_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name `TTBR1_EL2` or `TTBR1_EL1` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TTBR1_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0000	0b001

```
if !(IsFeatureImplemented(FEAT_VHE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = TTBR1_EL2<63:0>;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TTBR1_EL2<63:0>;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MSR TTBR1_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0000	0b001

```
if !(IsFeatureImplemented(FEAT_VHE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TTBR1_EL2<63:0> = X[t, 64];
elsif PSTATE.EL == EL3 then
    TTBR1_EL2<63:0> = X[t, 64];
```

```
MRS <Xt>, TTBR1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.TTBR1_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x210];
    else
        X[t, 64] = TTBR1_EL1<63:0>;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = TTBR1_EL2<63:0>;
    else
        X[t, 64] = TTBR1_EL1<63:0>;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TTBR1_EL1<63:0>;
```

```
MSR TTBR1_EL1, <Xt>
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.TTBR1_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x210] = X[t, 64];
    else
        TTBR1_EL1<63:0> = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        TTBR1_EL2<63:0> = X[t, 64];
    else
        TTBR1_EL1<63:0> = X[t, 64];
elsif PSTATE.EL == EL3 then
    TTBR1_EL1<63:0> = X[t, 64];
```

When FEAT_D128 is implemented MRRS <Xt>, <Xt+1>, TTBR1_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0000	0b001

```
if !(IsFeatureImplemented(FEAT_VHE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        X[t, t2, 128] = TTBR1_EL2;
elsif PSTATE.EL == EL3 then
    X[t, t2, 128] = TTBR1_EL2;
```

When FEAT_D128 is implemented MSRR TTBR1_EL2, <Xt>, <Xt+1>

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0000	0b001

```
if !(IsFeatureImplemented(FEAT_VHE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        TTBR1_EL2<127:0> = X[t, t2, 128];
elsif PSTATE.EL == EL3 then
    TTBR1_EL2<127:0> = X[t, t2, 128];
```

When FEAT_D128 is implemented MRRS <Xt>, <Xt+1>, TTBR1_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
         ↪&& HFGRTR_EL2.TTBR1_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.D128En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, t2, 128] = NVMem128[0x210];
    else
        X[t, t2, 128] = TTBR1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif ELIsInHost(EL2) then
        X[t, t2, 128] = TTBR1_EL2;
    else
        X[t, t2, 128] = TTBR1_EL1;
elsif PSTATE.EL == EL3 then
    X[t, t2, 128] = TTBR1_EL1;
```

When FEAT_D128 is implemented MSRR TTBR1_EL1, <Xt>, <Xt+1>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
         ↪&& HFGWTR_EL2.TTBR1_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif EL2Enabled() && (!IsHCRXEL2Enabled() || HCRX_EL2.D128En == '0') then
        AArch64.SystemAccessTrap(EL2, 0x14);
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem128[0x210] = X[t, t2, 128];
    else
        TTBR1_EL1<127:0> = X[t, t2, 128];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    elsif ELIsInHost(EL2) then
        TTBR1_EL2<127:0> = X[t, t2, 128];
    else
        TTBR1_EL1<127:0> = X[t, t2, 128];
elsif PSTATE.EL == EL3 then
    TTBR1_EL1<127:0> = X[t, t2, 128];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.214 VBAR_EL1, Vector Base Address Register (EL1)

The VBAR_EL1 characteristics are:

Purpose

Holds the vector base address for any exception that is taken to EL1.

Configuration

AArch64 System register VBAR_EL1 bits [31:0] are architecturally mapped to AArch32 System register VBAR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to VBAR_EL1 are UNDEFINED.

Attributes

VBAR_EL1 is a 64-bit register.

Field descriptions

VBA, bits [63:11]

Vector Base Address. Base address of the exception vectors for exceptions taken to EL1.

Note

If the implementation supports FEAT_LVA3, then:

- If tagged addresses are not being used, bits [63:56] of VBAR_EL1 must be the same or else the use of the vector address will result in a recursive exception.

Otherwise:
If the implementation supports FEAT_LVA, then:

- If tagged addresses are being used, bits [55:52] of VBAR_EL1 must be the same or else the use of the vector address will result in a recursive exception.
- If tagged addresses are not being used, bits [63:52] of VBAR_EL1 must be the same or else the use of the vector address will result in a recursive exception.

If the implementation does not support FEAT_LVA, then:

- If tagged addresses are being used, bits [55:48] of VBAR_EL1 must be the same or else the use of the vector address will result in a recursive exception.
- If tagged addresses are not being used, bits [63:48] of VBAR_EL1 must be the same or else the use of the vector address will result in a recursive exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [10:0]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing VBAR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name `VBAR_EL1` or `VBAR_EL12` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, VBAR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.VBAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x250];
    else
        X[t, 64] = VBAR_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = VBAR_EL2;
    else
        X[t, 64] = VBAR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = VBAR_EL1;
```

```
MSR VBAR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.VBAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x250] = X[t, 64];
    else
        VBAR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    if ELIsInHost(EL2) then
        VBAR_EL2 = X[t, 64];
    else
        VBAR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    VBAR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, VBAR_EL12

op0	op1	CRn	CRm	op2
0b11	0b101	0b1100	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x250];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = VBAR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = VBAR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented MSR VBAR_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b1100	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x250] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        VBAR_EL1 = X[t, 64];
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        VBAR_EL1 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.215 VBAR_EL2, Vector Base Address Register (EL2)

The VBAR_EL2 characteristics are:

Purpose

Holds the vector base address for any exception that is taken to EL2.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

AArch64 System register VBAR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HVBAR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to VBAR_EL2 are UNDEFINED.

Attributes

VBAR_EL2 is a 64-bit register.

Field descriptions

VBA, bits [63:11]

Vector Base Address. Base address of the exception vectors for exceptions taken to EL2.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Note

If FEAT_LVA3 is implemented:

- If the Effective value of HCR_EL2.E2H is 1:
  - If tagged addresses are not being used, bits [63:56] of VBAR_EL2 must be the same or else the use of the vector address will result in a recursive exception.
- If the Effective value of HCR_EL2.E2H is not 1:
  - If tagged addresses are not being used, bits [63:56] of VBAR_EL2 must be 0 or else the use of the vector address will result in a recursive exception.

Otherwise :
If FEAT_LVA is implemented:

- If the Effective value of HCR_EL2.E2H is 1:
  - If tagged addresses are being used, bits [55:52] of VBAR_EL2 must be the same or else the use of the vector address will result in a recursive exception.
  - If tagged addresses are not being used, bits [63:52] of VBAR_EL2 must be the same or else the use of the vector address will result in a recursive exception.
- If the Effective value of HCR_EL2.E2H is not 1:
  - If tagged addresses are being used, bits [55:52] of VBAR_EL2 must be 0 or else the use of the vector address will result in a recursive exception.
  - If tagged addresses are not being used, bits [63:52] of VBAR_EL2 must be 0 or else the use of the vector address will result in a recursive exception.

If FEAT_LVA is not implemented:

- If the Effective value of HCR_EL2.E2H is 1:
  - If tagged addresses are being used, bits [55:48] of VBAR_EL2 must be the same or else the use of the vector address will result in a recursive exception.
  - If tagged addresses are not being used, bits [63:48] of VBAR_EL2 must be the same or else the use of the vector address will result in a recursive exception.
- If the Effective value of HCR_EL2.E2H is not 1:
  - If tagged addresses are being used, bits [55:48] of VBAR_EL2 must be 0 or else the use of the vector address will result in a recursive exception.
  - If tagged addresses are not being used, bits [63:48] of VBAR_EL2 must be 0 or else the use of the vector address will result in a recursive exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [10:0]

Reserved, RES0.

Accessing VBAR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name `VBAR_EL2` or `VBAR_EL1` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, VBAR_EL2
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b100	0b1100	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = VBAR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = VBAR_EL2;
```

MSR VBAR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1100	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VBAR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    VBAR_EL2 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, VBAR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
        ↪&& HFGRTR_EL2.VBAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x250];
    else
        X[t, 64] = VBAR_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = VBAR_EL2;
    else
        X[t, 64] = VBAR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = VBAR_EL1;
```

When FEAT_VHE is implemented MSR VBAR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
         ↪&& HFGWTR_EL2.VBAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x250] = X[t, 64];
    else
        VBAR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        VBAR_EL2 = X[t, 64];
    else
        VBAR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    VBAR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.216 VBAR_EL3, Vector Base Address Register (EL3)

The VBAR_EL3 characteristics are:

Purpose

Holds the vector base address for any exception that is taken to EL3.

Configuration

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to VBAR_EL3 are UNDEFINED.

Attributes

VBAR_EL3 is a 64-bit register.

Field descriptions

VBA, bits [63:11]

Vector Base Address. Base address of the exception vectors for exceptions taken to EL3.

Note

If the implementation supports FEAT_LVA3, then:

- If tagged addresses are not being used, bits [63:56] of VBAR_EL3 must be 0 or else the use of the vector address will result in a recursive exception.

Otherwise:
If the implementation supports FEAT_LVA, then:

- If tagged addresses are being used, bits [55:52] of VBAR_EL3 must be 0 or else the use of the vector address will result in a recursive exception.
- If tagged addresses are not being used, bits [63:52] of VBAR_EL3 must be 0 or else the use of the vector address will result in a recursive exception.

If the implementation does not support FEAT_LVA, then:

- If tagged addresses are being used, bits [55:48] of VBAR_EL3 must be 0 or else the use of the vector address will result in a recursive exception.
- If tagged addresses are not being used, bits [63:48] of VBAR_EL3 must be 0 or else the use of the vector address will result in a recursive exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [10:0]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing VBAR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, VBAR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1100	0b0000	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = VBAR_EL3;
```

```
MSR VBAR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1100	0b0000	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.VBAR_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        VBAR_EL3 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.217 VMECID_A_EL2, Alternate MECID for EL1&0 stage 2 translation regime

The VMECID_A_EL2 characteristics are:

Purpose

Alternate MECID for EL1&0 stage 2 translation regime.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_MEC is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to VMECID_A_EL2 are UNDEFINED.

Attributes

VMECID_A_EL2 is a 64-bit register.

Field descriptions

Bits [63:16]

Reserved, RES0.

MECID, bits [15:0]

If MECIDWidth is less than 16 bits, bits[15:MECIDWidth] are RES0.

Note

MECIDWidth is defined in MECIDR_EL2.MECIDWidthm1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing VMECID_A_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, VMECID_A_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b1001	0b001

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_MEC) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Realm) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.MECEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.MECEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = VMECID_A_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = VMECID_A_EL2;
```

MSR VMECID_A_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b1001	0b001

```
if !(IsFeatureImplemented(FEAT_MEC) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Realm) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.MECEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.MECEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        VMECID_A_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    VMECID_A_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.218 VMECID_P_EL2, Primary MECID for EL1&0 stage 2 translation regime

The VMECID_P_EL2 characteristics are:

Purpose

Primary MECID for EL1&0 stage 2 translation regime.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_MEC is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to VMECID_P_EL2 are UNDEFINED.

Attributes

VMECID_P_EL2 is a 64-bit register.

Field descriptions

Bits [63:16]

Reserved, RES0.

MECID, bits [15:0]

If MECIDWidth is less than 16 bits, bits[15:MECIDWidth] are RES0.

Note

MECIDWidth is defined in MECIDR_EL2.MECIDWidthm1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing VMECID_P_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, VMECID_P_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b1001	0b000

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_MEC) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Realm) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.MECEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.MECEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = VMECID_P_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = VMECID_P_EL2;
```

MSR VMECID_P_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b1001	0b000

```
if !(IsFeatureImplemented(FEAT_MEC) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Realm) then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.MECEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.MECEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        VMECID_P_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    VMECID_P_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.219 VMPIDR_EL2, Virtualization Multiprocessor ID Register

The VMPIDR_EL2 characteristics are:

Purpose

Holds the value of the Virtualization Multiprocessor ID. This is the value returned by EL1 reads of MPIDR_EL1, which in a multiprocessor system, provides an additional PE identification system.

Configuration

If EL2 is not implemented, reads of this register return the value of the MPIDR_EL1 and writes to the register are ignored.

This register has no effect if EL2 is not enabled in the current Security state.

AArch64 System register VMPIDR_EL2 bits [31:0] are architecturally mapped to AArch32 System register VMPIDR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to VMPIDR_EL2 are UNDEFINED.

Attributes

VMPIDR_EL2 is a 64-bit register.

Field descriptions

Bits [63:40]

Reserved, RES0.

Aff3, bits [39:32]

Affinity level 3. See the description of VMPIDR_EL2.Aff0 for more information.

Aff3 is not supported in AArch32 state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [31]

Reserved, RES1.

U, bit [30]

Indicates a Uniprocessor system, as distinct from PE 0 in a multiprocessor system.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

U	Meaning
0b0	Processor is part of a multiprocessor system.
0b1	Processor is part of a uniprocessor system.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [29:25]

Reserved, RES0.

MT, bit [24]

Indicates whether the lowest level of affinity consists of logical PEs that are implemented using a multithreading type approach. See the description of VMPIDR_EL2.Aff0 for more information about affinity levels.

MT	Meaning
0b0	Performance of PEs at the lowest affinity level is largely independent.
0b1	Performance of PEs at the lowest affinity level is very interdependent.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Aff2, bits [23:16]

Affinity level 2. See the description of VMPIDR_EL2.Aff0 for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Aff1, bits [15:8]

Affinity level 1. See the description of VMPIDR_EL2.Aff0 for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Aff0, bits [7:0]

Affinity level 0.

The value of the MPIDR.{Aff2, Aff1, Aff0} or MPIDR_EL1.{Aff3, Aff2, Aff1, Aff0} set of fields of each PE must be unique within the system as a whole.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing VMPIDR_EL2

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

MRS <Xt>, VMPIDR_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b0000	0b0000	0b101

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x050];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = VMPIDR_EL2;
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        X[t, 64] = MPIDR_EL1;
    else
        X[t, 64] = VMPIDR_EL2;
```

MSR VMPIDR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0000	0b0000	0b101

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x050] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VMPIDR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        return;
    else
        VMPIDR_EL2 = X[t, 64];
```

MRS <Xt>, MPIDR_EL1

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0000	0b101

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
     ↪HFGRTR_EL2.MPIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() then
        X[t, 64] = VMPIDR_EL2;
    else
        X[t, 64] = MPIDR_EL1;
elsif PSTATE.EL == EL2 then
    X[t, 64] = MPIDR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPIDR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.220 VNCR_EL2, Virtual Nested Control Register

The VNCR_EL2 characteristics are:

Purpose

When FEAT_NV2 is implemented, holds the base address that is used to define the memory location that is accessed by transformed reads and writes of System registers.

Configuration

This register has no effect if EL2 is not enabled in the current Security state.

This register is present only when FEAT_NV2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to VNCR_EL2 are UNDEFINED.

Attributes

VNCR_EL2 is a 64-bit register.

Field descriptions

BADDR, bits [63:12]

Base Address. When a register read/write is transformed to be a Load or Store, the address of the load/store is to VNCR_EL2.BADDR:Offset<11:0>.

Bits[63:n] are RESS where n is one of the following:

- If FEAT_LVA3 is implemented, n is 57.
- Otherwise, if FEAT_LVA is implemented, n is 53.
- If FEAT_LVA is not implemented, n is 49.

If the bits marked as RESS do not all have the same value as bit[n-1], then there is a CONSTRAINED UNPREDICTABLE choice between:

- Generating a Translation fault when translating the transformed register read/write.
- The bits behave as the same value as bit[n-1] for all purposes other than reading back the register.
- The bits are treated as the same value as bit[n-1] for all purposes.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:0]

Reserved, RES0.

Accessing VNCR_EL2

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, VNCR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0010	0b000

```
if !(IsFeatureImplemented(FEAT_NV2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x0B0];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = VNCR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = VNCR_EL2;
```

```
MSR VNCR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0010	0b000

```
if !(IsFeatureImplemented(FEAT_NV2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x0B0] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VNCR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    VNCR_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.221 VPIDR_EL2, Virtualization Processor ID Register

The VPIDR_EL2 characteristics are:

Purpose

Holds the value of the Virtualization Processor ID. This is the value returned by EL1 reads of MIDR_EL1.

Configuration

If EL2 is not implemented, reads of this register return the value of the MIDR_EL1 and writes to the register are ignored.

This register has no effect if EL2 is not enabled in the current Security state.

AArch64 System register VPIDR_EL2 bits [31:0] are architecturally mapped to AArch32 System register VPIDR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to VPIDR_EL2 are UNDEFINED.

Attributes

VPIDR_EL2 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

Implementer, bits [31:24]

The Implementer code. This field must hold an implementer code that has been assigned by Arm.

Implementer	Meaning
0x00	Reserved for software use.
0x41	Arm Limited.
0x42	Broadcom Corporation.
0x43	Cavium Inc.
0x44	Digital Equipment Corporation.
0x46	Fujitsu Ltd.
0x49	Infineon Technologies AG.
0x4D	Motorola or Freescale Semiconductor Inc.
0x4E	NVIDIA Corporation.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Implementer	Meaning
0x50	Applied Micro Circuits Corporation.
0x51	Qualcomm Inc.
0x56	Marvell International Ltd.
0x69	Intel Corporation.
0xC0	Ampere Computing.

Arm can assign codes that are not published in this manual. All values not assigned by Arm are reserved and must not be used.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Variant, bits [23:20]

An IMPLEMENTATION DEFINED variant number. Typically, this field is used to distinguish between different product variants, or major revisions of a product.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Architecture, bits [19:16]

Architecture version.

Architecture	Meaning
0b0001	Armv4.
0b0010	Armv4T.
0b0011	Armv5 (obsolete).
0b0100	Armv5T.
0b0101	Armv5TE.
0b0110	Armv5TEJ.
0b0111	Armv6.
0b1111	Architectural features are individually identified in the ID_* registers.

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PartNum, bits [15:4]

An IMPLEMENTATION DEFINED primary part number for the device.

On processors implemented by Arm, if the top four bits of the primary part number are 0x0 or 0x7, the variant and architecture are encoded differently.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Revision, bits [3:0]

An IMPLEMENTATION DEFINED revision number for the device.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing VPIDR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, VPIDR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0000	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x088];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = VPIDR_EL2;
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        X[t, 64] = MIDR_EL1;
    else
        X[t, 64] = VPIDR_EL2;
```

```
MSR VPIDR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0000	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x088] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VPIDR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    if !HaveEL(EL2) then
        return;
    else
        VPIDR_EL2 = X[t, 64];
```

MRS <Xt>, MIDR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0000	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UnimplementedIDRegister();
elsif PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
     ↪HFGRTR_EL2.MIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() then
        X[t, 64] = VPIDR_EL2;
    else
        X[t, 64] = MIDR_EL1;
elsif PSTATE.EL == EL2 then
    X[t, 64] = MIDR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MIDR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.222 VSTCR_EL2, Virtualization Secure Translation Control Register

The VSTCR_EL2 characteristics are:

Purpose

The control register for stage 2 of the Secure EL1&0 translation regime.

Configuration

This register has no effect if EL2 is not enabled in the current Security state.

This register is present only when FEAT_SEL2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to VSTCR_EL2 are UNDEFINED.

Attributes

VSTCR_EL2 is a 64-bit register.

Field descriptions

IsProfile(PROFILE_A)

Any of the bits in VSTCR_EL2 are permitted to be cached in a TLB.

Bits [63:34]

Reserved, RES0.

SL2, bit [33]

When FEAT_LPA2 is implemented and (FEAT_D128 is not implemented or VTCR_EL2.D128 == '0'):

Starting level of the Secure stage 2 translation lookup controlled by VSTCR_EL2.

If VTCR_EL2.DS == 1, then VSTCR_EL2.SL2, in combination with VSTCR_EL2.SL0, gives encodings for the Secure stage 2 translation table walk initial lookup level.

If VTCR_EL2.DS == 0, then VSTCR_EL2.SL2 is RES0.

If the translation granule size is not 4KB, then this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [32]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bit [31]

Reserved, RES1.

SA, bit [30]

Secure stage 2 translation output address space.

SA	Meaning
0b0	All stage 2 translations for the Secure IPA space access the Secure PA space.
0b1	All stage 2 translations for the Secure IPA space access the Non-secure PA space.

When the value of VSTCR_EL2.SW is 1, this bit behaves as 1 for all purposes other than reading back the value of the bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SW, bit [29]

Secure stage 2 translation address space.

SW	Meaning
0b0	All stage 2 translation table walks for the Secure IPA space are to the Secure PA space.
0b1	All stage 2 translation table walks for the Secure IPA space are to the Non-secure PA space.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [28:16]

Reserved, RES0.

TG0, bits [15:14]

Secure stage 2 granule size for VSTTBR_EL2.

TG0	Meaning
0b00	4KB.
0b01	64KB.
0b10	16KB.

Other values are reserved.

If FEAT_GTG is implemented, ID_AA64MMFR0_EL1.{TGran4_2, TGran16_2, TGran64_2} indicate which granule sizes are supported for stage 2 translation.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

If FEAT_GTG is not implemented, ID_AA64MMFR0_EL1.{TGran4, TGran16, TGran64} indicate which granule sizes are supported.

If the value is programmed to either a reserved value, or a size that has not been implemented, then for all purposes other than read back from this register, the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [13:8]

Reserved, RES0.

SL0, bits [7:6]

When FEAT_TTST is implemented and (FEAT_D128 is not implemented or VTCR_EL2.D128 == '0'):

Starting level of the Secure stage 2 translation lookup, controlled by VSTCR_EL2. The meaning of this field depends on the value of VSTCR_EL2.TG0.

SL0	Meaning
`0b00`	If VSTCR_EL2.TG0 is `0b00` (4KB granule): • If FEAT_LPA2 is not implemented, start at level 2. • If FEAT_LPA2 is implemented and VSTCR_EL2.SL2 is `0b0`, start at level 2. • If FEAT_LPA2 is implemented and VSTCR_EL2.SL2 is `0b1`, start at level -1. If VSTCR_EL2.TG0 is `0b10` (16KB granule) or `0b01` (64KB granule), start at level 3.
`0b01`	If VSTCR_EL2.TG0 is `0b00` (4KB granule): • If FEAT_LPA2 is not implemented, start at level 1. • If FEAT_LPA2 is implemented and VSTCR_EL2.SL2 is `0b0`, start at level 1. • If FEAT_LPA2 is implemented, the combination of VSTCR_EL2.SL0 == 01 and VSTCR_EL2.SL2 == 1 is reserved. If VSTCR_EL2.TG0 is `0b10` (16KB granule) or `0b01` (64KB granule), start at level 2.
`0b10`	If VSTCR_EL2.TG0 is `0b00` (4KB granule): • If FEAT_LPA2 is not implemented, start at level 0. • If FEAT_LPA2 is implemented and VSTCR_EL2.SL2 is `0b0`, start at level 0. • If FEAT_LPA2 is implemented, the combination of VSTCR_EL2.SL0 == 10 and VSTCR_EL2.SL2 == 1 is reserved. If VSTCR_EL2.TG0 is `0b10` (16KB granule) or `0b01` (64KB granule), start at level 1.
`0b11`	If VSTCR_EL2.TG0 is `0b00` (4KB granule): • If FEAT_LPA2 is not implemented, start at level 3. • If FEAT_LPA2 is implemented and VSTCR_EL2.SL2 is `0b0`, start at level 3. • If FEAT_LPA2 is implemented, the combination of VSTCR_EL2.SL0 == 11 and VSTCR_EL2.SL2 == 1 is reserved. If VSTCR_EL2.TG0 is `0b10` (16KB granule) and FEAT_LPA2 is implemented, start at level 0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

If this field is programmed to a value that is not consistent with the programming of VSTCR_EL2.T0SZ, then a stage 2 level 0 Translation fault is generated.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_TTST is not implemented and (FEAT_D128 is not implemented or VTCR_EL2.D128 == '0'):

Starting level of the Secure stage 2 translation lookup, controlled by VSTCR_EL2. The meaning of this field depends on the value of VSTCR_EL2.TG0.

SL0	Meaning
`0b00`	If VSTCR_EL2.TG0 is `0b00` (4KB granule), start at level 2. If VSTCR_EL2.TG0 is `0b10` (16KB granule) or `0b01` (64KB granule), start at level 3.
`0b01`	If VSTCR_EL2.TG0 is `0b00` (4KB granule), start at level 1. If VSTCR_EL2.TG0 is `0b10` (16KB granule) or `0b01` (64KB granule), start at level 2.
`0b10`	If VSTCR_EL2.TG0 is `0b00` (4KB granule), start at level 0. If VSTCR_EL2.TG0 is `0b10` (16KB granule) or `0b01` (64KB granule), start at level 1.

All other values are reserved. If this field is programmed to a reserved value, or to a value that is not consistent with the programming of VSTCR_EL2.T0SZ, then a stage 2 level 0 Translation fault is generated.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

T0SZ, bits [5:0]

The size offset of the memory region addressed by VSTTBR_EL2. The region size is $2^{(64-T0SZ)}$ bytes.

The maximum and minimum possible values for this field depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

If this field is programmed to a value that is not consistent with the programming of SL0, then a stage 2 level 0 Translation fault is generated.

Note

For the 4KB translation granule, if FEAT_LPA2 is implemented and this field is less than 16, the translation table walk begins with a level -1 initial lookup.
For the 16KB translation granule, if FEAT_LPA2 is implemented and this field is less than 17, the translation table walk begins with a level 0 initial lookup.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing VSTCR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, VSTCR_EL2
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0110	0b010

```
if !(IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x048];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    else
        X[t, 64] = VSTCR_EL2;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        X[t, 64] = VSTCR_EL2;
```

```
MSR VSTCR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0110	0b010

```
if !(IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x048] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    else
        VSTCR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        VSTCR_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.223 VSTTBR_EL2, Virtualization Secure Translation Table Base Register

The VSTTBR_EL2 characteristics are:

Purpose

The base register for stage 2 translation tables to translate Secure IPAs in the Secure EL1&0 translation regime. Holds the base address of the translation table for the initial lookup for stage 2 of an address translation for a Secure IPA in the Secure EL1&0 translation regime, and other information for this translation stage.

Configuration

This register has no effect if EL2 is not enabled in the current Security state.

This register is present only when FEAT_SEL2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to VSTTBR_EL2 are UNDEFINED.

Attributes

VSTTBR_EL2 is a 64-bit register.

Field descriptions

When FEAT_D128 is implemented and VTCR_EL2.D128 == '1':

Bits [63:56]

Reserved, RES0.

BADDR, bits [55:5]

- Bits A[55:x] of the stage 2 translation table base address bits are in register bits[55:x].
- Bits A[(x-1):0] of the stage 2 translation table base address are zero.

Address bit x is the minimum address bit required to align the translation table to the size of the table. x is calculated based on LOG2(StartTableSize), as described in VMSAv9-128. The smallest permitted value of x is 5.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [4:3]

Reserved, RES0.

SKL, bits [2:1]

Skip Level. Skip Level determines the number of levels to be skipped from the regular start level of the Secure stage 2 translation table walk.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SKL	Meaning
0b00	Skip 0 level from the regular start level.
0b01	Skip 1 level from the regular start level.
0b10	Skip 2 levels from the regular start level.
0b11	Skip 3 levels from the regular start level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

Common not Private, for stage 2 of the Secure EL1&0 translation regime. In an implementation that includes FEAT_TTCNP, indicates whether each entry that is pointed to by VSTTBR_EL2 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of VSTTBR_EL2.CnP is 1.

CnP	Meaning
0b0	The translation table entries pointed to by VSTTBR_EL2 are permitted to differ from the entries for VSTTBR_EL2 for other PEs in the Inner Shareable domain. This is not affected by the value of the current VMID.
0b1	The translation table entries pointed to by VSTTBR_EL2 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of VSTTBR_EL2.CnP is 1 and the VMID is the same as the current VMID.

This bit is permitted to be cached in a TLB.

Note

If the value of VSTTBR_EL2.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those VSTTBR_EL2s do not point to the same translation table entries when using the current VMID, then the results of translations using VSTTBR_EL2 are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

When this register has an architecturally-defined reset value, this field resets to a value that is architecturally UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_D128 is not implemented or VTCR_EL2.D128 == '0':

Bits [63:48]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

BADDR, bits [47:1]

The BADDR field represents a 52-bit address if any of the following apply:

- The value of VTCR_EL2.PS represents an OA size of 52 bits.
- The Effective value of VTCR_EL2.DS is 1.

When VSTTBR_EL2.BADDR represents a 52-bit addresses, all of the following apply:

- Register bits[47:x] hold bits[47:x] of the stage 2 translation table base address, where x is determined by the size of the translation table at the start level.
- The smallest permitted value of x is 6.
- Register bits[5:2] hold bits[51:48] of the stage 2 translation table base address.
- Bits[x:0] of the translation table base address are zero.
- When x>6 register bits[(x-1):6] are RES0.
- Register bit[1] is RES0.

Otherwise, all of the following apply:

- Register bits[47:x] hold bits[47:x] of the stage 2 translation table base address.
- Register bits[(x-1):1] are RES0.
- If 52-bit PA is supported, then bits[51:48] of the stage 2 translation table base address are treated as `0b0000`.

Note

If BADDR represents a 52-bit address, and the translation table has fewer than eight entries, the table must be aligned to 64 bytes. Otherwise the translation table must be aligned to the size of the table.
For the 64KB granule, if 52-bit PA is not supported, and the value of VTCR_EL2.PS is `0b110` or `0b111`, one of the following IMPLEMENTATION DEFINED behaviors occur:

- BADDR uses the extended format to represent a 52-bit base address.
- BADDR does not use the extended format.

When the value of ID_AA64MMFR0_EL1.PARange indicates that the implementation supports a 56 bit PA size, bits [55:52] of the stage 2 translation table base address are zero.

If any VSTTBR_EL2[47:1] bit that is defined as RES0 has the value 1 when a translation table walk is performed using VSTTBR_EL2, then the translation table base address might be misaligned, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Bits[x-1:0] of the translation table base address are treated as if all the bits are zero. The value read back from the corresponding register bits is either the value written to the register or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The AArch64 Virtual Memory System Architecture chapter describes how x is calculated based on the value of VSTCR_EL2.T0SZ, the stage of translation, and the translation granule size.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

Common not Private, for stage 2 of the Secure EL1&0 translation regime. In an implementation that includes FEAT_TTCNP, indicates whether each entry that is pointed to by VSTTBR_EL2 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of VSTTBR_EL2.CnP is 1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CnP	Meaning
0b0	The translation table entries pointed to by VSTTBR_EL2 are permitted to differ from the entries for VSTTBR_EL2 for other PEs in the Inner Shareable domain. This is not affected by the value of the current VMID.
0b1	The translation table entries pointed to by VSTTBR_EL2 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of VSTTBR_EL2.CnP is 1 and the VMID is the same as the current VMID.

This bit is permitted to be cached in a TLB.

Note

If the value of VSTTBR_EL2.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those VSTTBR_EL2s do not point to the same translation table entries when using the current VMID, then the results of translations using VSTTBR_EL2 are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

When this register has an architecturally-defined reset value, this field resets to a value that is architecturally UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing VSTTBR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, VSTTBR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0110	0b000

```
if !(IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x030];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    else
        X[t, 64] = VSTTBR_EL2;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        X[t, 64] = VSTTBR_EL2;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MSR VSTTBR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0110	0b000

```
if !(IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x030] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    else
        VSTTBR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        VSTTBR_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.224 VTCR_EL2, Virtualization Translation Control Register

The VTCR_EL2 characteristics are:

Purpose

The control register for stage 2 of the EL1&0 translation regime.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

AArch64 System register VTCR_EL2 bits [31:0] are architecturally mapped to AArch32 System register VTCR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to VTCR_EL2 are UNDEFINED.

Attributes

VTCR_EL2 is a 64-bit register.

Field descriptions

IsProfile(PROFILE_A)

Unless stated otherwise, any of the bits in VTCR_EL2 are permitted to be cached in a TLB.

Bits [63:46]

Reserved, RES0.

HDBSS, bit [45]

When FEAT_HDBSS is implemented:

Enable use of HDBSS.

HDBSS	Meaning
0b0	Hardware tracking of Dirty state Structure is disabled.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

0b1	Hardware tracking of Dirty state Structure is enabled.

If VTCR_EL2.{HA, HD} is not {1, 1}, the Effective value of this field is 0.

If SCR_EL3.HDBSSEn is 0, then this field behaves as 0 for all purposes other than a direct read of the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HAFT, bit [44]

When FEAT_HAFT is implemented:

Hardware managed Access Flag for Table descriptors.

Enables the Hardware managed Access Flag for Table descriptors.

HAFT	Meaning
0b0	Hardware managed Access Flag for Table descriptors is disabled.
0b1	Hardware managed Access Flag for Table descriptors is enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [43:42]

Reserved, RES0.

TL0, bit [41]

When FEAT_THE is implemented:

Control bit to check for presence of MMU TopLevel0 permission attribute.

TL0	Meaning
0b0	This bit does not have any effect on stage 2 translations.
0b1	Enables MMU TopLevel0 permission attribute check for TTBR0_EL1 and TTBR1_EL1 translations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

GCSH, bit [40]

When FEAT_THE is implemented and FEAT_GCS is implemented:

Assured stage 1 translations for Guarded Control Stacks. Enforces use of the AssuredOnly attribute in stage 2 for the memory accessed by privileged Guarded Control Stack data accesses.

GCSH	Meaning
0b0	For the memory accessed by privileged Guarded Control Stack data accesses, the AssuredOnly attribute in stage 2 is not required to be set.
0b1	For the memory accessed by privileged Guarded Control Stack data accesses, the AssuredOnly attribute in stage 2 is required to be set.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [39]

Reserved, RES0.

D128, bit [38]

When FEAT_D128 is implemented:

Enables VMSAv9-128 translation system for stage 2 translation.

D128	Meaning
0b0	Translation system follows VMSAv8-64 translation process.
0b1	Translation system follows VMSAv9-128 translation process.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

S2POE, bit [37]

When FEAT_S2POE is implemented:

Enable Permission Overlay. Enables permission overlay in stage 2 Permission model.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

S2POE	Meaning
0b0	Overlay disabled.
0b1	Overaly enabled.

This bit is not permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

S2PIE, bit [36]

When FEAT_S2PIE is implemented:

Select Permission Model. Enables usage of permission indirection in stage 2 Permission model.

S2PIE	Meaning
0b0	Direct permission model.
0b1	Indirect permission model.

This field is RES1 when VTCR_EL2.D128 is set.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TL1, bit [35]

When FEAT_THE is implemented:

Control bit to check for presence of MMU TopLevel1 permission attribute.

TL1	Meaning
0b0	This bit does not have any effect on stage 2 translations.
0b1	Enables MMU TopLevel1 permission attribute check for TTBR0_EL1 and TTBR1_EL1 translations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AssuredOnly, bit [34]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_THE is implemented:

AssuredOnly attribute enable for VMSAv8-64. Configures use of bit[58] of the stage 2 translation table Block or Page descriptor.

AssuredOnly	Meaning
`0b0`	Bit[58] of each stage 2 translation Block or Page descriptor does not indicate AssuredOnly attribute.
`0b1`	Bit[58] of each stage 2 translation Block or Page descriptor indicates AssuredOnly attribute.

This field is RES0 when VTCR_EL2.D128 is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SL2, bit [33]

When FEAT_LPA2 is implemented and (FEAT_D128 is not implemented or VTCR_EL2.D128 == '0'):

Starting level of the stage 2 translation lookup controlled by VTCR_EL2.

If VTCR_EL2.DS == 1, then VTCR_EL2.SL2, in combination with VTCR_EL2.SL0, gives encodings for the stage 2 translation table walk initial lookup level.

If VTCR_EL2.DS == 0, then VTCR_EL2.SL2 is RES0.

If the translation granule size is not 4KB, then this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DS, bit [32]

When FEAT_LPA2 is implemented and (FEAT_D128 is not implemented or VTCR_EL2.D128 == '0'):

This field affects:

- Whether a 52-bit output address can be described by the translation tables of the 4KB or 16KB translation granules.
- The minimum value of VTCR_EL2.T0SZ and VSTCR_EL2.T0SZ.
- How and where shareability for Block and Page descriptors are encoded.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

DS	Meaning
0b0	Bits[49:48] of translation descriptors are RES0. Bits[9:8] in Block and Page descriptors encode shareability information in the SH[1:0] field. Bits[9:8] in Table descriptors are ignored by hardware. The minimum value of VTCR_EL2.T0SZ is 16. Any memory access using a smaller value generates a stage 2 level 0 translation table fault. The minimum value of VSTCR_EL2.T0SZ is 16. Any memory access using a smaller value generates a stage 2 level 0 translation table fault. Output address[51:48] is 0000.
0b1	Bits[49:48] of translation descriptors hold output address[49:48]. Bits[9:8] in translation descriptors hold output address[51:50]. The shareability information of Block and Page descriptors for cacheable locations is determined by VTCR_EL2.SH0. The minimum value of VTCR_EL2.T0SZ is 12. Any memory access using a smaller value generates a stage 2 level 0 translation table fault. The minimum value of VSTCR_EL2.T0SZ is 12. Any memory access using a smaller value generates a stage 2 level 0 translation table fault. **Note** As FEAT_LPA must be implemented if VTCR_EL2.DS == 1, the minimum values of VTCR_EL2.T0SZ and VSTCR_EL2.T0SZ are 12, as determined by that extension. For the TLBI range instructions affecting IPA, the format of the argument is changed so that bits[36:0] hold BaseADDR[52:16]. For the 4KB translation granule, bits[15:12] of BaseADDR are treated as 0000. For the 16KB translation granule, bits[15:14] of BaseADDR are treated as 00. **Note** This forces alignment of the ranges used by the TLBI range instructions.

This field is RES0 for a 64KB translation granule.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [31]

Reserved, RES1.

NSA, bit [30]

When FEAT_SEL2 is implemented:

Non-secure stage 2 translation output address space for the Secure EL1&0 translation regime.

NSA	Meaning
0b0	All stage 2 translations for the Non-secure IPA space of the Secure EL1&0 translation regime access the Secure PA space.
0b1	All stage 2 translations for the Non-secure IPA space of the Secure EL1&0 translation regime access the Non-secure PA space.

This bit behaves as 1 for all purposes other than reading back the value of the bit when one of the following is true:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- The value of VTCR_EL2.NSW is 1.
- The value of VSTCR_EL2.SA is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSW, bit [29]

When FEAT_SEL2 is implemented:

Non-secure stage 2 translation table address space for the Secure EL1&0 translation regime.

NSW	Meaning
0b0	All stage 2 translation table walks for the Non-secure IPA space of the Secure EL1&0 translation regime are to the Secure PA space.
0b1	All stage 2 translation table walks for the Non-secure IPA space of the Secure EL1&0 translation regime are to the Non-secure PA space.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU62, bit [28]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 2 translation table Block or Page entry.

HWU62	Meaning
0b0	Bit[62] of each stage 2 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[62] of each stage 2 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU61, bit [27]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 2 translation table Block or Page entry.

HWU61	Meaning
0b0	Bit[61] of each stage 2 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[61] of each stage 2 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU60, bit [26]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 2 translation table Block or Page entry.

HWU60	Meaning
0b0	Bit[60] of each stage 2 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.
0b1	Bit[60] of each stage 2 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU59, bit [25]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 2 translation table Block or Page entry.

HWU59	Meaning
0b0	Bit[59] of each stage 2 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

HWU59	Meaning
0b1	Bit[59] of each stage 2 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [24:23]

Reserved, RES0.

HD, bit [22]

When FEAT_HAFDBS is implemented:

Hardware management of dirty state in stage 2 translations when EL2 is enabled in the current Security state.

HD	Meaning
0b0	Stage 2 hardware management of dirty state disabled.
0b1	Stage 2 hardware management of dirty state enabled.

When the Effective value of VTCR_EL2.HA is 0, this field behaves as 0 for all purposes other than a direct read of the value of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HA, bit [21]

When FEAT_HAFDBS is implemented:

Hardware Access flag update in stage 2 translations when EL2 is enabled in the current Security state.

HA	Meaning
0b0	Stage 2 Access flag update disabled.
0b1	Stage 2 Access flag update enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bit [20]

Reserved, RES0.

VS, bit [19]

When FEAT_VMID16 is implemented:

VMID Size.

VS	Meaning
0b0	8-bit VMID. The upper 8 bits of VTTBR_EL2 are ignored by the hardware, and treated as if they are all zeros, for every purpose except when reading back the register.
0b1	16-bit VMID. The upper 8 bits of VTTBR_EL2 are used for allocation and matching in the TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PS, bits [18:16]

Physical address Size for the second stage of translation.

PS	Meaning
0b000	32 bits, 4GB.
0b001	36 bits, 64GB.
0b010	40 bits, 1TB.
0b011	42 bits, 4TB.
0b100	44 bits, 16TB.
0b101	48 bits, 256TB.
0b110	52 bits, 4PB.
0b111	56 bits, 64PB.

The values 0b110 and 0b111 represent different output address sizes depending on implementation choices and translation configuration.

The following table captures the output address size represented by the value 0b110:

Descriptor Format	ID_AA64MMFR0_EL1.PARange	Translation Granule	DS[1] | Represented OA size |	
Any	0b0101	Any	Any	48 bits, 256TB
VMSAv8-64	0b011x	4KB or 16KB	0	48 bits, 256TB
VMSAv8-64	0b011x	4KB or 16KB	1	52 bits, 4PB

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Descriptor Format	ID_AA64MMFR0_EL1.PARange	Translation Granule	DS[1] | Represented OA size |	
VMSAv8-64	`0b011x`	64KB	N/A	52 bits, 4PB
VMSAv9-128	`0b011x`	Any	N/A	52 bits, 4PB

[1] This column represents the value of VTCR_EL2.DS.

The following table captures the output address size represented by the value `0b111`:

Descriptor Format	ID_AA64MMFR0_EL1.PARange	Translation Granule	Represented OA size
Any	`0b0110`	Any	OA size represented by `0b110`
VMSAv8-64	`0b0111`	Any	OA size represented by `0b110`
VMSAv9-128	`0b0111`	Any	56 bits, 64PB

If 52-bit PA is supported, and the translation table descriptors cannot express an OA larger than 48-bits, then bits[51:48] of every translation table base address are treated as `0b0000` for the stage of translation controlled by VTCR_EL2.

If 56-bit PA is supported, and the translation table descriptors cannot express an OA larger than 52-bits, then bits[55:52] of every translation table base address are treated as `0b0000` for the stage of translation controlled by VTCR_EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TG0, bits [15:14]

Granule size for the VTTBR_EL2.

TG0	Meaning
`0b00`	4KB.
`0b01`	64KB.
`0b10`	16KB.

Other values are reserved.

If FEAT_GTG is implemented, ID_AA64MMFR0_EL1.{TGran4_2, TGran16_2, TGran64_2} indicate which granule sizes are supported for stage 2 translation.

If FEAT_GTG is not implemented, ID_AA64MMFR0_EL1.{TGran4, TGran16, TGran64} indicate which granule sizes are supported.

If the value is programmed to either a reserved value or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SH0, bits [13:12]

Shareability attribute for memory associated with translation table walks using VTTBR_EL2 or VSTTBR_EL2.

SH0	Meaning
0b00	Non-shareable.
0b10	Outer Shareable.
0b11	Inner Shareable.

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ORGN0, bits [11:10]

Outer cacheability attribute for memory associated with translation table walks using VTTBR_EL2 or VSTTBR_EL2.

ORGN0	Meaning
0b00	Normal memory, Outer Non-cacheable.
0b01	Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRGN0, bits [9:8]

Inner cacheability attribute for memory associated with translation table walks using VTTBR_EL2 or VSTTBR_EL2.

IRGN0	Meaning
0b00	Normal memory, Inner Non-cacheable.
0b01	Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.
0b10	Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11	Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SL0, bits [7:6]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_TTST is implemented and (FEAT_D128 is not implemented or VTCR_EL2.D128 == '0'):

Starting level of the stage 2 translation lookup, controlled by VTCR_EL2. The meaning of this field depends on the value of VTCR_EL2.TG0.

SL0	Meaning
`0b00`	If VTCR_EL2.TG0 is `0b00` (4KB granule): • If FEAT_LPA2 is not implemented, start at level 2. • If FEAT_LPA2 is implemented and VTCR_EL2.SL2 is `0b0`, start at level 2. • If FEAT_LPA2 is implemented and VTCR_EL2.SL2 is `0b1`, start at level -1. If VTCR_EL2.TG0 is `0b10` (16KB granule) or `0b01` (64KB granule), start at level 3.
`0b01`	If VTCR_EL2.TG0 is `0b00` (4KB granule): • If FEAT_LPA2 is not implemented, start at level 1. • If FEAT_LPA2 is implemented and VTCR_EL2.SL2 is `0b0`, start at level 1. • If FEAT_LPA2 is implemented, the combination of VTCR_EL2.SL0 == 01 and VTCR_EL2.SL2 == 1 is reserved. If VTCR_EL2.TG0 is `0b10` (16KB granule) or `0b01` (64KB granule), start at level 2.
`0b10`	If VTCR_EL2.TG0 is `0b00` (4KB granule): • If FEAT_LPA2 is not implemented, start at level 0. • If FEAT_LPA2 is implemented and VTCR_EL2.SL2 is `0b0`, start at level 0. • If FEAT_LPA2 is implemented, the combination of VTCR_EL2.SL0 == 10 and VTCR_EL2.SL2 == 1 is reserved. If VTCR_EL2.TG0 is `0b10` (16KB granule) or `0b01` (64KB granule), start at level 1.
`0b11`	If VTCR_EL2.TG0 is `0b00` (4KB granule): • If FEAT_LPA2 is not implemented, start at level 3. • If FEAT_LPA2 is implemented and VTCR_EL2.SL2 is `0b0`, start at level 3. • If FEAT_LPA2 is implemented, the combination of VTCR_EL2.SL0 == 11 and VTCR_EL2.SL2 == 1 is reserved. If VTCR_EL2.TG0 is `0b10` (16KB granule) and FEAT_LPA2 is implemented, start at level 0.

If this field is programmed to a value that is not consistent with the programming of VTCR_EL2.T0SZ, then a stage 2 level 0 Translation fault is generated.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_TTST is not implemented and (FEAT_D128 is not implemented or VTCR_EL2.D128 == '0'):

Starting level of the stage 2 translation lookup, controlled by VTCR_EL2. The meaning of this field depends on the value of VTCR_EL2.TG0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SL0	Meaning
0b00	If VTCR_EL2.TG0 is 0b00 (4KB granule), start at level 2. If VTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 3.
0b01	If VTCR_EL2.TG0 is 0b00 (4KB granule), start at level 1. If VTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 2.
0b10	If VTCR_EL2.TG0 is 0b00 (4KB granule), start at level 0. If VTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 1.

All other values are reserved. If this field is programmed to a reserved value, or to a value that is not consistent with the programming of VTCR_EL2.T0SZ, then a stage 2 level 0 Translation fault is generated.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

T0SZ, bits [5:0]

The size offset of the memory region addressed by VTTBR_EL2. The region size is $2^{(64-T0SZ)}$ bytes.

The maximum and minimum possible values for T0SZ depend on the level of translation table and the memory translation granule size, as described in 'The AArch64 Virtual Memory System Architecture'.

If this field is programmed to a value that is not consistent with the programming of SL0, then a stage 2 level 0 Translation fault is generated.

Note

For the 4KB translation granule, if FEAT_LPA2 is implemented, VTCR_EL2.DS is 1, and this field is less than 16, the translation table walk begins with a level -1 initial lookup.
For the 16KB translation granule, if FEAT_LPA2 is implemented, VTCR_EL2.DS is 1, and this field is less than 17, the translation table walk begins with a level 0 initial lookup.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing VTCR_EL2

Unless stated otherwise, any of the bits in VTCR_EL2 are permitted to be cached in a TLB.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, VTCR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0001	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x040];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = VTCR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = VTCR_EL2;
```

MSR VTCR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0001	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x040] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VTCR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    VTCR_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.225 VTTBR_EL2, Virtualization Translation Table Base Register

The VTTBR_EL2 characteristics are:

Purpose

Holds the base address of the translation table for the initial lookup for stage 2 of an address translation in the EL1&0 translation regime, and other information for this translation regime.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

VTTBR_EL2 is a 128-bit register that can also be accessed as a 64-bit value. If it is accessed as a 64-bit register, accesses read and write bits [63:0] and do not modify bits [127:64].

AArch64 System register VTTBR_EL2 bits [63:0] are architecturally mapped to AArch32 System register VTTBR[63:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to VTTBR_EL2 are UNDEFINED.

Attributes

VTTBR_EL2 is a:

- 128-bit register when FEAT_D128 is implemented and VTCR_EL2.D128 == '1'.
- 64-bit register when FEAT_D128 is not implemented or VTCR_EL2.D128 == '0'.

Field descriptions

When FEAT_D128 is implemented and VTCR_EL2.D128 == '1':

Bits [127:88]

Reserved, RES0.

BADDR, bits [87:80, 47:5]

Translation table base address:

- Bits A[55:x] of the stage 2 translation table base address bits are in register bits[87:80, 47:x].
- Bits A[(x-1):0] of the stage 2 translation table base address are zero.

Address bit x is the minimum address bit required to align the translation table to the size of the table. x is calculated based on LOG2(StartTableSize), as described in VMSAv9-128. The smallest permitted value of x is 5.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [79:64]

Reserved, RES0.

VMID, bits [63:48]

When FEAT_VMID16 is implemented and VTCR_EL2.VS == '1':

VMID, bits [15:0]

The VMID for the translation table.

If the implementation has an 8-bit VMID, bits [15:8] of this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_VMID16 is not implemented or VTCR_EL2.VS == '0':

Bits [15:8]

Reserved, RES0.

VMID, bits [7:0]

The VMID for the translation table.

The VMID is 8 bits when any of the following are true:

- EL2 is using AArch32.
- VTCR_EL2.VS is 0.
- FEAT_VMID16 is not implemented.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [4:3]

Reserved, RES0.

SKL, bits [2:1]

Skip Level. Skip Level determines the number of levels to be skipped from the regular start level of the Non-Secure stage 2 translation table walk.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SKL	Meaning
0b00	Skip 0 level from the regular start level.
0b01	Skip 1 level from the regular start level.
0b10	Skip 2 levels from the regular start level.
0b11	Skip 3 levels from the regular start level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When FEAT_TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by VTTBR_EL2 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of VTTBR_EL2.CnP is 1.

CnP	Meaning
0b0	The translation table entries pointed to by VTTBR_EL2 are permitted to differ from the entries for VTTBR_EL2 for other PEs in the Inner Shareable domain. This is not affected by the value of the current VMID.
0b1	The translation table entries pointed to by VTTBR_EL2 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of VTTBR_EL2.CnP is 1 and the VMID is the same as the current VMID.

This bit is permitted to be cached in a TLB.

Note

If the value of VTTBR_EL2.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those VTTBR_EL2s do not point to the same translation table entries when using the current VMID, then the results of translations using VTTBR_EL2 are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

When FEAT_D128 is not implemented or VTCR_EL2.D128 == '0':

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

VMID, bits [63:48]

When FEAT_VMID16 is implemented and VTCR_EL2.VS == '1':

15	0
VMID	

VMID, bits [15:0]

The VMID for the translation table.

If the implementation has an 8-bit VMID, bits [15:8] of this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_VMID16 is not implemented or VTCR_EL2.VS == '0':

15	8	7	0
RES0		VMID	

Bits [15:8]

Reserved, RES0.

VMID, bits [7:0]

The VMID for the translation table.

The VMID is 8 bits when any of the following are true:

- EL2 is using AArch32.
- VTCR_EL2.VS is 0.
- FEAT_VMID16 is not implemented.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

BADDR, bits [47:1]

Translation table base address, A[47:x] or A[51:x], bits[47:1].

The BADDR field represents a 52-bit address if any of the following apply:

- The value of VTCR_EL2.PS represents an OA size of 52 bits.
- The Effective value of VTCR_EL2.DS is 1.

When VTTBR_EL2.BADDR represents a 52-bit addresses, all of the following apply:

- Register bits[47:x] hold bits[47:x] of the stage 2 translation table base address, where x is determined by the size of the translation table at the start level.
- The smallest permitted value of x is 6.
- Register bits[5:2] hold bits[51:48] of the stage 2 translation table base address.
- Bits[x:0] of the translation table base address are zero.
- When x>6 register bits[(x-1):6] are RES0.
- Register bit[1] is RES0.

Otherwise, all of the following apply:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- Register bits[47:x] hold bits[47:x] of the stage 2 translation table base address.
- Register bits[(x-1):1] are RES0.
- If 52-bit PA is supported, then bits[51:48] of the stage 2 translation table base address are treated as `0b0000`.

Note

If BADDR represents a 52-bit address, and the translation table has fewer than eight entries, the table must be aligned to 64 bytes. Otherwise the translation table must be aligned to the size of the table.
For the 64KB granule, if 52-bit PA is not supported, and the value of VTCR_EL2.PS is `0b110` or `0b111`, one of the following IMPLEMENTATION DEFINED behaviors occur:

- BADDR uses the extended format to represent a 52-bit base address.
- BADDR does not use the extended format.

When the value of ID_AA64MMFR0_EL1.PARange indicates that the implementation supports a 56 bit PA size, bits [55:52] of the stage 2 translation table base address are zero.

If any VTTBR_EL2[47:0] bit that is defined as RES0 has the value 1 when a translation table walk is performed using VTTBR_EL2, then the translation table base address might be misaligned, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Bits[x-1:0] of the translation table base address are treated as if all the bits are zero. The value read back from the corresponding register bits is either the value written to the register or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The AArch64 Virtual Memory System Architecture chapter describes how x is calculated based on the value of VTCR_EL2.T0SZ, the stage of translation, and the translation granule size.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When FEAT_TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by VTTBR_EL2 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of VTTBR_EL2.CnP is 1.

CnP	Meaning
`0b0`	The translation table entries pointed to by VTTBR_EL2 are permitted to differ from the entries for VTTBR_EL2 for other PEs in the Inner Shareable domain. This is not affected by the value of the current VMID.
`0b1`	The translation table entries pointed to by VTTBR_EL2 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of VTTBR_EL2.CnP is 1 and the VMID is the same as the current VMID.

This bit is permitted to be cached in a TLB.

Note

If the value of VTTBR_EL2.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those VTTBR_EL2s do not point to the same translation table entries when using the current VMID, then the results of translations using VTTBR_EL2 are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing VTTBR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, VTTBR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x020];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = VTTBR_EL2<63:0>;
elsif PSTATE.EL == EL3 then
    X[t, 64] = VTTBR_EL2<63:0>;
```

```
MSR VTTBR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x020] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VTTBR_EL2<63:0> = X[t, 64];
elsif PSTATE.EL == EL3 then
    VTTBR_EL2<63:0> = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_D128 is implemented MRRS <Xt>, <Xt+1>, VTTBR_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, t2, 128] = NVMem128[0x020];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        X[t, t2, 128] = VTTBR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, t2, 128] = VTTBR_EL2;
```

When FEAT_D128 is implemented MSRR VTTBR_EL2, <Xt>, <Xt+1>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem128[0x020] = X[t, t2, 128];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x14);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.D128En == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.D128En == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x14);
    else
        VTTBR_EL2<127:0> = X[t, t2, 128];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL3 then
    VTTBR_EL2<127:0> = X[t, t2, 128];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.226 ZCR_EL1, SVE Control Register (EL1)

The ZCR_EL1 characteristics are:

Purpose

This register controls aspects of SVE visible at Exception levels EL1 and EL0.

Configuration

This register has no effect when FEAT_SME is implemented and the PE is in Streaming SVE mode.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this register has no effect on execution at EL0.

This register is present only when FEAT_SME is implemented. Otherwise, direct accesses to ZCR_EL1 are UNDEFINED.

Attributes

ZCR_EL1 is a 64-bit register.

Field descriptions

Bits [63:9]

Reserved, RES0.

Bits [8:4]

Reserved, RAZ/WI.

LEN, bits [3:0]

Requests an Effective Non-streaming SVE vector length at EL1 of (LEN+1)*128 bits. This field also defines the Effective Non-streaming SVE vector length at EL0 when EL2 is not implemented, or EL2 is not enabled in the current Security state, or the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.

The Non-streaming SVE vector length can be any power of two from 128 bits to 2048 bits inclusive. An implementation can support a subset of the architecturally permitted lengths. An implementation is required to support all lengths that are powers of two, from 128 bits up to its maximum implemented Non-streaming SVE vector length.

When FEAT_SME is not implemented, or the PE is not in Streaming SVE mode, the Effective SVE vector length (VL) is equal to the Effective Non-streaming SVE vector length.

When FEAT_SME is implemented and the PE is in Streaming SVE mode, VL is equal to the Effective Streaming SVE vector length. See SMCR_EL1.

For all purposes other than returning the result of a direct read of ZCR_EL1, the PE selects the Effective Non-streaming SVE vector length by performing checks in the following order:

1. If EL2 is implemented and enabled in the current Security state, and the requested length is greater than the Effective length at EL2, then the Effective length at EL2 is used.
2. If EL3 is implemented and the requested length is greater than the Effective length at EL3, then the Effective length at EL3 is used.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

3. Otherwise, the Effective length is the highest supported Non-streaming SVE vector length that is less than or equal to the requested length.

An indirect read of ZCR_EL1.LEN appears to occur in program order relative to a direct write of the same register, without the need for explicit synchronization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ZCR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name `ZCR_EL1` or `ZCR_EL12` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ZCR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_SVE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.EZ == '0' then
        UNDEFINED;
    elsif CPACR_EL1.ZEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL1, 0x19);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif ELIsInHost(EL2) && CPTR_EL2.ZEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x19);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x1E0];
    else
        X[t, 64] = ZCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.EZ == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL2) && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif ELIsInHost(EL2) && CPTR_EL2.ZEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x19);
    elsif ELIsInHost(EL2) then
        X[t, 64] = ZCR_EL2;
    else
        X[t, 64] = ZCR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
    else
        X[t, 64] = ZCR_EL1;
```

MSR ZCR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_SVE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.EZ == '0' then
        UNDEFINED;
    elsif CPACR_EL1.ZEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL1, 0x19);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif ELIsInHost(EL2) && CPTR_EL2.ZEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x19);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x1E0] = X[t, 64];
    else
        ZCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.EZ == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL2) && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif ELIsInHost(EL2) && CPTR_EL2.ZEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x19);
    elsif ELIsInHost(EL2) then
        ZCR_EL2 = X[t, 64];
    else
        ZCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
    else
        ZCR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, ZCR_EL12

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_SVE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x1E0];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.EZ == '0' then
            UNDEFINED;
        elsif CPTR_EL2.ZEN IN {'x0'} then
            AArch64.SystemAccessTrap(EL2, 0x19);
        elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x19);
        else
            X[t, 64] = ZCR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        if CPTR_EL3.EZ == '0' then
            AArch64.SystemAccessTrap(EL3, 0x19);
        else
            X[t, 64] = ZCR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented MSR ZCR_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_SVE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x1E0] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.EZ == '0' then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        elsif CPTR_EL2.ZEN IN {'x0'} then
            AArch64.SystemAccessTrap(EL2, 0x19);
        elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x19);
        else
            ZCR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        if CPTR_EL3.EZ == '0' then
            AArch64.SystemAccessTrap(EL3, 0x19);
        else
            ZCR_EL1 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.227 ZCR_EL2, SVE Control Register (EL2)

The ZCR_EL2 characteristics are:

Purpose

This register controls aspects of SVE visible at Exception levels EL2, EL1, and EL0.

Configuration

This register has no effect when EL2 is not enabled in the current Security state, or when FEAT_SME is implemented and the PE is in Streaming SVE mode.

This register is present only when FEAT_SVE is implemented. Otherwise, direct accesses to ZCR_EL2 are UNDEFINED.

Attributes

ZCR_EL2 is a 64-bit register.

Field descriptions

Bits [63:9]

Reserved, RES0.

Bits [8:4]

Reserved, RAZ/WI.

LEN, bits [3:0]

Requests an Effective Non-streaming SVE vector length at EL2 of (LEN+1)*128 bits. This field also defines the Effective Non-streaming SVE vector length at EL0 when the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}.

The Non-streaming SVE vector length can be any power of two from 128 bits to 2048 bits inclusive. An implementation can support a subset of the architecturally permitted lengths. An implementation is required to support all lengths that are powers of two, from 128 bits up to its maximum implemented Non-streaming SVE vector length.

When FEAT_SME is not implemented, or the PE is not in Streaming SVE mode, the Effective SVE vector length (VL) is equal to the Effective Non-streaming SVE vector length.

When FEAT_SME is implemented and the PE is in Streaming SVE mode, VL is equal to the Effective Streaming SVE vector length. See SMCR_EL2.

For all purposes other than returning the result of a direct read of ZCR_EL2, the PE selects the Effective Non-streaming SVE vector length by performing checks in the following order:

1. If EL3 is implemented and the requested length is greater than the Effective length at EL3, then the Effective length at EL3 is used.
2. Otherwise, the Effective length is the highest supported Non-streaming SVE vector length that is less than or equal to the requested length.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

An indirect read of ZCR_EL2.LEN appears to occur in program order relative to a direct write of the same register, without the need for explicit synchronization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ZCR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name `ZCR_EL2` or `ZCR_EL1` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ZCR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_SVE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.EZ == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL2) && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif ELIsInHost(EL2) && CPTR_EL2.ZEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x19);
    else
        X[t, 64] = ZCR_EL2;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
    else
        X[t, 64] = ZCR_EL2;
```

```
MSR ZCR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0010	0b000

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_SVE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.EZ == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL2) && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif ELIsInHost(EL2) && CPTR_EL2.ZEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x19);
    else
        ZCR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
    else
        ZCR_EL2 = X[t, 64];
```

```
MRS <Xt>, ZCR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_SVE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.EZ == '0' then
        UNDEFINED;
    elsif CPACR_EL1.ZEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL1, 0x19);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif ELIsInHost(EL2) && CPTR_EL2.ZEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x19);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x1E0];
    else
        X[t, 64] = ZCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.EZ == '0' then
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif !ELIsInHost(EL2) && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif ELIsInHost(EL2) && CPTR_EL2.ZEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x19);
    elsif ELIsInHost(EL2) then
        X[t, 64] = ZCR_EL2;
    else
        X[t, 64] = ZCR_EL1;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
    else
        X[t, 64] = ZCR_EL1;
```

MSR ZCR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_SVE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.EZ == '0' then
        UNDEFINED;
    elsif CPACR_EL1.ZEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL1, 0x19);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif ELIsInHost(EL2) && CPTR_EL2.ZEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x19);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x1E0] = X[t, 64];
    else
        ZCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.EZ == '0' then
        UNDEFINED;
    elsif !ELIsInHost(EL2) && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif ELIsInHost(EL2) && CPTR_EL2.ZEN IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x19);
    elsif ELIsInHost(EL2) then
        ZCR_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    else
        ZCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
    else
        ZCR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.2.228 ZCR_EL3, SVE Control Register (EL3)

The ZCR_EL3 characteristics are:

Purpose

This register controls aspects of SVE visible at all Exception levels.

Configuration

This register has no effect when FEAT_SME is implemented and the PE is in Streaming SVE mode.

This register is present only when FEAT_SVE is implemented. Otherwise, direct accesses to ZCR_EL3 are UNDEFINED.

Attributes

ZCR_EL3 is a 64-bit register.

Field descriptions

Bits [63:9]

Reserved, RES0.

Bits [8:4]

Reserved, RAZ/WI.

LEN, bits [3:0]

Requests an Effective Non-streaming SVE vector length at EL3 of (LEN+1)*128 bits.

The Non-streaming SVE vector length can be any power of two from 128 bits to 2048 bits inclusive. An implementation can support a subset of the architecturally permitted lengths. An implementation is required to support all lengths that are powers of two, from 128 bits up to its maximum implemented Non-streaming SVE vector length.

When FEAT_SME is not implemented, or the PE is not in Streaming SVE mode, the Effective SVE vector length (VL) is equal to the Effective Non-streaming SVE vector length.

When FEAT_SME is implemented and the PE is in Streaming SVE mode, VL is equal to the Effective Streaming SVE vector length. See SMCR_EL3.

For all purposes other than returning the result of a direct read of ZCR_EL3, the PE selects the highest supported Non-streaming SVE vector length that is less than or equal to the requested length.

An indirect read of ZCR_EL3.LEN appears to occur in program order relative to a direct write of the same register, without the need for explicit synchronization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ZCR_EL3

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, ZCR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_SVE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
    else
        X[t, 64] = ZCR_EL3;
```

```
MSR ZCR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_SVE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
    else
        ZCR_EL3 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.3 Debug registers

This section lists the Debug System registers in AArch64 state, in alphabetic order:

- The principal encoding space for debug registers is `op0` == `0b10`, `op1` == {0, 3, 4}. Instructions for accessing debug System registers summarizes the registers in this encoding space and lists them in order of their encodings.
- In addition, the following registers in the `op0` == `0b11` encoding space are classified as Debug registers:
  - DLR_EL0.
  - DSPSR_EL0.
  - MDCR_EL2.
  - MDCR_EL3.
  - SDER32_EL3.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.3.1 DBGAUTHSTATUS_EL1, Debug Authentication Status Register

The DBGAUTHSTATUS_EL1 characteristics are:

Purpose

Provides information about the state of the IMPLEMENTATION DEFINED authentication interface for debug.

Configuration

AArch64 System register DBGAUTHSTATUS_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGAUTHSTATUS[31:0].

AArch64 System register DBGAUTHSTATUS_EL1 bits [31:0] are architecturally mapped to External register DBGAUTHSTATUS_EL1[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DBGAUTHSTATUS_EL1 are UNDEFINED.

Attributes

DBGAUTHSTATUS_EL1 is a 64-bit register.

Field descriptions

Bits [63:28]

Reserved, RES0.

RTNID, bits [27:26]

Root non-invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.RTID.

RTID, bits [25:24]

Root invasive debug.

RTID	Meaning
`0b00`	Not implemented.
`0b10`	Implemented and disabled. `ExternalRootInvasiveDebugEnabled()` == FALSE.
`0b11`	Implemented and enabled. `ExternalRootInvasiveDebugEnabled()` == TRUE.

All other values are reserved.

If FEAT_RME is not implemented, the only permitted value is `0b00`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bits [23:16]

Reserved, RES0.

RLNID, bits [15:14]

Realm non-invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.RLID.

RLID, bits [13:12]

Realm invasive debug.

RLID	Meaning
`0b00`	Not implemented.
`0b10`	Implemented and disabled. `ExternalRealmInvasiveDebugEnabled()` == FALSE.
`0b11`	Implemented and enabled. `ExternalRealmInvasiveDebugEnabled()` == TRUE.

All other values are reserved.

If FEAT_RME is not implemented, the only permitted value is `0b00`.

Bits [11:8]

Reserved, RES0.

SNID, bits [7:6]

When FEAT_Debugv8p4 is implemented:

Secure non-invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.SID.

Otherwise:

Secure non-invasive debug.

SNID	Meaning
`0b00`	Secure state is not implemented.
`0b10`	Implemented and disabled. ExternalSecureNoninvasiveDebugEnabled() == FALSE.
`0b11`	Implemented and enabled. ExternalSecureNoninvasiveDebugEnabled() == TRUE.

All other values are reserved.

SID, bits [5:4]

Secure invasive debug.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SID	Meaning
0b00	Secure state is not implemented.
0b10	Implemented and disabled. ExternalSecureInvasiveDebugEnabled() == FALSE.
0b11	Implemented and enabled. ExternalSecureInvasiveDebugEnabled() == TRUE.

All other values are reserved.

NSNID, bits [3:2]

When FEAT_Debugv8p4 is implemented:

Non-secure non-invasive debug.

NSNID	Meaning
0b00	Non-secure state is not implemented.
0b11	Implemented and enabled. EL3 is implemented or the Effective value of SCR_EL3.NS is 1.

All other values are reserved.

Otherwise:

Non-secure non-invasive debug.

NSNID	Meaning
0b00	Non-secure state is not implemented.
0b10	Implemented and disabled. ExternalNoninvasiveDebugEnabled() == FALSE.
0b11	Implemented and enabled. ExternalNoninvasiveDebugEnabled() == TRUE.

All other values are reserved.

NSID, bits [1:0]

Non-secure invasive debug.

NSID	Meaning
0b00	Non-secure state is not implemented.
0b10	Implemented and disabled. ExternalInvasiveDebugEnabled() == FALSE.
0b11	Implemented and enabled. ExternalInvasiveDebugEnabled() == TRUE.

All other values are reserved.

Accessing DBGAUTHSTATUS_EL1

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, DBGAUTHSTATUS_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0111	0b1110	0b110

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.DBGAUTHSTATUS_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = DBGAUTHSTATUS_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = DBGAUTHSTATUS_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = DBGAUTHSTATUS_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.3.2 DBGBCR<n>_EL1, Debug Breakpoint Control Registers, n = 0 - 63

The DBGBCR<n>_EL1 characteristics are:

Purpose

Holds control information for a breakpoint. Forms breakpoint n together with value register DBGBVR<n>_EL1.

Configuration

If breakpoint n is not implemented, accesses to this register are UNDEFINED.

AArch64 System register DBGBCR<n>_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGBCR<n>[31:0].

AArch64 System register DBGBCR<n>_EL1 bits [31:0] are architecturally mapped to External register DBGBCR<n>_EL1[31:0].

When FEAT_Debugv8p9 is implemented, AArch64 System register DBGBCR<n>_EL1 bits [63:32] are architecturally mapped to External register DBGBCR<n>_EL1[63:32].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DBGBCR<n>_EL1 are UNDEFINED.

Attributes

DBGBCR<n>_EL1 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

LBNX, bits [31:30]

When FEAT_Debugv8p9 is implemented:

Linked Breakpoint Number.

For Linked address matching breakpoints, with DBGBCR<n>_EL1.LBN, specifies the index of the breakpoint linked to.

For all other breakpoint types, this field is ignored and reads of the register return an UNKNOWN value.

This field extends DBGBCR<n>_EL1.LBN to support up to 64 implemented breakpoints.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SSCE, bit [29]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_RME is implemented:

Security State Control Extended.

The fields that indicate when the breakpoint can be generated are: HMC, PMC, SSC, and SSCE. These fields must be considered in combination, and the values that are permitted for these fields are constrained.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MASK, bits [28:24]

When FEAT_BWE is implemented:

Address Mask. Only address ranges up to 2GB can be watched using a single mask.

MASK	Meaning
`0b00000`	No mask.
`0b00011..0b11111`	Number of address bits masked.

All other values are reserved.

Indicates the number of masked address bits, from `0b00011` masking 3 address bits (`0x00000007` mask for address) to `0b11111` masking 31 address bits (`0x7FFFFFFF` mask for address).

If any of the following apply, then the behavior of the breakpoint is CONSTRAINED UNPREDICTABLE:

- DBGBCR<n>_EL1.MASK is a reserved value.
- DBGBCR<n>_EL1.MASK is zero, DBGBCR<n>_EL1.{BT2, BT} is {0, `0b010x`} (address mismatch breakpoint without linking enabled), and AArch32 is not supported at EL1.
- DBGBCR<n>_EL1.MASK is a valid nonzero value and any of the following apply:
  - DBGBCR<n>_EL1.BAS is not `0b1111`, and AArch32 is supported at EL0.
  - DBGBVR<n>_EL1[(MASK-1):0] is nonzero.
  - DBGBCR<n>_EL1.{BT2, BT} is {0, `0b0x1x`} or {0, `0b1xxx`} (Context matching breakpoint).

When any of these conditions apply, the breakpoint behaves as if one of the following:

- DBGBCR<n>_EL1.MASK has been programmed with a defined value, which might be `0b00000` (no mask), other than for a direct read of DBGBCR<n>_EL1.
- The breakpoint is disabled.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BT, bits [23:20]

Breakpoint Type.

With DBGBCR<n>_EL1.BT2 when implemented, specifies breakpoint type.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

BT	Meaning	Applies when
0b0000	Unlinked instruction address match. DBGBVR<n>_EL1 is the address of an instruction.	
0b0001	Linked instruction address match. As 0b0000, but linked to a breakpoint that has linking enabled.	
0b0010	Unlinked Context ID match. If the Effective value of HCR_EL2.E2H is 1, and either the PE is executing at EL0 with HCR_EL2.TGE set to 1 or the PE is executing at EL2, then DBGBVR<n>_EL1.ContextID is compared against CONTEXTIDR_EL2. Otherwise, DBGBVR<n>_EL1.ContextID is compared against CONTEXTIDR_EL1.	breakpoint n is context-aware
0b0011	As 0b0010, with linking enabled.	breakpoint n is context-aware
0b0100	Unlinked instruction address mismatch. DBGBVR<n>_EL1 is the address of an instruction.	FEAT_BWE is implemented
0b0101	Linked instruction address mismatch. As 0b0100, but linked to a breakpoint that has linking enabled.	FEAT_BWE is implemented
0b0110	Unlinked CONTEXTIDR_EL1 match. DBGBVR<n>_EL1.ContextID is a Context ID compared against CONTEXTIDR_EL1.	EL2 is implemented, FEAT_Debugv8p1 is implemented, and breakpoint n is context-aware
0b0111	As 0b0110, with linking enabled.	EL2 is implemented, FEAT_Debugv8p1 is implemented, and breakpoint n is context-aware
0b1000	Unlinked VMID match. DBGBVR<n>_EL1.VMID is a VMID compared against VTTBR_EL2.VMID.	EL2 is implemented and breakpoint n is context-aware
0b1001	As 0b1000, with linking enabled.	EL2 is implemented and breakpoint n is context-aware
0b1010	Unlinked VMID and Context ID match. DBGBVR<n>_EL1.ContextID is a Context ID compared against CONTEXTIDR_EL1, and DBGBVR<n>_EL1.VMID is a VMID compared against VTTBR_EL2.VMID.	EL2 is implemented and breakpoint n is context-aware
0b1011	As 0b1010, with linking enabled.	EL2 is implemented and breakpoint n is context-aware
0b1100	Unlinked CONTEXTIDR_EL2 match. DBGBVR<n>_EL1.ContextID2 is a Context ID compared against CONTEXTIDR_EL2.	EL2 is implemented, FEAT_Debugv8p1 is implemented, and breakpoint n is context-aware
0b1101	As 0b1100, with linking enabled.	EL2 is implemented, FEAT_Debugv8p1 is implemented, and breakpoint n is context-aware
0b1110	Unlinked Full Context ID match. DBGBVR<n>_EL1.ContextID is compared against CONTEXTIDR_EL1, and DBGBVR<n>_EL1.ContextID2 is compared against CONTEXTIDR_EL2.	EL2 is implemented, FEAT_Debugv8p1 is implemented, and breakpoint n is context-aware
0b1111	As 0b1110, with linking enabled.	EL2 is implemented, FEAT_Debugv8p1 is implemented, and breakpoint n is context-aware

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

LBN, bits [19:16]

Linked Breakpoint Number.

For Linked address matching breakpoints, with DBGBCR<n>_EL1.LBNX when implemented, specifies the index of the breakpoint linked to.

For all other breakpoint types, this field is ignored and reads of the register return an UNKNOWN value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

SSC, bits [15:14]

Security state control. Determines the Security states under which a Breakpoint debug event for breakpoint n is generated.

The fields that indicate when the breakpoint can be generated are: HMC, PMC, SSC, and SSCE. These fields must be considered in combination, and the values that are permitted for these fields are constrained.

For more information on the operation of these fields, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions'.

For more information on the effect of programming the fields to a reserved set of values, see 'Reserved DBGBCR<n>_EL1.{SSC, HMC, PMC} values'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

HMC, bit [13]

Higher mode control. Determines the debug perspective for deciding when a Breakpoint debug event for breakpoint n is generated.

The fields that indicate when the breakpoint can be generated are: HMC, PMC, SSC, and SSCE. These fields must be considered in combination, and the values that are permitted for these fields are constrained.

For more information on the operation of these fields, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions'.

For more information, see DBGBCR<n>_EL1.SSC.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [12:9]

Reserved, RES0.

BAS, bits [8:5]

When FEAT_AA32 is implemented:

Byte address select. Defines which half-words an address-matching breakpoint matches, regardless of the instruction set and Execution state.

The permitted values depend on the breakpoint type.

For Address match breakpoints, the permitted values are:

BAS	Match instruction at	Constraint for debuggers
0b0011	DBGBVR<n>_EL1	Use for T32 instructions.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

BAS	Match instruction at	Constraint for debuggers
0b1100	DBGBVR<n>_EL1 + 2	Use for T32 instructions.
0b1111	DBGBVR<n>_EL1	Use for A64 and A32 instructions.

All other values are reserved. For more information, see 'Reserved DBGBCR<n>_EL1.BAS values'.

For more information on using the BAS field in address match breakpoints, see 'Using the BAS field in Address Match breakpoints'.

For Context matching breakpoints, this field is RES1 and ignored.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

Bit [4]

Reserved, RES0.

BT2, bit [3]

When FEAT_ABLE is implemented and breakpoint n supports address breakpoint linking:

Breakpoint Type 2. With DBGBCR<n>_EL1.BT, specifies breakpoint type.

BT2	Meaning
0b0	As DBGBCR<n>_EL1.BT.
0b1	As DBGBCR<n>_EL1.BT, but with linking enabled. This value is only defined for the following DBGBCR<n>_EL1.BT values: 0b0000, 0b0001, 0b0100, and 0b0101. All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMC, bits [2:1]

Privilege mode control. Determines the Exception level or levels at which a Breakpoint debug event for breakpoint n is generated.

The fields that indicate when the breakpoint can be generated are: HMC, PMC, SSC, and SSCE. These fields must be considered in combination, and the values that are permitted for these fields are constrained.

For more information on the operation of these fields, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions'.

For more information, see DBGBCR<n>_EL1.SSC.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E, bit [0]

Enable breakpoint n.

E	Meaning
0b0	Breakpoint n disabled.
0b1	Breakpoint n enabled.

This field is ignored by the PE and treated as zero when all of the following are true:

- Any of the following are true:
  - `HaltOnBreakpointOrWatchpoint()` is FALSE and the Effective value of MDSCR_EL1.EMBWE is 0.
  - `HaltOnBreakpointOrWatchpoint()` is TRUE and the Effective value of EDSCR2.EHBWE is 0.
- FEAT_Debugv8p9 is implemented.
- n >= 16.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGBCR<n>_EL1

When FEAT_Debugv8p9 is implemented, a PE is permitted to support up to 64 implemented breakpoints.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, DBGBCR<m>_EL1 ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	m[3:0]	0b101

```
integer m = UInt(CRm<3:0>);

if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif (!IsFeatureImplemented(FEAT_Debugv8p9) && m >= NUM_BREAKPOINTS) ||
     ↪(IsFeatureImplemented(FEAT_Debugv8p9) && m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16) >=
     ↪NUM_BREAKPOINTS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.DBGBCRn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            X[t, 64] = DBGBCR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)];
        else
            X[t, 64] = DBGBCR_EL1[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            X[t, 64] = DBGBCR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)];
        else
            X[t, 64] = DBGBCR_EL1[m];
elsif PSTATE.EL == EL3 then
    if OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            X[t, 64] = DBGBCR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)];
        else
            X[t, 64] = DBGBCR_EL1[m];
```

```
MSR DBGBCR<m>_EL1, <Xt> ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	m[3:0]	0b101

```
integer m = UInt(CRm<3:0>);

if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif (!IsFeatureImplemented(FEAT_Debugv8p9) && m >= NUM_BREAKPOINTS) ||
     ↪(IsFeatureImplemented(FEAT_Debugv8p9) && m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16) >=
     ↪NUM_BREAKPOINTS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.DBGBCRn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            DBGBCR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)] = X[t, 64];
        else
            DBGBCR_EL1[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            DBGBCR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)] = X[t, 64];
        else
            DBGBCR_EL1[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            DBGBCR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)] = X[t, 64];
        else
            DBGBCR_EL1[m] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.3.3 DBGBVR<n>_EL1, Debug Breakpoint Value Registers, n = 0 - 63

The DBGBVR<n>_EL1 characteristics are:

Purpose

Holds a virtual address, or a VMID and/or a context ID, for use in breakpoint matching. Forms breakpoint n together with control register DBGBCR<n>_EL1.

Configuration

How this register is interpreted depends on the value of DBGBCR<n>_EL1.BT.

- When DBGBCR<n>_EL1.BT is `0b000x`, this register holds a virtual address.
- When DBGBCR<n>_EL1.BT is `0b001x`, `0b011x`, or `0b110x`, this register holds a Context ID.
- When DBGBCR<n>_EL1.BT is `0b100x`, this register holds a VMID.
- When DBGBCR<n>_EL1.BT is `0b101x`, this register holds a VMID and a Context ID.
- When DBGBCR<n>_EL1.BT is `0b111x`, this register holds two Context ID values.

For other values of DBGBCR<n>_EL1.BT, this register is RES0.

If breakpoint n is not implemented then accesses to this register are UNDEFINED.

AArch64 System register DBGBVR<n>_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGBVR<n>[31:0].

AArch64 System register DBGBVR<n>_EL1 bits [63:32] are architecturally mapped to AArch32 System register DBGBXVR<n>[31:0].

AArch64 System register DBGBVR<n>_EL1 bits [63:0] are architecturally mapped to External register DBGBVR<n>_EL1[63:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DBGBVR<n>_EL1 are UNDEFINED.

Attributes

DBGBVR<n>_EL1 is a 64-bit register.

Field descriptions

When DBGBCR<n>_EL1.BT IN {'000x'}:

RESS[14:8], bits [63:57]

Reserved, Sign extended. Software must set all bits in this field to the same value as the most significant bit of the VA field. If all bits in this field are not the same value as the most significant bit of the VA field, then all of the following apply:

- It is CONSTRAINED UNPREDICTABLE whether the PE ignores this field when comparing an address.
- If the breakpoint is not context-aware, it is IMPLEMENTATION DEFINED whether the value read back in each bit of this field is a copy of the most significant bit of the VA field or the value written.

Bits [56:53]

When FEAT_LVA3 is implemented:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

VA[56:53], bits [56:53]

Extension to VA[48:2]. For more information, see VA[48:2].

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RESS[7:4], bits [56:53]

Extension to RESS[14:8]. For more information, see RESS[14:8].

Bits [52:49]

When FEAT_LVA is implemented:

VA[52:49], bits [52:49]

Extension to VA[48:2]. For more information, see VA[48:2].

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RESS[3:0], bits [52:49]

Extension to RESS[14:8]. For more information, see RESS[14:8].

VA[48:2], bits [48:2]

Bits[48:2] of the address value for comparison.

When FEAT_LVA3 is implemented, (VA[56:53]:VA[52:49]) forms the upper part of the address value. If FEAT_LVA3 is not implemented, bits VA[56:53] are part of the RESS field.

When FEAT_LVA is implemented, VA[52:49] forms the upper part of the address value. If FEAT_LVA is not implemented, bits [52:49] are part of the RESS field.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [1:0]

Reserved, RES0.

When DBGBCR<n>_EL1.BT IN {'001x'}:

Bits [63:32]

Reserved, RES0.

ContextID, bits [31:0]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Context ID value for comparison.

The value is compared against CONTEXTIDR_EL2 when the Effective value of HCR_EL2.E2H is 1, and either:

- The PE is executing at EL2.
- HCR_EL2.TGE is 1, the PE is executing at EL0, and EL2 is enabled in the current Security state.

Otherwise, the value is compared against CONTEXTIDR_EL1.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When DBGBCR<n>_EL1.BT IN {'011x'}, EL2 is implemented, and FEAT_Debugv8p1 is implemented:

Bits [63:32]

Reserved, RES0.

ContextID, bits [31:0]

Context ID value for comparison against CONTEXTIDR_EL1.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When DBGBCR<n>_EL1.BT IN {'100x'} and EL2 is implemented:

Bits [63:48]

Reserved, RES0.

VMID[15:8], bits [47:40]

When FEAT_VMID16 is implemented, VTCR_EL2.VS == '1', and EL2 is using AArch64:

Extension to VMID[7:0]. For more information, see DBGBVR<n>_EL1.VMID[7:0].

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

VMID[7:0], bits [39:32]

VMID value for comparison.

The VMID is 8 bits when any of the following are true:

- EL2 is using AArch32.
- VTCR_EL2.VS is 0.
- FEAT_VMID16 is not implemented.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [31:0]

Reserved, RES0.

When DBGBCR<n>_EL1.BT IN {'101x'} and EL2 is implemented:

Bits [63:48]

Reserved, RES0.

VMID[15:8], bits [47:40]

When FEAT_VMID16 is implemented, VTCR_EL2.VS == '1', and EL2 is using AArch64:

Extension to VMID[7:0]. For more information, see DBGBVR<n>_EL1.VMID[7:0].

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VMID[7:0], bits [39:32]

VMID value for comparison.

The VMID is 8 bits when any of the following are true:

- EL2 is using AArch32.
- VTCR_EL2.VS is 0.
- FEAT_VMID16 is not implemented.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

ContextID, bits [31:0]

Context ID value for comparison against CONTEXTIDR_EL1.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When DBGBCR<n>_EL1.BT IN {'110x'}, EL2 is implemented, and FEAT_Debugv8p1 is implemented:

ContextID2, bits [63:32]

Context ID value for comparison against CONTEXTIDR_EL2.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [31:0]

Reserved, RES0.

When DBGBCR<n>_EL1.BT IN {'111x'}, EL2 is implemented, and FEAT_Debugv8p1 is implemented:

ContextID2, bits [63:32]

Context ID value for comparison against CONTEXTIDR_EL2.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

ContextID, bits [31:0]

Context ID value for comparison against CONTEXTIDR_EL1.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGBVR<n>_EL1

When FEAT_Debugv8p9 is implemented, a PE is permitted to support up to 64 implemented breakpoints.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, DBGBVR<m>_EL1 ; Where m = 0-15
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	m[3:0]	0b100

```
integer m = UInt(CRm<3:0>);

if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif (!IsFeatureImplemented(FEAT_Debugv8p9) && m >= NUM_BREAKPOINTS) ||
     ↪(IsFeatureImplemented(FEAT_Debugv8p9) && m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16) >=
     ↪NUM_BREAKPOINTS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.DBGBVRn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            X[t, 64] = DBGBVR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)];
        else
            X[t, 64] = DBGBVR_EL1[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            X[t, 64] = DBGBVR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)];
        else
            X[t, 64] = DBGBVR_EL1[m];
elsif PSTATE.EL == EL3 then
    if OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            X[t, 64] = DBGBVR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)];
        else
            X[t, 64] = DBGBVR_EL1[m];
```

```
MSR DBGBVR<m>_EL1, <Xt> ; Where m = 0-15
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	m[3:0]	0b100

```
integer m = UInt(CRm<3:0>);

if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif (!IsFeatureImplemented(FEAT_Debugv8p9) && m >= NUM_BREAKPOINTS) ||
     ↪(IsFeatureImplemented(FEAT_Debugv8p9) && m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16) >=
     ↪NUM_BREAKPOINTS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.DBGBVRn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            DBGBVR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)] = X[t, 64];
        else
            DBGBVR_EL1[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            DBGBVR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)] = X[t, 64];
        else
            DBGBVR_EL1[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            DBGBVR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)] = X[t, 64];
        else
            DBGBVR_EL1[m] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.3.4 DBGCLAIMCLR_EL1, Debug CLAIM Tag Clear Register

The DBGCLAIMCLR_EL1 characteristics are:

Purpose

Used by software to read the values of the CLAIM tag bits, and to clear CLAIM tag bits to 0.

The architecture does not define any functionality for the CLAIM tag bits.

Note

CLAIM tags are typically used for communication between the debugger and target software.

Used in conjunction with the DBGCLAIMSET_EL1 register.

Configuration

An implementation must include eight CLAIM tag bits.

AArch64 System register DBGCLAIMCLR_EL1 bits [63:0] are architecturally mapped to AArch64 System register DBGCLAIMSET_EL1[63:0].

AArch64 System register DBGCLAIMCLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGCLAIMCLR[31:0].

AArch64 System register DBGCLAIMCLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGCLAIMSET[31:0].

AArch64 System register DBGCLAIMCLR_EL1 bits [31:0] are architecturally mapped to External register DBGCLAIMCLR_EL1[31:0].

AArch64 System register DBGCLAIMCLR_EL1 bits [31:0] are architecturally mapped to External register DBGCLAIMSET_EL1[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DBGCLAIMCLR_EL1 are UNDEFINED.

Attributes

DBGCLAIMCLR_EL1 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

Bits [31:8]

Reserved, RAZ/WI.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CLAIM<m> , bits [m], for m = 7 to 0

Claim Tag Clear. Indicates the current status of Claim Tag bit <m>, and is used to clear Claim Tag bit <m> to 0.

CLAIM<m>	Meaning
0b0	On a read: Claim Tag bit <m> is not set. On a write: Ignored.
0b1	On a read: Claim Tag bit <m> is set. On a write: Clear Claim tag bit <m> to 0.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Access to this field is W1C.

Accessing DBGCLAIMCLR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, DBGCLAIMCLR_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0111	0b1001	0b110

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.DBGCLAIM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = DBGCLAIMCLR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        X[t, 64] = DBGCLAIMCLR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = DBGCLAIMCLR_EL1;
```

```
MSR DBGCLAIMCLR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0111	0b1001	0b110

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.DBGCLAIM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        DBGCLAIMCLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        DBGCLAIMCLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    DBGCLAIMCLR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.3.5 DBGCLAIMSET_EL1, Debug CLAIM Tag Set Register

The DBGCLAIMSET_EL1 characteristics are:

Purpose

Used by software to set the CLAIM tag bits to 1.

The architecture does not define any functionality for the CLAIM tag bits.

Note

CLAIM tags are typically used for communication between the debugger and target software.

Used in conjunction with the DBGCLAIMCLR_EL1 register.

Configuration

An implementation must include eight CLAIM tag bits.

AArch64 System register DBGCLAIMSET_EL1 bits [63:0] are architecturally mapped to AArch64 System register DBGCLAIMCLR_EL1[63:0].

AArch64 System register DBGCLAIMSET_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGCLAIMSET[31:0].

AArch64 System register DBGCLAIMSET_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGCLAIMCLR[31:0].

AArch64 System register DBGCLAIMSET_EL1 bits [31:0] are architecturally mapped to External register DBGCLAIMSET_EL1[31:0].

AArch64 System register DBGCLAIMSET_EL1 bits [31:0] are architecturally mapped to External register DBGCLAIMCLR_EL1[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DBGCLAIMSET_EL1 are UNDEFINED.

Attributes

DBGCLAIMSET_EL1 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

Bits [31:8]

Reserved, RAZ/WI.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CLAIM<m> , bits [m], for m = 7 to 0

Claim Tag Set. Used to set Claim Tag bit <m> to 1.

CLAIM<m>	Meaning
0b0	On a write: Ignored.
0b1	On a write: Set Claim Tag bit <m> to 1.

Access to this field is RAO/W1S.

Accessing DBGCLAIMSET_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, DBGCLAIMSET_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0111	0b1000	0b110

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.DBGCLAIM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = DBGCLAIMSET_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = DBGCLAIMSET_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = DBGCLAIMSET_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MSR DBGCLAIMSET_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0111	0b1000	0b110

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.DBGCLAIM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        DBGCLAIMSET_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        DBGCLAIMSET_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    DBGCLAIMSET_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.3.6 DBGDTR_EL0, Debug Data Transfer Register, half-duplex

The DBGDTR_EL0 characteristics are:

Purpose

Transfers 64 bits of data between the PE and an external debugger. Can transfer both ways using only a single register.

Configuration

When written, AArch64 System register DBGDTR_EL0 bits [63:32] are architecturally mapped to AArch32 System register DBGDTRRXint[31:0].

When written, AArch64 System register DBGDTR_EL0 bits [63:32] are architecturally mapped to External register DBGDTRRX_EL0[31:0].

When written, AArch64 System register DBGDTR_EL0 bits [63:32] are architecturally mapped to AArch64 System register DBGDTRRX_EL0[31:0].

When written, AArch64 System register DBGDTR_EL0 bits [31:0] are architecturally mapped to AArch32 System register DBGDTRTXint[31:0].

When written, AArch64 System register DBGDTR_EL0 bits [31:0] are architecturally mapped to External register DBGDTRTX_EL0[31:0].

When written, AArch64 System register DBGDTR_EL0 bits [31:0] are architecturally mapped to AArch64 System register DBGDTRTX_EL0[31:0].

When read, AArch64 System register DBGDTR_EL0 bits [63:32] are architecturally mapped to AArch32 System register DBGDTRTXint[31:0].

When read, AArch64 System register DBGDTR_EL0 bits [63:32] are architecturally mapped to External register DBGDTRTX_EL0[31:0].

When read, AArch64 System register DBGDTR_EL0 bits [63:32] are architecturally mapped to AArch64 System register DBGDTRTX_EL0[31:0].

When read, AArch64 System register DBGDTR_EL0 bits [31:0] are architecturally mapped to AArch32 System register DBGDTRRXint[31:0].

When read, AArch64 System register DBGDTR_EL0 bits [31:0] are architecturally mapped to External register DBGDTRRX_EL0[31:0].

When read, AArch64 System register DBGDTR_EL0 bits [31:0] are architecturally mapped to AArch64 System register DBGDTRRX_EL0[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DBGDTR_EL0 are UNDEFINED.

Attributes

DBGDTR_EL0 is a 64-bit register.

Field descriptions

HighWord, bits [63:32]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Writes to this register set DTRRX to the value in this field and do not change RXfull.

Reads of this register:

- If RXfull is 1, return the last value written to DTRTX.
- If RXfull is 0, return an UNKNOWN value.

After the read, RXfull is cleared to 0.

LowWord, bits [31:0]

Writes to this register set DTRTX to the value in this field and set TXfull to 1.

Reads of this register:

- If RXfull is 1, return the last value written to DTRRX.
- If RXfull is 0, return an UNKNOWN value.

After the read, RXfull is cleared to 0.

Accessing DBGDTR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, DBGDTR_EL0
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b0000	0b0100	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif Halted() then
    X[t, 64] = Read_DBGDTR_EL0(64);
elsif PSTATE.EL == EL0 then
    if MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> != '00') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = Read_DBGDTR_EL0(64);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = Read_DBGDTR_EL0(64);
elsif PSTATE.EL == EL2 then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = Read_DBGDTR_EL0(64);
elsif PSTATE.EL == EL3 then
    X[t, 64] = Read_DBGDTR_EL0(64);
```

MSR DBGDTR_EL0, <Xt>

op0	op1	CRn	CRm	op2
0b10	0b011	0b0000	0b0100	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif Halted() then
    Write_DBGDTR_EL0(X[t, 64]);
elsif PSTATE.EL == EL0 then
    if MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> != '00') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        Write_DBGDTR_EL0(X[t, 64]);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        Write_DBGDTR_EL0(X[t, 64]);
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        Write_DBGDTR_EL0(X[t, 64]);
elsif PSTATE.EL == EL3 then
    Write_DBGDTR_EL0(X[t, 64]);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.3.7 DBGDTRRX_EL0, Debug Data Transfer Register, Receive

The DBGDTRRX_EL0 characteristics are:

Purpose

Transfers data from an external debugger to the PE. For example, it is used by a debugger transferring commands and data to a debug target. See DBGDTR_EL0 for additional architectural mappings. It is a component of the Debug Communications Channel.

Configuration

AArch64 System register DBGDTRRX_EL0 bits [31:0] are architecturally mapped to AArch32 System register DBGDTRRXint[31:0].

AArch64 System register DBGDTRRX_EL0 bits [31:0] are architecturally mapped to External register DBGDTRRX_EL0[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DBGDTRRX_EL0 are UNDEFINED.

Attributes

DBGDTRRX_EL0 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

DTRRX, bits [31:0]

DTRRX. Reads of this register:

- If RXfull is 1, return the last value written to DTRRX.
- If RXfull is 0, return an UNKNOWN value.

After the read, RXfull is cleared to 0.

For the full behavior of the Debug Communications Channel, see ‘The Debug Communication Channel and Instruction Transfer Register’.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGDTRRX_EL0

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, DBGDTRRX_EL0
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b0000	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif Halted() then
    X[t, 32] = Read_DBGDTR_EL0(32);
elsif PSTATE.EL == EL0 then
    if MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> != '00') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 32] = Read_DBGDTR_EL0(32);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 32] = Read_DBGDTR_EL0(32);
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 32] = Read_DBGDTR_EL0(32);
elsif PSTATE.EL == EL3 then
    X[t, 32] = Read_DBGDTR_EL0(32);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.3.8 DBGDTRTX_EL0, Debug Data Transfer Register, Transmit

The DBGDTRTX_EL0 characteristics are:

Purpose

Transfers data from the PE to an external debugger. For example, it is used by a debug target to transfer data to the debugger. See DBGDTR_EL0 for additional architectural mappings. It is a component of the Debug Communication Channel.

Configuration

AArch64 System register DBGDTRTX_EL0 bits [31:0] are architecturally mapped to AArch32 System register DBGDTRTXint[31:0].

AArch64 System register DBGDTRTX_EL0 bits [31:0] are architecturally mapped to External register DBGDTRTX_EL0[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DBGDTRTX_EL0 are UNDEFINED.

Attributes

DBGDTRTX_EL0 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

DTRTX, bits [31:0]

DTRTX. Writes to this register:

- If TXfull is 1, DTRTX is set to an UNKNOWN value.
- If TXfull is 0, update the value in DTRTX.

After the write, TXfull is set to 1.

For the full behavior of the Debug Communications Channel, see ‘The Debug Communication Channel and Instruction Transfer Register’.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGDTRTX_EL0

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MSR DBGDTRTX_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b0000	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif Halted() then
    Write_DBGDTR_EL0(X[t, 32]);
elsif PSTATE.EL == EL0 then
    if MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> != '00') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        Write_DBGDTR_EL0(X[t, 32]);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        Write_DBGDTR_EL0(X[t, 32]);
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        Write_DBGDTR_EL0(X[t, 32]);
elsif PSTATE.EL == EL3 then
    Write_DBGDTR_EL0(X[t, 32]);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.3.9 DBGPRCR_EL1, Debug Power Control Register

The DBGPRCR_EL1 characteristics are:

Purpose

Controls behavior of the PE on powerdown request.

Configuration

AArch64 System register DBGPRCR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGPRCR[31:0].

AArch64 System register DBGPRCR_EL1 bit [0] is architecturally mapped to External register EDPRCR[0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DBGPRCR_EL1 are UNDEFINED.

Attributes

DBGPRCR_EL1 is a 64-bit register.

Field descriptions

Bits [63:1]

Reserved, RES0.

CORENPDRQ, bit [0]

When FEAT_DoPD is implemented:

Core no powerdown request. Requests emulation of powerdown.

This request is typically passed to an external power controller. This means that whether a request causes power up is dependent on the IMPLEMENTATION DEFINED nature of the system. The power controller must not allow the Core power domain to switch off while this bit is 1.

CORENPDRQ	Meaning
0b0	If the system responds to a powerdown request, it powers down Core power domain.
0b1	If the system responds to a powerdown request, it does not powerdown the Core power domain, but instead emulates a powerdown of that domain.

In an implementation that includes the recommended external debug interface, this bit drives the **DBGNOPWRDWN** signal.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

It is IMPLEMENTATION DEFINED whether this bit is reset to its Cold reset value on exit from an IMPLEMENTATION DEFINED software-visible retention state. For more information about retention states see 'Core power domain power states'.

Note

Writes to this bit are not prohibited by the IMPLEMENTATION DEFINED authentication interface. This means that a debugger can request emulation of powerdown regardless of whether invasive debug is permitted.

On a Cold reset, if the powerup request is implemented and the powerup request has been asserted, this field is set to an IMPLEMENTATION DEFINED choice of 0 or 1. If the powerup request is not asserted, this field is set to 0.

Otherwise:

Core no powerdown request. Requests emulation of powerdown.

This request is typically passed to an external power controller. This means that whether a request causes power up is dependent on the IMPLEMENTATION DEFINED nature of the system. The power controller must not allow the Core power domain to switch off while this bit is 1.

CORENPDRQ	Meaning
0b0	If the system responds to a powerdown request, it powers down Core power domain.
0b1	If the system responds to a powerdown request, it does not powerdown the Core power domain, but instead emulates a powerdown of that domain.

In an implementation that includes the recommended external debug interface, this bit drives the **DBGNOPWRDWN** signal.

It is IMPLEMENTATION DEFINED whether this bit is reset to the value of EDPRCR.COREPURQ on exit from an IMPLEMENTATION DEFINED software-visible retention state. For more information about retention states see 'Core power domain power states'.

Note

Writes to this bit are not prohibited by the IMPLEMENTATION DEFINED authentication interface. This means that a debugger can request emulation of powerdown regardless of whether invasive debug is permitted.

The reset behavior of this field is:

- On a Cold reset, this field resets to the value in EDPRCR.COREPURQ.

Accessing DBGPRCR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, DBGPRCR_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0001	0b0100	0b100

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.DBGPRCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDOSA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = DBGPRCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = DBGPRCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = DBGPRCR_EL1;
```

```
MSR DBGPRCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0001	0b0100	0b100

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.DBGPRCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDOSA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        DBGPRCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        DBGPRCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    DBGPRCR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.3.10 DBGVCR32_EL2, Debug Vector Catch Register

The DBGVCR32_EL2 characteristics are:

Purpose

Allows access to the AArch32 register DBGVCR from AArch64 state only. Its value has no effect on execution in AArch64 state.

Configuration

If EL2 is not implemented but EL3 is implemented, and EL1 is capable of using AArch32, then this register is not RES0.

AArch64 System register DBGVCR32_EL2 bits [31:0] are architecturally mapped to AArch32 System register DBGVCR[31:0].

This register is present only when FEAT_AA32EL1 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to DBGVCR32_EL2 are UNDEFINED.

Attributes

DBGVCR32_EL2 is a 64-bit register.

Field descriptions

When EL3 is implemented:

Bits [63:32]

Reserved, RES0.

NSF, bit [31]

FIQ vector catch enable in Non-secure state.

The exception vector offset is `0x1C`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSI, bit [30]

IRQ vector catch enable in Non-secure state.

The exception vector offset is `0x18`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [29]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

NSD, bit [28]

Data Abort exception vector catch enable in Non-secure state.

The exception vector offset is `0x10`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSP, bit [27]

Prefetch Abort vector catch enable in Non-secure state.

The exception vector offset is `0x0C`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSS, bit [26]

Supervisor Call (SVC) vector catch enable in Non-secure state.

The exception vector offset is `0x08`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSU, bit [25]

Undefined Instruction vector catch enable in Non-secure state.

The exception vector offset is `0x04`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [24:8]

Reserved, RES0.

SF, bit [7]

FIQ vector catch enable in Secure state.

The exception vector offset is `0x1C`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SI, bit [6]

IRQ vector catch enable in Secure state.

The exception vector offset is `0x18`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [5]

Reserved, RES0.

SD, bit [4]

Data Abort exception vector catch enable in Secure state.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The exception vector offset is `0x10`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SP, bit [3]

Prefetch Abort vector catch enable in Secure state.

The exception vector offset is `0x0C`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SS, bit [2]

Supervisor Call (SVC) vector catch enable in Secure state.

The exception vector offset is `0x08`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SU, bit [1]

Undefined Instruction vector catch enable in Secure state.

The exception vector offset is `0x04`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [0]

Reserved, RES0.

When EL3 is not implemented:

Bits [63:8]

Reserved, RES0.

F, bit [7]

FIQ vector catch enable.

The exception vector offset is `0x1C`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

I, bit [6]

IRQ vector catch enable.

The exception vector offset is `0x18`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [5]

Reserved, RES0.

D, bit [4]

Data Abort exception vector catch enable.

The exception vector offset is `0x10`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P, bit [3]

Prefetch Abort vector catch enable.

The exception vector offset `0x0C`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

S, bit [2]

Supervisor Call (SVC) vector catch enable.

The exception vector offset is `0x08`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

U, bit [1]

Undefined Instruction vector catch enable.

The exception vector offset is `0x04`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [0]

Reserved, RES0.

Accessing DBGVCR32_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, DBGVCR32_EL2
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b10	0b100	0b0000	0b0111	0b000

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = DBGVCR32_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = DBGVCR32_EL2;
```

```
MSR DBGVCR32_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b100	0b0000	0b0111	0b000

```
if !(IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        DBGVCR32_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    DBGVCR32_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.3.11 DBGWCR<n>_EL1, Debug Watchpoint Control Registers, n = 0 - 63

The DBGWCR<n>_EL1 characteristics are:

Purpose

Holds control information for a watchpoint. Forms watchpoint n together with value register DBGWVR<n>_EL1.

Configuration

If watchpoint n is not implemented then accesses to this register are UNDEFINED.

AArch64 System register DBGWCR<n>_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGWCR<n>[31:0].

AArch64 System register DBGWCR<n>_EL1 bits [31:0] are architecturally mapped to External register DBGWCR<n>_EL1[31:0].

When FEAT_Debugv8p9 is implemented, AArch64 System register DBGWCR<n>_EL1 bits [63:32] are architecturally mapped to External register DBGWCR<n>_EL1[63:32].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DBGWCR<n>_EL1 are UNDEFINED.

Attributes

DBGWCR<n>_EL1 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

LBNX, bits [31:30]

When FEAT_Debugv8p9 is implemented:

Linked Breakpoint Number.

For Linked data address watchpoints, with DBGWCR<n>_EL1.LBN, specifies the index of the breakpoint linked to.

For all other watchpoint types, this field is ignored and reads of the register return an UNKNOWN value.

This field extends DBGWCR<n>_EL1.LBN to support up to 64 implemented breakpoints.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SSCE, bit [29]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_RME is implemented:

Security State Control Extended.

The fields that indicate when the watchpoint can be generated are: HMC, PAC, SSC, and SSCE. These fields must be considered in combination, and the values that are permitted for these fields are constrained.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MASK, bits [28:24]

Address Mask. Only address ranges up to 2GB can be watched using a single mask.

MASK	Meaning
`0b00000`	No mask.
`0b00011..0b11111`	Number of address bits masked.

All other values are reserved.

Indicates the number of masked address bits, from `0b00011` masking 3 address bits (`0x00000007` mask for address) to `0b11111` masking 31 address bits (`0x7FFFFFFF` mask for address).

If programmed with a reserved value, the watchpoint behaves as if one of the following:

- DBGWCR<n>_EL1.MASK has been programmed with a defined value, which might be `0b00000` (no mask), other than for a direct read of DBGWCR<n>_EL1.
- The watchpoint is disabled.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bit [23]

Reserved, RES0.

WT2, bit [22]

When FEAT_BWE2 is implemented:

Watchpoint Type 2. With DBGWCR<n>_EL1.WT, specifies watchpoint type.

WT2	Meaning
`0b0`	Watchpoint n is an address match watchpoint.
`0b1`	Watchpoint n is an address mismatch watchpoint.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

Bit [21]

Reserved, RES0.

WT, bit [20]

Watchpoint type. Possible values are:

WT	Meaning
0b0	Unlinked watchpoint.
0b1	Linked watchpoint.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

LBN, bits [19:16]

Linked Breakpoint Number.

For Linked data address watchpoints, with DBGWCR<n>_EL1.LBNX when implemented, specifies the index of the breakpoint linked to.

For all other watchpoint types, this field is ignored and reads of the register return an UNKNOWN value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

SSC, bits [15:14]

Security state control. Determines the Security states under which a Watchpoint debug event for watchpoint n is generated.

The fields that indicate when the watchpoint can be generated are: HMC, PAC, SSC, and SSCE. These fields must be considered in combination, and the values that are permitted for these fields are constrained.

For more information on the operation of these fields, see ‘Execution conditions for which a watchpoint generates Watchpoint exceptions’.

For more information on the effect of programming the fields to a reserved value, see ’Reserved DBGWCR<n>_EL1.{SSC, HMC, PAC} values’.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

HMC, bit [13]

Higher mode control. Determines the debug perspective for deciding when a Watchpoint debug event for watchpoint n is generated.

The fields that indicate when the watchpoint can be generated are: HMC, PAC, SSC, and SSCE. These fields must be considered in combination, and the values that are permitted for these fields are constrained.

For more information on the operation of these fields, see ‘Execution conditions for which a watchpoint generates Watchpoint exceptions’.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

BAS, bits [12:5]

Byte address select. Each bit of this field selects whether a byte from within the word or double-word addressed by DBGWVR<n>_EL1 is being watched.

BAS	Description
xxxxxxx1	Match byte at DBGWVR<n>_EL1
xxxxxx1x	Match byte at DBGWVR<n>_EL1 + 1
xxxxx1xx	Match byte at DBGWVR<n>_EL1 + 2
xxxx1xxx	Match byte at DBGWVR<n>_EL1 + 3

In cases where DBGWVR<n>_EL1 addresses a double-word:

BAS	Description, if DBGWVR<n>_EL1[2] == 0
xxx1xxxx	Match byte at DBGWVR<n>_EL1 + 4
xx1xxxxx	Match byte at DBGWVR<n>_EL1 + 5
x1xxxxxx	Match byte at DBGWVR<n>_EL1 + 6
1xxxxxxx	Match byte at DBGWVR<n>_EL1 + 7

If DBGWVR<n>_EL1[2] == 1, only BAS[3:0] are used and BAS[7:4] are ignored. Arm deprecates setting DBGWVR<n>_EL1[2] == 1.

The valid values for BAS are nonzero binary numbers all of whose set bits are contiguous. All other values are reserved and must not be used by software. See 'Reserved DBGWCR<n>_EL1.BAS values'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

LSC, bits [4:3]

Load/store control. This field enables watchpoint matching on the type of access being made. Possible values of this field are:

LSC	Meaning
`0b01`	Match instructions that load from a watchpointed address.
`0b10`	Match instructions that store to a watchpointed address.
`0b11`	Match instructions that load from or store to a watchpointed address.

All other values are reserved, but must behave as if the watchpoint is disabled. Software must not rely on this property as the behavior of reserved values might change in a future revision of the architecture.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

PAC, bits [2:1]

Privilege of access control. Determines the Exception level or levels at which a Watchpoint debug event for watchpoint n is generated.

The fields that indicate when the watchpoint can be generated are: HMC, PAC, SSC, and SSCE. These fields must be considered in combination, and the values that are permitted for these fields are constrained.

For more information on the operation of these fields, see 'Execution conditions for which a watchpoint generates Watchpoint exceptions'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

E, bit [0]

Enable watchpoint n.

E	Meaning
0b0	Watchpoint n disabled.
0b1	Watchpoint n enabled.

This field is ignored by the PE and treated as zero when all of the following are true:

- Any of the following are true:
  - `HaltOnBreakpointOrWatchpoint()` is FALSE and the Effective value of MDSCR_EL1.EMBWE is 0.
  - `HaltOnBreakpointOrWatchpoint()` is TRUE and the Effective value of EDSCR2.EHBWE is 0.
- FEAT_Debugv8p9 is implemented.
- n >= 16.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGWCR<n>_EL1

When FEAT_Debugv8p9 is implemented, a PE is permitted to support up to 64 implemented watchpoints.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, DBGWCR<m>_EL1 ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	m[3:0]	0b111

```
integer m = UInt(CRm<3:0>);

if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif (!IsFeatureImplemented(FEAT_Debugv8p9) && m >= NUM_WATCHPOINTS) ||
     ↪(IsFeatureImplemented(FEAT_Debugv8p9) && m + (UInt(MDSELR_EL1.BANK) * 16) >= NUM_WATCHPOINTS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.DBGWCRn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            X[t, 64] = DBGWCR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)];
        else
            X[t, 64] = DBGWCR_EL1[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            X[t, 64] = DBGWCR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)];
        else
            X[t, 64] = DBGWCR_EL1[m];
elsif PSTATE.EL == EL3 then
    if OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            X[t, 64] = DBGWCR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)];
        else
            X[t, 64] = DBGWCR_EL1[m];
```

```
MSR DBGWCR<m>_EL1, <Xt> ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	m[3:0]	0b111

```
integer m = UInt(CRm<3:0>);

if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif (!IsFeatureImplemented(FEAT_Debugv8p9) && m >= NUM_WATCHPOINTS) ||
     ↪(IsFeatureImplemented(FEAT_Debugv8p9) && m + (UInt(MDSELR_EL1.BANK) * 16) >= NUM_WATCHPOINTS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.DBGWCRn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            DBGWCR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)] = X[t, 64];
        else
            DBGWCR_EL1[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            DBGWCR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)] = X[t, 64];
        else
            DBGWCR_EL1[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            DBGWCR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)] = X[t, 64];
        else
            DBGWCR_EL1[m] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.3.12 DBGWVR<n>_EL1, Debug Watchpoint Value Registers, n = 0 - 63

The DBGWVR<n>_EL1 characteristics are:

Purpose

Holds a data address value for use in watchpoint matching. Forms watchpoint n together with control register DBGWCR<n>_EL1.

Configuration

If watchpoint n is not implemented then accesses to this register are UNDEFINED.

AArch64 System register DBGWVR<n>_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGWVR<n>[31:0].

AArch64 System register DBGWVR<n>_EL1 bits [63:0] are architecturally mapped to External register DBGWVR<n>_EL1[63:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DBGWVR<n>_EL1 are UNDEFINED.

Attributes

DBGWVR<n>_EL1 is a 64-bit register.

Field descriptions

RESS[14:8], bits [63:57]

Reserved, Sign extended. Software must set all bits in this field to the same value as the most significant bit of the VA field. If all bits in this field are not the same value as the most significant bit of the VA field, then all of the following apply:

- It is CONSTRAINED UNPREDICTABLE whether the PE ignores this field when comparing an address.
- It is IMPLEMENTATION DEFINED whether the value read back in each bit of this field is a copy of the most significant bit of the VA field or the value written.

Bits [56:53]

When FEAT_LVA3 is implemented:

VA[56:53], bits [56:53]

Extension to VA[48:2]. For more information, see VA[48:2].

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RESS[7:4], bits [56:53]

Extension to RESS[14:8]. For more information, see RESS[14:8].

Bits [52:49]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_LVA is implemented:

VA[52:49], bits [52:49]

Extension to VA[48:2]. For more information, see VA[48:2].

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

RESS[3:0], bits [52:49]

Extension to RESS[14:8]. For more information, see RESS[14:8].

VA[48:2], bits [48:2]

Bits[48:2] of the address value for comparison.

When FEAT_LVA3 is implemented, (VA[56:53]:VA[52:49]) forms the upper part of the address value. If FEAT_LVA3 is not implemented, bits VA[56:53] are part of the RESS field.

When FEAT_LVA is implemented, VA[52:49] forms the upper part of the address value. If FEAT_LVA is not implemented, bits [52:49] are part of the RESS field.

Arm deprecates setting DBGWVR<n>_EL1[2] == 1.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [1:0]

Reserved, RES0.

Accessing DBGWVR<n>_EL1

When FEAT_Debugv8p9 is implemented, a PE is permitted to support up to 64 implemented watchpoints.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, DBGWVR<m>_EL1 ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	m[3:0]	0b110

```
integer m = UInt(CRm<3:0>);

if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif (!IsFeatureImplemented(FEAT_Debugv8p9) && m >= NUM_WATCHPOINTS) ||
     ↪(IsFeatureImplemented(FEAT_Debugv8p9) && m + (UInt(MDSELR_EL1.BANK) * 16) >= NUM_WATCHPOINTS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.DBGWVRn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            X[t, 64] = DBGWVR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)];
        else
            X[t, 64] = DBGWVR_EL1[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            X[t, 64] = DBGWVR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)];
        else
            X[t, 64] = DBGWVR_EL1[m];
elsif PSTATE.EL == EL3 then
    if OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            X[t, 64] = DBGWVR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)];
        else
            X[t, 64] = DBGWVR_EL1[m];
```

```
MSR DBGWVR<m>_EL1, <Xt> ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	m[3:0]	0b110

```
integer m = UInt(CRm<3:0>);

if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif (!IsFeatureImplemented(FEAT_Debugv8p9) && m >= NUM_WATCHPOINTS) ||
     ↪(IsFeatureImplemented(FEAT_Debugv8p9) && m + (UInt(MDSELR_EL1.BANK) * 16) >= NUM_WATCHPOINTS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.DBGWVRn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            DBGWVR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)] = X[t, 64];
        else
            DBGWVR_EL1[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            DBGWVR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)] = X[t, 64];
        else
            DBGWVR_EL1[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        if IsFeatureImplemented(FEAT_Debugv8p9) then
            DBGWVR_EL1[m + (UInt(EffectiveMDSELR_EL1_BANK()) * 16)] = X[t, 64];
        else
            DBGWVR_EL1[m] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.3.13 DLR_EL0, Debug Link Register

The DLR_EL0 characteristics are:

Purpose

In Debug state, holds the address to restart from.

Configuration

AArch64 System register DLR_EL0 bits [31:0] are architecturally mapped to AArch32 System register DLR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DLR_EL0 are UNDEFINED.

Attributes

DLR_EL0 is a 64-bit register.

Field descriptions

ADDR, bits [63:0]

Restart address.

Accessing DLR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, DLR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b0100	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !Halted() then
    UNDEFINED;
else
    X[t, 64] = DLR_EL0;
```

```
MSR DLR_EL0, <Xt>
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b011	0b0100	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !Halted() then
    UNDEFINED;
else
    DLR_EL0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.3.14 DSPSR_EL0, Debug Saved Program Status Register

The DSPSR_EL0 characteristics are:

Purpose

Holds the saved process state for Debug state. On entering Debug state, PSTATE information is written to this register. On exiting Debug state, values are copied from this register to PSTATE.

Configuration

AArch64 System register DSPSR_EL0 bits [31:0] are architecturally mapped to AArch32 System register DSPSR[31:0].

When FEAT_Debugv8p9 is implemented, AArch64 System register DSPSR_EL0 bits [63:32] are architecturally mapped to AArch32 System register DSPSR2[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to DSPSR_EL0 are UNDEFINED.

Attributes

DSPSR_EL0 is a 64-bit register.

Field descriptions

When FEAT_AA32 is implemented and exiting Debug state to AArch32 state:

Bits [63:37]

Reserved, RES0.

UINJ, bit [36]

When FEAT_UINJ is implemented:

Inject Undefined Instruction exception. Copied to PSTATE.UINJ on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [35:34]

Reserved, RES0.

PPEND, bit [33]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_SEBEP is implemented:

PMU Profiling exception pending bit. Copied to PSTATE.PPEND on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [32]

Reserved, RES0.

N, bit [31]

Negative Condition flag. Copied to PSTATE.N on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Copied to PSTATE.Z on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Copied to PSTATE.C on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Copied to PSTATE.V on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Copied to PSTATE.Q on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Copied to PSTATE.IT on exiting Debug state.

On exiting Debug state, DSPSR_EL0.IT must contain a value that is valid for the instruction being returned to.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DIT, bit [24]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_DIT is implemented:

Data Independent Timing. Copied to PSTATE.DIT on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Copied to PSTATE.SSBS on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. Copied to PSTATE.PAN on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SS, bit [21]

Software Step. Copied to PSTATE.SS on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IL, bit [20]

Illegal Execution state. Copied to PSTATE.IL on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Copied to PSTATE.GE on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Copied to PSTATE.E on exiting Debug state.

If the implementation does not support big-endian operation, DSPSR_EL0.E is RES0. If the implementation does not support little-endian operation, DSPSR_EL0.E is RES1. On exiting Debug state, if the implementation does not

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

support big-endian operation at the Exception level being returned to, DSPSR_EL0.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, DSPSR_EL0.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError exception mask. Copied to PSTATE.A on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Copied to PSTATE.I on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Copied to PSTATE.F on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Copied to PSTATE.T on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4], bit [4]

Execution state. Copied to PSTATE.nRW on exiting Debug state.

M[4]	Meaning
`0b1`	AArch32 execution state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[3:0], bits [3:0]

AArch32 Mode. Copied to PSTATE.M[3:0] on exiting Debug state.

M[3:0]	Meaning	Applies when
`0b0000`	User.	

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

M[3:0]	Meaning	Applies when
0b0001	FIQ.	
0b0010	IRQ.	
0b0011	Supervisor.	
0b0110	Monitor.	FEAT_EL3 is implemented
0b0111	Abort.	
0b1010	Hyp.	
0b1011	Undefined.	
0b1111	System.	

Other values are reserved. If DSPSR_EL0.M[3:0] has a Reserved value, or a value for an unimplemented Exception level, exiting Debug state is an illegal return event, as described in 'Illegal return events from AArch64 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_AA64 is implemented and entering or exiting Debug state from or to AArch64 state:

Bits [63:37]

Reserved, RES0.

UINJ, bit [36]

When FEAT_UINJ is implemented:

Inject Undefined Instruction exception. Set to the value of PSTATE.UINJ on entering Debug state, and copied to PSTATE.UINJ on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PACM, bit [35]

When FEAT_PAuth_LR is implemented:

PACM. Set to the value of PSTATE.PACM on entering Debug state, and copied to PSTATE.PACM on exiting Debug state.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLOCK, bit [34]

When FEAT_GCS is implemented:

Exception return state lock. Set to the value of PSTATE.EXLOCK on entering Debug state, and copied to PSTATE.EXLOCK on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PPEND, bit [33]

When FEAT_SEBEP is implemented:

PMU Profiling exception pending bit. Set to the value of PSTATE.PPEND on entering Debug state, and conditionally copied to PSTATE.PPEND on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PM, bit [32]

When FEAT_EBEP is implemented:

Profiling exception mask bit. Set to the value of PSTATE.PM on entering Debug state, and copied to PSTATE.PM on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on entering Debug state, and copied to PSTATE.N on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on entering Debug state, and copied to PSTATE.Z on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on entering Debug state, and copied to PSTATE.C on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on entering Debug state, and copied to PSTATE.V on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [27:26]

Reserved, RES0.

TCO, bit [25]

When FEAT_MTE is implemented:

Tag Check Override. Set to the value of PSTATE.TCO on entering Debug state, and copied to PSTATE.TCO on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [24]

When FEAT_DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on entering Debug state, and copied to PSTATE.DIT on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UAO, bit [23]

When FEAT_UAO is implemented:

User Access Override. Set to the value of PSTATE.UAO on entering Debug state, and copied to PSTATE.UAO on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on entering Debug state, and copied to PSTATE.PAN on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SS, bit [21]

Software Step. Set to the value of PSTATE.SS on entering Debug state, and conditionally copied to PSTATE.SS on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on entering Debug state, and copied to PSTATE.IL on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:14]

Reserved, RES0.

ALLINT, bit [13]

When FEAT_NMI is implemented:

All IRQ or FIQ interrupts mask. Set to the value of PSTATE.ALLINT on entering Debug state, and copied to PSTATE.ALLINT on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SSBS, bit [12]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on entering Debug state, and copied to PSTATE.SSBS on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BTYPE, bits [11:10]

When FEAT_BTI is implemented:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Branch Type Indicator. Set to the value of PSTATE.BTYPE on entering Debug state, and copied to PSTATE.BTYPE on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

D, bit [9]

Debug exception mask. Set to the value of PSTATE.D on entering Debug state, and copied to PSTATE.D on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError exception mask. Set to the value of PSTATE.A on entering Debug state, and copied to PSTATE.A on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on entering Debug state, and copied to PSTATE.I on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on entering Debug state, and copied to PSTATE.F on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [5]

Reserved, RES0.

M[4], bit [4]

Execution state. Set to `0b0`, the value of PSTATE.nRW, on entering Debug state from AArch64 state, and copied to PSTATE.nRW on exiting Debug state.

M[4]	Meaning
`0b0`	AArch64 execution state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

M[3:0], bits [3:0]

AArch64 Exception level and selected Stack Pointer.

M[3:0]	Meaning	Applies when
0b0000	EL0.	
0b0100	EL1 with SP_EL0 (EL1t).	
0b0101	EL1 with SP_EL1 (EL1h).	
0b1000	EL2 with SP_EL0 (EL2t).	
0b1001	EL2 with SP_EL2 (EL2h).	
0b1100	EL3 with SP_EL0 (EL3t).	FEAT_EL3 is implemented
0b1101	EL3 with SP_EL3 (EL3h).	FEAT_EL3 is implemented

Other values are reserved. If DSPSR_EL0.M[3:0] has a Reserved value, or a value for an unimplemented Exception level, exiting Debug state is an illegal return event, as described in 'Illegal return events from AArch64 state'.

The bits in this field are interpreted as follows:

- M[3:2] is set to the value of PSTATE.EL on entering Debug state and copied to PSTATE.EL on exiting Debug state.
- M[1] is unused and is 0 for all non-reserved values.
- M[0] is set to the value of PSTATE.SP on entering Debug state and copied to PSTATE.SP on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing DSPSR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, DSPSR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b0100	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !Halted() then
    UNDEFINED;
else
    X[t, 64] = DSPSR_EL0;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MSR DSPSR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b0100	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif !Halted() then
    UNDEFINED;
else
    DSPSR_EL0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.3.15 MDCCINT_EL1, Monitor DCC Interrupt Enable Register

The MDCCINT_EL1 characteristics are:

Purpose

Enables interrupt requests to be signaled based on the DCC status flags.

Configuration

AArch64 System register MDCCINT_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGDCCINT[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to MDCCINT_EL1 are UNDEFINED.

Attributes

MDCCINT_EL1 is a 64-bit register.

Field descriptions

Bits [63:31]

Reserved, RES0.

RX, bit [30]

DCC interrupt request enable control for DTRRX. Enables a common **COMMIRQ** interrupt request to be signaled based on the DCC status flags.

RX	Meaning
0b0	No interrupt request generated by DTRRX.
0b1	Interrupt request will be generated on RXfull == 1.

If legacy **COMMRX** and **COMMTX** signals are implemented, then these are not affected by the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

TX, bit [29]

DCC interrupt request enable control for DTRTX. Enables a common **COMMIRQ** interrupt request to be signaled based on the DCC status flags.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TX	Meaning
0b0	No interrupt request generated by DTRTX.
0b1	Interrupt request will be generated on TXfull == 0.

If legacy **COMMRX** and **COMMTX** signals are implemented, then these are not affected by the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Bits [28:0]

Reserved, RES0.

Accessing MDCCINT_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MDCCINT_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    X[t, 64] = MDCCINT_EL1;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1'
        then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MDCCINT_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1'
        then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MDCCINT_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MDCCINT_EL1;
```

```
MSR MDCCINT_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    MDCCINT_EL1 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1'
        ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MDCCINT_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1'
        ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MDCCINT_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MDCCINT_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.3.16 MDCCSR_EL0, Monitor DCC Status Register

The MDCCSR_EL0 characteristics are:

Purpose

Read-only register containing control status flags for the DCC.

Configuration

AArch64 System register MDCCSR_EL0 bits [30:29] are architecturally mapped to External register EDSCR[30:29].

AArch64 System register MDCCSR_EL0 bits [30:29] are architecturally mapped to AArch32 System register DBGDSCRint[30:29].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to MDCCSR_EL0 are UNDEFINED.

Attributes

MDCCSR_EL0 is a 64-bit register.

Field descriptions

Bits [63:31]

Reserved, RES0.

RXfull, bit [30]

DTRRX full. Read-only view of the equivalent bit in the EDSCR.

TXfull, bit [29]

DTRTX full. Read-only view of the equivalent bit in the EDSCR.

Bits [28:19]

Reserved, RES0.

Bits [18:15]

Reserved, RAZ.

Bits [14:13]

Reserved, RES0.

Bit [12]

Reserved, RAZ.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bits [11:6]

Reserved, RES0.

Bits [5:2]

Reserved, RAZ.

Bits [1:0]

Reserved, RES0.

Accessing MDCCSR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MDCCSR_EL0
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b0000	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    X[t, 64] = MDCCSR_EL0;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1'
        ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> != '00') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MDCCSR_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1'
        ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MDCCSR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1'
        ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MDCCSR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MDCCSR_EL0;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.3.17 MDCR_EL2, Monitor Debug Configuration Register (EL2)

The MDCR_EL2 characteristics are:

Purpose

Provides EL2 configuration options for self-hosted debug and the Performance Monitors Extension.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

AArch64 System register MDCR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HDCR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to MDCR_EL2 are UNDEFINED.

Attributes

MDCR_EL2 is a 64-bit register.

Field descriptions

Bits [63:51]

Reserved, RES0.

EnSTEPOP, bit [50]

When FEAT_STEP2 is implemented:

EnSTEPOP	Meaning
0b0	Execution from MDSTEPOP_EL1 is disabled.
0b1	Execution from MDSTEPOP_EL1 is not disabled by this control.

If EL2 is not implemented or EL2 is disabled in the current Security state, then the Effective value of this field is 0b1, other than for a direct read of the register.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [49:44]

Reserved, RES0.

EBWE, bit [43]

When FEAT_Debugv8p9 is implemented:

Extended Breakpoint and Watchpoint Enable. Enables use of additional breakpoints or watchpoints.

EBWE	Meaning
0b0	The Effective value of MDSCR_EL1.EMBWE is 0. The Effective value of MDSELR_EL1.BANK is zero at EL2.
0b1	The Effective values of MDSCR_EL1.EMBWE and MDSELR_EL1.BANK are not affected by this field.

It is IMPLEMENTATION DEFINED whether this field is implemented or is RES0 when 16 or fewer breakpoints are implemented, 16 or fewer watchpoints are implemented, and MDSELR_EL1 is implemented as RAZ/WI.

If EL2 is not implemented or EL2 is disabled in the current Security state, then the Effective value of this field is 1, other than for a direct read of the register.

This field is ignored by the PE and treated as 0 when EL3 is implemented and MDCR_EL3.EBWE is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [42]

Reserved, RES0.

PMEE, bits [41:40]

When FEAT_EBEP is implemented:

Performance Monitors Exception Enable. Controls the generation of the **PMUIRQ** signal and the PMU Profiling exception at EL0, EL1, and EL2.

PMEE	Meaning
0b00	The **PMUIRQ** signal is asserted on a PMU overflow, and the PMU Profiling exception is disabled.
0b01	The **PMUIRQ** signal and the PMU Profiling exception are both controlled by PMECR_EL1.PMEE.
0b11	The **PMUIRQ** signal is deasserted, and the PMU Profiling exception is enabled.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

All other values are reserved.

If EL2 is not implemented or EL2 is disabled in the current Security state, then the Effective value of this field is `0b01`, other than for a direct read of the register.

This field is ignored by the PE when all of the following are true:

- EL3 is implemented.
- MDCR_EL3.PMEE != `0b01`.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'00'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [39:37]

Reserved, RES0.

HPMFZS, bit [36]

When FEAT_SPEv1p2 is implemented:

Hyp Performance Monitors Freeze-on-SPE event. Stop counters when PMBLIMITR_EL1.{PMFZ,E} is {1,1} and profiling is stopped.

HPMFZS	Meaning
`0b0`	Do not freeze on a Statistical Profiling Buffer Management event.
`0b1`	Affected counters do not count following a Statistical Profiling Buffer Management event.

The pseudocode function `SPEProfilingStopped` describes when profiling is stopped.

The counters affected by this field are the event counters in the second range. This applies even when EL2 is disabled in the current Security state.

Other event counters and PMCCNTR_EL0 are not affected by this field.

When FEAT_PMUv3_ICNTR is implemented, PMICNTR_EL0 is not affected by this field.

If MDCR_EL2.HPMN is equal to `GetNumEventCountersSelfHosted()`, then this field has no effect.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [35:32]

Reserved, RES0.

PMSSE, bits [31:30]

When FEAT_PMUv3_SS is implemented:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Performance Monitors Snapshot Enable. Controls the generation of Capture events.

PMSSE	Meaning
0b00	Capture events are disabled.
0b01	Capture events are controlled by PMECR_EL1.SSE.
0b10	Capture events are enabled and prohibited.
0b11	Capture events are enabled and allowed.

If EL2 is not implemented, then the Effective value of this field is 0b01.

The reset behavior of this field is:

- On a Cold reset:
  - When the highest implemented Exception level is EL2, this field resets to '00'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPMFZO, bit [29]

When FEAT_PMUv3p7 is implemented:

Hyp Performance Monitors Freeze-on-overflow. Stop event counters on overflow.

HPMFZO	Meaning
0b0	Do not freeze on overflow.
0b1	Affected counters do not count when all of the following are true for any event counter PMEVCNTR<m>_EL0 in the second range: • PMOVSCLR_EL0[m] is 1. • Either FEAT_SEBEP is not implemented or PMEVTYPER<m>_EL0.SYNC is 0.

The counters affected by this field are the event counters in the second range. This applies even when EL2 is disabled in the current Security state.

Other event counters and PMCCNTR_EL0 are not affected by this field.

When FEAT_PMUv3_ICNTR is implemented, PMICNTR_EL0 is not affected by this field.

If MDCR_EL2.HPMN is equal to GetNumEventCountersSelfHosted(), then this field has no effect.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MTPME, bit [28]

When FEAT_MTPMU is implemented and EL3 is not implemented:

Multi-threaded PMU Enable. Enables use of the PMEVTYPER<n>_EL0.MT bits.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

MTPME	Meaning
0b0	FEAT_MTPMU is disabled. The Effective value of PMEVTYPER<n>_EL0.MT is 0.
0b1	PMEVTYPER<n>_EL0.MT bits not affected by this field.

If FEAT_MTPMU is disabled for any other PE in the system that has the same level 1 Affinity as the PE, it is IMPLEMENTATION DEFINED whether the PE behaves as if this field is 0.

The reset behavior of this field is:

- On a Cold reset, this field resets to '1'.

Otherwise:

Reserved, RES0.

TDCC, bit [27]

When FEAT_FGT is implemented:

Trap DCC. Traps use of the Debug Comms Channel at EL1 and EL0 to EL2.

TDCC	Meaning
0b0	This control does not cause any register accesses to be trapped.
0b1	If EL2 is implemented and enabled in the current Security state, accesses to the DCC registers at EL1 and EL0 generate a Trap exception to EL2, unless the access also generates a higher priority exception. Traps on the DCC data transfer registers are ignored when the PE is in Debug state.

The DCC registers trapped by this control are:

AArch64: OSDTRRX_EL1, OSDTRTX_EL1, MDCCSR_EL0, MDCCINT_EL1, and, when the PE is in Non-debug state, DBGDTR_EL0, DBGDTRRX_EL0, and DBGDTRTX_EL0.

AArch32: DBGDTRRXext, DBGDTRTXext, DBGDSCRint, DBGDCCINT, and, when the PE is in Non-debug state, DBGDTRRXint and DBGDTRTXint.

The traps are reported with EC syndrome value:

- 0x05 for trapped AArch32 MRC and MCR accesses with coproc == 0b1110.
- 0x06 for trapped AArch32 LDC to DBGDTRTXint and STC from DBGDTRRXint.
- 0x18 for trapped AArch64 MRS and MSR accesses.

When the PE is in Debug state, MDCR_EL2.TDCC does not trap any accesses to:

AArch64: DBGDTR_EL0, DBGDTRRX_EL0, and DBGDTRTX_EL0.

AArch32: DBGDTRRXint and DBGDTRTXint.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HLP, bit [26]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_PMUv3p5 is implemented:

Hypervisor Long Event Counter Enable. Determines which event counter bit generates an overflow recorded by PMOVSR[n].

HLP	Meaning
0b0	Affected counters overflow on increment that causes unsigned overflow of PMEVCNTR<n>_EL0[31:0].
0b1	Affected counters overflow on increment that causes unsigned overflow of PMEVCNTR<n>_EL0[63:0].

When FEAT_EBEP is implemented and the PMU Profiling exception is enabled, the Effective value of this field is 1.

The counters affected by this field are the event counters in the second range. This applies even when EL2 is disabled in the current Security state.

The following are not affected by this field:

- Other event counters.
- PMCCNTR_EL0.
- If FEAT_PMUv3_ICNTR is implemented, PMICNTR_EL0.

If MDCR_EL2.HPMN is equal to `GetNumEventCountersSelfHosted()`, then this field has no effect.

For more information see the description of MDCR_EL2.HPMN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E2TB, bits [25:24]

When FEAT_TRBE is implemented:

EL2 Trace Buffer.

If EL2 is implemented and enabled in the trace buffer owning Security state, then this field controls the trace buffer owning translation regime.

If EL2 is implemented and enabled in the current Security state, then this field controls access to trace buffer System registers from EL1.

E2TB	Meaning
0b00	If EL2 is implemented and enabled in the trace buffer owning Security state, then the trace buffer owning Exception level is EL2. Otherwise, the trace buffer owning Exception level is EL1 and, if `TraceBufferEnabled()` == TRUE, tracing is prohibited at EL2. If EL2 is implemented and enabled in the current Security state, then accesses to trace buffer System registers at EL1 are trapped to EL2, unless the access generates a higher priority exception.
0b10	Trace buffer owning Exception level is EL1. If `TraceBufferEnabled()` == TRUE, then tracing is prohibited at EL2. If EL2 is implemented and enabled in the current Security state, then accesses to trace buffer System registers at EL1 are trapped to EL2, unless the access generates a higher priority exception.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E2TB	Meaning
0b11	Trace buffer owning Exception level is EL1. If TraceBufferEnabled() == TRUE, then tracing is prohibited at EL2. Accesses to trace buffer System registers at EL1 are not trapped by this mechanism.

All other values are reserved.

In AArch64 state, the instructions affected by this control are:

- MRS and MSR accesses to TRBBASER_EL1, TRBLIMITR_EL1, TRBMAR_EL1, TRBPTR_EL1, TRBSR_EL1, and TRBTRG_EL1.
- If FEAT_TRBE_MPAM is implemented, MRS and MSR accesses to TRBMPAM_EL1.
- If FEAT_TRBE_EXC is implemented, MRS and MSR accesses to TRBSR_EL2 and TRBSR_EL12.

Unless the instruction generates a higher priority exception, trapped instructions generate an exception to EL2, reported using EC syndrome value 0x18.

When TraceBufferEnabled()==FALSE, this field has no effect on whether tracing is prohibited.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HCCD, bit [23]

When FEAT_PMUv3p5 is implemented:

Hypervisor Cycle Counter Disable. Prohibits PMCCNTR_EL0 from counting at EL2.

HCCD	Meaning
0b0	Cycle counting by PMCCNTR_EL0 is not affected by this mechanism.
0b1	Cycle counting by PMCCNTR_EL0 is prohibited at EL2.

This field does not affect the CPU_CYCLES event or any other event that counts cycles.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

Bits [22:20]

Reserved, RES0.

TTRF, bit [19]

When FEAT_TRF is implemented:

Traps use of the Trace Filter Control registers at EL1 to EL2, as follows:

- Access to TRFCR_EL1 is trapped to EL2, reported using EC syndrome value 0x18.
- Access to TRFCR is trapped to EL2, reported using EC syndrome value 0x03.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TTRF	Meaning
0b0	Accesses to the specified registers at EL1 are not affected by this control.
0b1	Accesses to the specified registers at EL1 generate a trap exception to EL2 when EL2 is enabled in the current Security state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [18]

Reserved, RES0.

HPMD, bit [17]

When FEAT_PMUv3p1 is implemented and FEAT_Debugv8p2 is implemented:

Guest Performance Monitors Disable. Controls PMU operation at EL2.

HPMD	Meaning
0b0	Counters are not affected by this mechanism.
0b1	Affected counters are prohibited from counting at EL2. If PMCR_EL0.DP is 1, then PMCCNTR_EL0 is disabled at EL2. Otherwise, PMCCNTR_EL0 is not affected by this mechanism.

The counters affected by this field are:

- Event counters in the first range.
- If FEAT_PMUv3_ICNTR is implemented, the instruction counter PMICNTR_EL0.
- If PMCR_EL0.DP is 1, the cycle counter PMCCNTR_EL0.

Other event counters are not affected by this field.

When PMCR_EL0.DP is 0, PMCCNTR_EL0 is not affected by this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

When FEAT_PMUv3p1 is implemented:

Guest Performance Monitors Disable. Controls PMU operation at EL2 when `ExternalSecureNoninvasiveDebugEnabled()` is FALSE.

HPMD	Meaning
0b0	Counters are not affected by this mechanism.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

HPMD	Meaning
0b1	If ExternalSecureNoninvasiveDebugEnabled() is FALSE, then all the following apply: • Affected event counters are prohibited from counting at EL2. • If PMCR_EL0.DP is 1, then PMCCNTR_EL0 is disabled at EL2. Otherwise, PMCCNTR_EL0 is not affected by this mechanism.

If ExternalSecureNoninvasiveDebugEnabled() is TRUE, then the event counters and PMCCNTR_EL0 are not affected by this field.

Otherwise, the counters affected by this field are:

- Event counters in the first range.
- If PMCR_EL0.DP is 1, the cycle counter, PMCCNTR_EL0.

Other event counters are not affected by this field. When PMCR_EL0.DP is 0, PMCCNTR_EL0 is not affected by this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

Bit [16]

Reserved, RES0.

EnSPM, bit [15]

When FEAT_SPMU is implemented:

Enable access to System PMU registers. When disabled, accesses to System PMU registers generate a trap to EL2.

EnSPM	Meaning
0b0	Accesses of the specified System PMU registers at EL1 and EL0 are trapped to EL2, unless the instruction generates a higher priority exception.
0b1	Accesses of the specified System PMU registers are not trapped by this mechanism.

In AArch64 state, the instructions affected by this control are: MRS and MSR accesses to SPMACCESSR_EL1, SPMCFGR_EL1, SPMCGCR<n>_EL1, SPMCNTENCLR_EL0, SPMCNTENSET_EL0, SPMCR_EL0, SPMDEVAFF_EL1, SPMDEVARCH_EL1, SPMEVCNTR<n>_EL0, SPMEVFILT2R<n>_EL0, SPMEVFILTR<n>_EL0, SPMEVTYPER<n>_EL0, SPMIIDR_EL1, SPMINTENCLR_EL1, SPMINTENSET_EL1, SPMOVSCLR_EL0, SPMOVSSET_EL0, SPMSCR_EL1, and SPMSELR_EL0.

Trapped instructions are reported using EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TPMS, bit [14]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_SPE is implemented:

Trap Performance Monitor Sampling. Enables a trap to EL2 on accesses of SPE registers.

TPMS	Meaning
0b0	Accesses of the specified SPE registers are not trapped by this mechanism.
0b1	Accesses of the specified SPE registers at EL1 are trapped to EL2, unless the instruction generates a higher priority exception.

In AArch64 state, the instructions affected by this control are:

- MRS and MSR accesses to PMSCR_EL1, PMSEVFR_EL1, PMSFCR_EL1, PMSICR_EL1, PMSIRR_EL1, and PMSLATFR_EL1.
- MRS accesses to PMSIDR_EL1.
- If FEAT_SPE_FnE is implemented, MRS and MSR accesses to PMSNEVFR_EL1.
- If FEAT_SPE_FDS is implemented, MRS and MSR accesses to PMSDSFR_EL1.

Trapped instructions are reported using EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E2PB, bits [13:12]

When FEAT_SPE is implemented:

EL2 Profiling Buffer.

If EL2 is implemented and enabled in the Profiling Buffer owning Security state, then this field controls the Profiling Buffer owning translation regime.

If EL2 is implemented and enabled in the current Security state, then this field controls access to Profiling Buffer System registers from EL1.

E2PB	Meaning
0b00	If EL2 is implemented and enabled in the Profiling Buffer owning Security state, then the Profiling Buffer owning Exception level is EL2. Otherwise, the Profiling Buffer owning Exception level is EL1 and, Profiling is prohibited at EL2. If EL2 is implemented and enabled in the current Security state, then accesses to Profiling Buffer System registers at EL1 are trapped to EL2, unless the access generates a higher priority exception.
0b10	Profiling Buffer owning Exception level is EL1. Profiling is prohibited at EL2. If EL2 is implemented and enabled in the current Security state, then accesses to Profiling Buffer System registers at EL1 are trapped to EL2, unless the access generates a higher priority exception.
0b11	Profiling Buffer owning Exception level is EL1. Profiling is prohibited at EL2. Accesses to Profiling Buffer System registers at EL1 are not trapped by this mechanism.

All other values are reserved.

In AArch64 state, the instructions affected by this control are:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- MRS and MSR accesses to PMBLIMITR_EL1, PMBPTR_EL1, and PMBSR_EL1.
- If FEAT_SPE_nVM is implemented, MRS and MSR accesses to PMBMAR_EL1.

Unless the instruction generates a higher priority exception, trapped instructions generate an exception to EL2, reported using EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TDRA, bit [11]

Trap Debug ROM Address register access. Traps System register accesses to the Debug ROM registers to EL2 when EL2 is enabled in the current Security state as follows:

- If EL1 is using AArch64, accesses to MDRAR_EL1 are trapped to EL2, reported using EC syndrome value 0x18.
- If EL0 or EL1 is using AArch32 state, MRC or MCR accesses to the following registers are trapped to EL2, reported using EC syndrome value 0x05 and MRRC or MCRR accesses are trapped to EL2, reported using EC syndrome value 0x0C:
  - DBGDRAR, DBGDSAR.

TDRA	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL0 and EL1 System register accesses to the Debug ROM registers are trapped to EL2 when EL2 is enabled in the current Security state, unless it is trapped by the following: • DBGDSCRext.UDCCdis. • MDSCR_EL1.TDCC.

This field is treated as being 1 for all purposes other than a direct read when one or more of the following are true:

- MDCR_EL2.TDE == 1.
- HCR_EL2.TGE == 1.

Note

EL2 does not provide traps on debug register accesses through the optional memory-mapped external debug interfaces.

System register accesses to the debug registers might have side-effects. When a System register access is trapped to EL2, no side-effects occur before the exception is taken to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TDOSA, bit [10]

When FEAT_DoubleLock is implemented:

Trap debug OS-related register access. Traps EL1 System register accesses to the powerdown debug registers to EL2, from both Execution states as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2, reported using EC syndrome value 0x18:
  - OSLAR_EL1, OSLSR_EL1, OSDLR_EL1, and DBGPRCR_EL1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- Any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this bit.
- In AArch32 state, accesses to the following registers are trapped to EL2, reported using EC syndrome value `0x05`:
  - DBGOSLSR, DBGOSLAR, DBGOSDLR, and DBGPRCR.
  - Any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this bit.

TDOSA	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	EL1 System register accesses to the powerdown debug registers are trapped to EL2 when EL2 is enabled in the current Security state.

Note

These registers are not accessible at EL0.

This field is treated as being 1 for all purposes other than a direct read when one or more of the following are true:

- MDCR_EL2.TDE == 1.
- HCR_EL2.TGE == 1.

System register accesses to the debug registers might have side-effects. When a System register access is trapped to EL2, no side-effects occur before the exception is taken to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Trap debug OS-related register access. Traps EL1 System register accesses to the powerdown debug registers to EL2, from both Execution states as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2, reported using EC syndrome value `0x18`:
  - OSLAR_EL1, OSLSR_EL1, and DBGPRCR_EL1.
  - Any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this bit.
- In AArch32 state, accesses to the following registers are trapped to EL2, reported using EC syndrome value `0x05`:
  - DBGOSLSR, DBGOSLAR, and DBGPRCR.
  - Any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this bit.

It is IMPLEMENTATION DEFINED whether accesses to OSDLR_EL1 are trapped.

It is IMPLEMENTATION DEFINED whether accesses to DBGOSDLR are trapped.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TDOSA	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL1 System register accesses to the powerdown debug registers are trapped to EL2 when EL2 is enabled in the current Security state.

Note

These registers are not accessible at EL0.

This field is treated as being 1 for all purposes other than a direct read when one or more of the following are true:

- MDCR_EL2.TDE == 1.
- HCR_EL2.TGE == 1.

Note

EL2 does not provide traps on debug register accesses through the optional memory-mapped external debug interfaces.

System register accesses to the debug registers might have side-effects. When a System register access is trapped to EL2, no side-effects occur before the exception is taken to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TDA, bit [9]

Trap accesses of debug System registers. Enables a trap to EL2 on accesses of debug System registers.

TDA	Meaning
0b0	Accesses of the specified debug System registers are not trapped by this mechanism.
0b1	Accesses of the specified debug System registers at EL1 and EL0 are trapped to EL2, unless the instruction generates a higher priority exception.

In AArch64 state, the instructions affected by this control are:

- MRS and MSR accesses to DBGAUTHSTATUS_EL1, DBGBCR<n>_EL1, DBGBVR<n>_EL1, DBGCLAIMCLR_EL1, DBGCLAIMSET_EL1, DBGWCR<n>_EL1, DBGWVR<n>_EL1, MDCCINT_EL1, MDCCSR_EL0, MDSCR_EL1, OSDTRRX_EL1, OSDTRTX_EL1, and OSECCR_EL1.
- If FEAT_Debugv8p9 is implemented, MRS and MSR accesses to MDSELR_EL1.
- If FEAT_STEP2 is implemented, MRS and MSR accesses to MDSTEPOP_EL1.
- In Non-debug state, MRS accesses to DBGDTRRX_EL0 and DBGDTR_EL0 and MSR accesses to DBGDTRTX_EL0 and DBGDTR_EL0.

In AArch32 state, the instructions affected by this control are:

- MRC and MCR accesses to DBGAUTHSTATUS, DBGBCR<n>, DBGBVR<n>, DBGBXVR<n>, DBGCLAIMCLR, DBGCLAIMSET, DBGDCCINT, DBGDEVID, DBGDEVID1, DBGDEVID2, DBGDIDR, DBGDSCRext, DBGDSCRint, DBGDTRRXext, DBGDTRTXext, DBGOSECCR, DBGVCR, DBGWCR<n>, DBGWFAR, and DBGWVR<n>.
- STC accesses to DBGDTRRXint and LDC accesses to DBGDTRTXint.
- In Non-debug state, MRC accesses to DBGDTRRXint and MCR accesses to DBGDTRTXint.

Trapped AArch64 instructions are reported using EC syndrome value 0x18.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Trapped AArch32 instructions are reported using EC syndrome value `0x05` for `MRC` and `MCR` accesses, and `0x06` for `LDC` and `STC` accesses.

The following instructions are not trapped in Debug state:

- AArch64 `MRS` accesses to DBGDTRRX_EL0 and DBGDTR_EL0 and `MSR` accesses to DBGDTRTX_EL0 and DBGDTR_EL0.
- AArch32 `MRC` accesses to DBGDTRRXint and `MCR` accesses to DBGDTRTXint.

If 16 or fewer breakpoints and 16 or fewer watchpoints are implemented, and MDSELR_EL1 is implemented as RAZ/WI, then it is IMPLEMENTATION DEFINED whether AArch64 accesses to MDSELR_EL1 are trapped to EL2 when MDCR_EL2.TDA is 1.

This field is ignored by the PE and treated as one when any of the following are true:

- MDCR_EL2.TDE == 1.
- HCR_EL2.TGE == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TDE, bit [8]

Trap Debug Exceptions. Controls routing of Debug exceptions, and defines the debug target Exception level, EL_D.

TDE	Meaning
`0b0`	The debug target Exception level is EL1.
`0b1`	If EL2 is enabled for the current Effective value of SCR_EL3.NS, the debug target Exception level is EL2, otherwise the debug target Exception level is EL1. The MDCR_EL2.{TDRA, TDOSA, TDA} fields are treated as being 1 for all purposes other than returning the result of a direct read of the register.

For more information, see 'Routing debug exceptions'.

This field is treated as being 1 for all purposes other than a direct read when HCR_EL2.TGE == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

HPME, bit [7]

When FEAT_PMUv3 is implemented:

Hyp Enable.

HPME	Meaning
`0b0`	Affected counters are disabled and do not count.
`0b1`	Affected counters are enabled by PMCNTENSET_EL0.

The counters affected by this field are the event counters in the second range. This applies even when EL2 is disabled in the current Security state.

The following counters are not affected by this field:

- Other event counters.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- PMCCNTR_EL0.
- If FEAT_PMUv3_ICNTR is implemented, PMICNTR_EL0.

If MDCR_EL2.HPMN is equal to `GetNumEventCountersSelfHosted()`, then this field has no effect.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TPM, bit [6]

When FEAT_PMUv3 is implemented:

Trap accesses of PMU registers. Enables a trap to EL2 on accesses of PMU registers.

TPM	Meaning
`0b0`	Accesses of the specified PMU registers are not trapped by this mechanism.
`0b1`	Accesses of the specified PMU registers at EL1 and EL0 are trapped to EL2, unless the instruction generates a higher priority exception.

In AArch64 state, the instructions affected by this control are:

- `MRS` and `MSR` accesses to PMCCFILTR_EL0, PMCCNTR_EL0, PMCNTENCLR_EL0, PMCNTENSET_EL0, PMCR_EL0, PMEVCNTR<n>_EL0, PMEVTYPER<n>_EL0, PMINTENCLR_EL1, PMINTENSET_EL1, PMOVSCLR_EL0, PMOVSSET_EL0, PMSELR_EL0, PMSWINC_EL0, PMUSERENR_EL0, PMXEVCNTR_EL0, and PMXEVTYPER_EL0.
- `MRS` accesses to PMCEID0_EL0 and PMCEID1_EL0.
- If FEAT_PMUv3p4 is implemented, `MRS` accesses to PMMIR_EL1.
- If FEAT_PMUv3p9 is implemented, `MSR` accesses to PMZR_EL0.
- If FEAT_PMUv3_ICNTR is implemented, `MRS` accesses to PMICFILTR_EL0 and PMICNTR_EL0.
- If FEAT_EBEP is implemented or FEAT_PMUv3_SS is implemented, `MRS` and `MSR` accesses to PMECR_EL1.
- If FEAT_SEBEP is implemented, `MRS` and `MSR` accesses to PMIAR_EL1.

In AArch32 state, the instructions affected by this control are:

- `MRC` and `MCR` accesses to PMCCFILTR, PMCCNTR, PMCNTENCLR, PMCNTENSET, PMCR, PMEVCNTR<n>, PMEVTYPER<n>, PMINTENCLR, PMINTENSET, PMOVSR, PMOVSSET, PMSELR, PMSWINC, PMUSERENR, PMXEVCNTR, and PMXEVTYPER.
- `MRC` accesses to PMCEID0 and PMCEID1.
- `MRRC` and `MCRR` accesses to PMCCNTR.
- If FEAT_PMUv3p1 is implemented, `MRC` accesses to PMCEID2 and PMCEID3.
- If FEAT_PMUv3p4 is implemented, `MRC` accesses to PMMIR.

Trapped AArch64 instructions are reported using EC syndrome value `0x18`.

Trapped AArch32 instructions are reported using EC syndrome value `0x03` for `MRC` and `MCR` accesses, and `0x04` for `MRRC` and `MCRR` accesses.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TPMCR, bit [5]

When FEAT_PMUv3 is implemented:

Trap PMCR_EL0 or PMCR accesses. Traps EL0 and EL1 accesses to EL2, when EL2 is enabled in the current Security state, as follows:

- In AArch64 state, accesses to PMCR_EL0 are trapped to EL2, reported using EC syndrome value `0x18`.
- In AArch32 state, accesses to PMCR are trapped to EL2, reported using EC syndrome value `0x03`.

TPMCR	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	EL0 and EL1 accesses to the specified registers are trapped to EL2 when EL2 is enabled in the current Security state, unless they are trapped by the following: • PMUSERENR.EN. • PMUSERENR_EL0.EN.

Note

EL2 does not provide traps on Performance Monitor register accesses through the optional memory-mapped external debug interface.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPMN, bits [4:0]

When FEAT_PMUv3 is implemented:

Defines the number of event counters PMEVCNTR<n>_EL0 and, if FEAT_PMUv3_SS is implemented, event counter snapshot registers PMEVCNTSVR<n>_EL1, that are accessible from EL1 and, if permitted, from EL0.

MDCR_EL2.HPMN divides the event counters accessible from self-hosted software into a first range and a second range.

When FEAT_PMUv3_EXTPMN is implemented, all of the following apply:

- PMCCR.EPMN divides the `NUM_PMU_COUNTERS` event counters implemented by the PE into the event counters that MDCR_EL2.HPMN divides into the first and second ranges, and a third range that is inaccessible from self-hosted software.
- If MDCR_EL2.HPMN is not 0 and is less than the Effective value of PMCCR.EPMN, then event counters [0..(MDCR_EL2.HPMN-1)] are in the first range, and the remaining event counters [MDCR_EL2.HPMN..(PMCCR.EPMN-1)] are in the second range.
- The pseudcode function `GetNumEventCountersSelfHosted()` returns the Effective value of PMCCR.EPMN.

When FEAT_PMUv3_EXTPMN is not implemented, all of the following apply:

- All of the `NUM_PMU_COUNTERS` event counters are accessible to self-hosted software and no counters are in the third range.
- If MDCR_EL2.HPMN is not 0 and is less than `NUM_PMU_COUNTERS`, then event counters [0..(MDCR_EL2.HPMN-1)] are in the first range, and the remaining event counters [MDCR_EL2.HPMN..(`NUM_PMU_COUNTERS`-1)] are in the second range.
- The pseudocode function `GetNumEventCountersSelfHosted()` returns `NUM_PMU_COUNTERS`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

If FEAT_HPMN0 is implemented and MDCR_EL2.HPMN is 0, then all of the following apply:

- No event counters are in the first range.
- If FEAT_PMUv3_EXTPMN is implemented, then event counters [0..(PMCCR.EPMN-1)] are in the second range.
- If FEAT_PMUv3_EXTPMN is not implemented, then all event counters are in the second range.

If MDCR_EL2.HPMN is equal to `GetNumEventCountersSelfHosted()`, or EL2 is not implemented, then all of the following apply:

- If FEAT_PMUv3_EXTPMN is implemented, then event counters [0..(PMCCR.EPMN-1)] are in the first range.
- If FEAT_PMUv3_EXTPMN is not implemented, then all event counters are in the first range.
- No event counters are in the second range.

All of the following apply for an event counter PMEVCNTR<n>_EL0 in the first range:

- The counter is accessible from EL1, EL2, and EL3.
- The counter is accessible from EL0 if permitted by PMUSERENR_EL0 and PMUACR_EL1, or by PMUSERENR.
- If FEAT_PMUv3p5 is implemented, then PMCR_EL0.LP or PMCR.LP determines whether the counter overflow flag PMOVSSET_EL0[n] is set on unsigned overflow of PMEVCNTR<n>_EL0[31:0] or PMEVCNTR<n>_EL0[63:0].
- PMCR_EL0.E and PMCNTENSET_EL0[n] enable the operation of the event counter.

All of the following apply for an event counter PMEVCNTR<n>_EL0 in the second range:

- The counter is accessible from EL2 and EL3.
- If EL2 is disabled in the current Security state, then the event counter is accessible from EL1, and from EL0 if permitted by PMUSERENR_EL0 and PMUACR_EL1, or by PMUSERENR.
- If FEAT_PMUv3p5 is implemented, MDCR_EL2.HLP determines whether the counter overflow flag PMOVSSET_EL0[n] is set on unsigned overflow of PMEVCNTR<n>_EL0[31:0] or PMEVCNTR<n>_EL0[63:0].
- MDCR_EL2.HPME and PMCNTENSET_EL0[n] enable the operation of the event counter.

If FEAT_PMUv3_SS is implemented, then all of the following apply:

- For an event counter snapshot register PMEVCNTSVR<n>_EL1 in the first range, the register is accessible from EL1, EL2, and EL3.

- For an event counter snapshot register PMEVCNTSVR<n>_EL1 in the second range, the register is accessible from EL2 and EL3. If EL2 is disabled in the current Security state, the event counter is also accessible from EL1.

For information about counters in the third range, see the description of PMCCR.EPMN.

Values greater than `GetNumEventCountersSelfHosted()` are reserved. If FEAT_HPMN0 is not implemented, then the value 0 is reserved.

When FEAT_PMUv3_EXTPMN is not implemented and this field is set to a reserved value, the following CONSTRAINED UNPREDICTABLE behaviors apply:

- The value returned by a direct read of MDCR_EL2.HPMN is UNKNOWN.
- The number of event counters in each of the first and second ranges is UNKNOWN. That is, either:
  - The PE behaves as if MDCR_EL2.HPMN is set to an UNKNOWN nonzero value less than or equal to `NUM_PMU_COUNTERS`.
  - All counters are in the second range and none are in the first range.

When FEAT_PMUv3_EXTPMN is implemented and this field is set to a reserved value, the following behaviors apply:

- The value returned by a direct read of MDCR_EL2.HPMN is the Effective value of PMCCR.EPMN.
- No event counters are in the second range.
- The value returned by an indirect read of MDCR_EL2.HPMN as a result of direct reads of PMCR_EL0.N or PMCR.N is the Effective value of PMCCR.EPMN.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to the expression `NUM_PMU_COUNTERS`.

Otherwise:

Reserved, RES0.

Accessing MDCR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MDCR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MDCR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MDCR_EL2;
```

```
MSR MDCR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MDCR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MDCR_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.3.18 MDCR_EL3, Monitor Debug Configuration Register (EL3)

The MDCR_EL3 characteristics are:

Purpose

Provides EL3 configuration options for self-hosted debug and the Performance Monitors Extension.

Configuration

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to MDCR_EL3 are UNDEFINED.

Attributes

MDCR_EL3 is a 64-bit register.

Field descriptions

Bits [63:56]

Reserved, RES0.

EnPMS4, bit [55]

When FEAT_SPE_nVM is implemented:

Enable access to SPE registers. When disabled, accesses to SPE registers generate a trap to EL3.

EnPMS4	Meaning
0b0	Accesses of the specified SPE registers at EL2 and EL1 are trapped to EL3, unless the instruction generates a higher priority exception.
0b1	Accesses of the specified SPE registers are not trapped by this mechanism.

In AArch64 state, the instructions affected by this control are: MRS and MSR accesses to PMBMAR_EL1.

Trapped instructions are reported using EC syndrome value 0x18.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRBEE, bits [54:53]

When FEAT_TRBE_EXC is implemented:

Trace buffer management event Exception Enable.

TRBEE	Meaning
0b00	Disabled. TRBE Profiling exceptions for all Exception levels are disabled. All of the following apply: • No trace buffer management events are recorded in TRBSR_EL3. • TRBE Profiling exceptions are not generated. • TRBSR_EL1.IRQ drives the interrupt request signal **TRBIRQ**. • Accesses to TRBSR_EL2 at EL2 are trapped to EL3. • Accesses to TRBSR_EL1 at EL1 ignore the value of HCR_EL2.NV1 and accesses to TRBSR_EL1 at EL2 ignore the value of HCR_EL2.E2H.
0b01	Delegated. TRBE Profiling exceptions for EL3 are disabled, but might be enabled for EL2 or EL1 by TRFCR_EL2.EE or TRFCR_EL1.EE. All of the following apply: • No trace buffer management events are recorded in TRBSR_EL3. • TRBSR_EL3.IRQ is ignored and TRBE Profiling exceptions are not taken to EL3.

0b10	Enabled. TRBE Profiling exceptions for EL3 are enabled for trace buffer management events targeting EL3, as follows:
	• Trace buffer management events due to a fault on a write to the trace buffer that would generate a Data Abort exception taken to EL3 if generated by a store instruction executed at the owning Exception level are recorded in TRBSR_EL3. That is, any of the following faults:
	  • Granule Protection Check faults other than Granule Protection Faults (GPFs).
	  • If SCR_EL3.GPF is 1, GPFs.
	  • If SCR_EL3.EA is 1, External aborts.
	• TRBE Profiling exceptions are generated and taken to EL3 when unmasked and TRBSR_EL3.IRQ is 1.

0b11	Trap all. TRBE Profiling exceptions for EL3 are enabled for all trace buffer management events, as follows: • All trace buffer management events are recorded in TRBSR_EL3. • TRBE Profiling exceptions are generated and taken to EL3 when unmasked and TRBSR_EL3.IRQ is 1.

If EL3 is not implemented, then the Effective value of MDCR_EL3.TRBEE is 0b01.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3, this field resets to '00'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMSEE, bits [52:51]

When FEAT_SPE_EXC is implemented:

Profiling Buffer management event Exception Enable.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

PMSEE	Meaning
0b00	Disabled. SPE Profiling exceptions for all Exception levels are disabled. All of the following apply: • No Profiling Buffer management events are recorded in PMBSR_EL3. • SPE Profiling exceptions are not generated. • PMBSR_EL1.S drives the interrupt request signal **PMBIRQ**. • Accesses to PMBSR_EL2 at EL2 are trapped to EL3. • Accesses to PMBSR_EL1 at EL1 ignore the value of HCR_EL2.NV1 and accesses to PMBSR_EL1 at EL2 ignore the value of HCR_EL2.E2H.
0b01	Delegated. SPE Profiling exceptions for EL3 are disabled, but might be enabled for EL2 or EL1 by PMSCR_EL2.EE or PMSCR_EL1.EE. All of the following apply: • No Profiling Buffer management events are recorded in PMBSR_EL3. • PMBSR_EL3.S is ignored and SPE Profiling exceptions are not taken to EL3.

0b10	Enabled. SPE Profiling exceptions for EL3 are enabled for Profiling Buffer management events targeting EL3, as follows:
	• Profiling Buffer management events due to a fault on a write to the Profiling Buffer that would generate a Data Abort exception taken to EL3 if generated by a store instruction executed at the owning Exception level are recorded in PMBSR_EL3. That is, any of the following faults:
	  • Granule Protection Check faults other than Granule Protection Faults (GPFs).
	  • If SCR_EL3.GPF is 1, GPFs.
	  • If SCR_EL3.EA is 1, External aborts.
	• SPE Profiling exceptions are generated and taken to EL3 when unmasked and PMBSR_EL3.S is 1.

0b11	Trap all. SPE Profiling exceptions for EL3 are enabled for all Profiling Buffer management events, as follows: • All Profiling Buffer management events are recorded in PMBSR_EL3. • SPE Profiling exceptions are generated and taken to EL3 when unmasked and PMBSR_EL3.S is 1.

If EL3 is not implemented, then the Effective value of MDCR_EL3.PMSEE is 0b01.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3, this field resets to '00'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnSTEPOP, bit [50]

When FEAT_STEP2 is implemented:

EnSTEPOP	Meaning
0b0	Execution from MDSTEPOP_EL1 is disabled. EL2, EL1 and EL0 System register accesses to MDSTEPOP_EL1 are trapped to EL3, unless the access generates a higher priority exception.
0b1	Execution from MDSTEPOP_EL1 is not disabled by this control. System register accesses to MDSTEPOP_EL1 are not trapped by this mechanism.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ETBAD, bits [49:48]

When FEAT_TRBE_EXT is implemented:

External Trace Buffer Access Disable. Controls access to the Trace Buffer registers from an external debugger.

ETBAD	Meaning	Applies when
0b00	Non-secure accesses from an external debugger to Trace Buffer registers are prohibited. If FEAT_RME is implemented, Secure and Realm accesses from an external debugger to Trace Buffer registers are prohibited and Root accesses to Trace Buffer registers are allowed. If FEAT_RME is not implemented, Secure accesses to Trace Buffer registers are allowed.	
0b01	Secure and Non-secure accesses from an external debugger to Trace Buffer registers are prohibited. Root and Realm accesses to Trace Buffer registers are allowed.	FEAT_RME is implemented
0b10	Realm and Non-secure accesses from an external debugger to Trace Buffer registers are prohibited. Root and Secure accesses to Trace Buffer registers are allowed.	FEAT_RME is implemented
0b11	All accesses from an external debugger to Trace Buffer registers are allowed.	

If EL3 is not implemented, then the Effective value of this field is 0b11.

The reset behavior of this field is:

- On a Warm reset, this field resets to '00'.

Otherwise:

Reserved, RES0.

EnITE, bit [47]

When FEAT_ITE is implemented:

Enable access to Instrumentation trace registers. When disabled, accesses to Instrumentation trace registers generate a trap to EL3.

EnITE	Meaning
0b0	Accesses of the specified Instrumentation trace registers at EL2 and EL1 are trapped to EL3, unless the instruction generates a higher priority exception.
0b1	Accesses of the specified Instrumentation trace registers are not trapped by this mechanism.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

In AArch64 state, the instructions affected by this control are: `MRS` and `MSR` accesses to TRCITECR_EL1, TRCITECR_EL2, and TRCITECR_EL12.

Trapped instructions are reported using EC syndrome value `0x18`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EPMSSAD, bits [46:45]

When FEAT_PMUv3_SS is implemented:

External PMU Snapshot Access Disable. Controls access to the PMU Snapshot registers from an external debugger.

External accesses of the following registers are affected by this control:

- PMCCNTSVR_EL1, PMEVCNTSVR<n>_EL1, and PMSSCR_EL1.
- If FEAT_PMUv3_ICNTR is implemented, PMICNTSVR_EL1.

EPMSSAD	Meaning	Applies when
`0b00`	Non-secure accesses from an external debugger to the affected PMU Snapshot registers are prohibited. If FEAT_RME is implemented, Secure and Realm accesses from an external debugger to the affected PMU Snapshot registers are prohibited. Other accesses from an external debugger to the specified registers are not affected by this control.	
`0b01`	Secure and Non-secure accesses from an external debugger to the affected PMU Snapshot registers are prohibited. Other accesses from an external debugger to the specified registers are not affected by this control.	FEAT_RME is implemented
`0b10`	Realm and Non-secure accesses from an external debugger to the affected PMU Snapshot registers are prohibited. Other accesses from an external debugger to the specified registers are not affected by this control.	FEAT_RME is implemented
`0b11`	No accesses from an external debugger to the affected PMU Snapshot registers are prohibited by this control.	

If EL3 is not implemented, then the Effective value of this field is `0b11`.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'00'`.

Otherwise:

Reserved, RES0.

EnPMSS, bit [44]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_PMUv3_SS is implemented:

Enable access to PMU Snapshot registers. When disabled, accesses to PMU Snapshot registers generate a trap to EL3.

EnPMSS	Meaning
0b0	Accesses of the specified PMU Snapshot registers at EL2 and EL1 are trapped to EL3, unless the instruction generates a higher priority exception.
0b1	Accesses of the specified PMU Snapshot registers are not trapped by this mechanism.

In AArch64 state, the instructions affected by this control are:

- MRS and MSR accesses to PMCCNTSVR_EL1, PMEVCNTSVR<n>_EL1, and PMSSCR_EL1.
- If FEAT_PMUv3_ICNTR is implemented, MRS and MSR accesses to PMICNTSVR_EL1.

Trapped instructions are reported using EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EBWE, bit [43]

When FEAT_Debugv8p9 is implemented:

Extended Breakpoint and Watchpoint Enable. Enables use of additional breakpoints or watchpoints, and enables a trap to EL3 on accesses to debug registers.

EBWE	Meaning
0b0	The Effective values of MDSCR_EL1.EMBWE and MDCR_EL2.EBWE are 0. The Effective value of MDSELR_EL1.BANK is zero at EL3. Accesses of MDSELR_EL1 at EL2 and EL1 are trapped to EL3, unless the instruction generates a higher priority exception.
0b1	The Effective values of MDSCR_EL1.EMBWE, MDCR_EL2.EBWE, and MDSELR_EL1.BANK are not affected by this field. Accesses of MDSELR_EL1 are not trapped by this mechanism.

In AArch64 state, the instructions affected by this control are: MRS and MSR accesses to MDSELR_EL1.

Trapped instructions are reported using EC syndrome value 0x18.

It is IMPLEMENTATION DEFINED whether this field is implemented or is RES0 when 16 or fewer breakpoints are implemented, 16 or fewer watchpoints are implemented, and MDSELR_EL1 is implemented as RAZ/WI.

If EL3 is not implemented, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EnPMS3, bit [42]

When FEAT_SPE_FDS is implemented:

Enable access to SPE registers. When disabled, accesses to SPE registers generate a trap to EL3.

EnPMS3	Meaning
0b0	Accesses of the specified SPE registers at EL2 and EL1 are trapped to EL3, unless the instruction generates a higher priority exception.
0b1	Accesses of the specified SPE registers are not trapped by this mechanism.

In AArch64 state, the instructions affected by this control are: MRS and MSR accesses to PMSDSFR_EL1.

Trapped instructions are reported using EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMEE, bits [41:40]

When FEAT_EBEP is implemented:

Performance Monitors Exception Enable. Controls the generation of the **PMUIRQ** signal and the PMU Profiling exception at all Exception levels.

PMEE	Meaning
0b00	The **PMUIRQ** signal is asserted on a PMU overflow, and the PMU Profiling exception is disabled.
0b01	The **PMUIRQ** signal and the PMU Profiling exception are both controlled by MDCR_EL2.PMEE.
0b11	The **PMUIRQ** signal is deasserted, and the PMU Profiling exception is enabled.

All other values are reserved.

If EL3 is not implemented, then the Effective value of this field is 0b01.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3, this field resets to '00'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnTB2, bit [39]

When FEAT_TRBE_MPAM is implemented:

Enable access to Trace Buffer registers. When disabled, accesses to Trace Buffer registers generate a trap to EL3.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EnTB2	Meaning
0b0	Accesses of the specified Trace Buffer registers at EL2 and EL1 are trapped to EL3, unless the instruction generates a higher priority exception.
0b1	Accesses of the specified Trace Buffer registers are not trapped by this mechanism.

In AArch64 state, the instructions affected by this control are: MRS and MSR accesses to TRBMPAM_EL1.

Trapped instructions are reported using EC syndrome value 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E3BREC, bit [38]

When FEAT_BRBEv1p1 is implemented:

Branch Record Buffer EL3 Cold Reset Enable. With MDCR_EL3.E3BREW, controls branch recording at EL3.

E3BREC	Meaning
0b0	When MDCR_EL3.E3BREW == 0: Branch recording at EL3 is disabled. When MDCR_EL3.E3BREW == 1: Branch recording at EL3 is enabled.
0b1	When MDCR_EL3.E3BREW == 0: Branch recording at EL3 is enabled. When MDCR_EL3.E3BREW == 1: Branch recording at EL3 is disabled.

The reset behavior of this field is:

- On a Cold reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

E3BREW, bit [37]

When FEAT_BRBEv1p1 is implemented:

Branch Record Buffer EL3 Warm Reset Enable. With MDCR_EL3.E3BREC, controls branch recording at EL3.

For a description of the values derived by evaluating MDCR_EL3.E3BREC and MDCR_EL3.E3BREW together, see MDCR_EL3.E3BREC.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

EnPMSN, bit [36]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_SPE_FnE is implemented:

Trap accesses to PMSNEVFR_EL1. Controls access to Statistical Profiling PMSNEVFR_EL1 System register from EL2 and EL1.

EnPMSN	Meaning
`0b0`	Accesses to PMSNEVFR_EL1 at EL2 and EL1 generate a Trap exception to EL3.
`0b1`	Do not trap PMSNEVFR_EL1 to EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MPMX, bit [35]

When FEAT_PMUv3p7 is implemented:

Monitor Performance Monitors Extended control. With MDCR_EL3.SPME, controls PMU operation at EL3.

MPMX	Meaning
`0b0`	Counters are not affected by this mechanism.
`0b1`	Affected counters are prohibited from counting at EL3. If PMCR_EL0.DP is 1, PMCCNTR_EL0 is disabled at EL3. Otherwise, PMCCNTR_EL0 is not affected by this mechanism.

The counters affected by this field are:

- If EL2 is implemented and MDCR_EL3.SPME is 1, event counters PMEVCNTR<n>_EL0 in the first range.
- If EL2 is not implemented or MDCR_EL3.SPME is 0, event counters in the first and second ranges.
- If FEAT_PMUv3_ICNTR is implemented, the instruction counter, PMICNTR_EL0.
- If PMCR_EL0.DP is 1, the cycle counter, PMCCNTR_EL0.

Other event counters are not affected by this field. When PMCR_EL0.DP is 0, PMCCNTR_EL0 is not affected by this field.

For more information about event counter ranges, see MDCR_EL2.HPMN.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Otherwise:

Reserved, RES0.

MCCD, bit [34]

When FEAT_PMUv3p7 is implemented:

Monitor Cycle Counter Disable. Prohibits the Cycle Counter, PMCCNTR_EL0, from counting at EL3.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

MCCD	Meaning
0b0	Cycle counting by PMCCNTR_EL0 is not affected by this mechanism.
0b1	Cycle counting by PMCCNTR_EL0 is prohibited at EL3.

This field does not affect the CPU_CYCLES event or any other event that counts cycles.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

SBRBE, bits [33:32]

When FEAT_BRBE is implemented:

Secure Branch Record Buffer Enable. Controls branch recording by the BRBE, and access to BRBE registers and instructions at EL2 and EL1.

SBRBE	Meaning
0b00	Direct accesses to BRBE registers and instructions, except when in EL3, generate a Trap exception to EL3. EL0, EL1, and EL2 are prohibited regions.
0b01	Direct accesses to BRBE registers and instructions in Secure state, except when in EL3, generate a Trap exception to EL3. EL0, EL1, and EL2 in Secure state are prohibited regions. This control does not cause any direct accesses to BRBE registers when not in Secure state to be trapped, and does not cause any Exception levels when not in Secure state to be a prohibited region.
0b10	Direct accesses to BRBE registers and instructions, except when in EL3, generate a Trap exception to EL3. This control does not cause any Exception levels to be prohibited regions.
0b11	This control does not cause any direct accesses to BRBE registers or instruction to be trapped, and does not cause any Exception levels to be a prohibited region.

The Branch Record Buffer registers trapped by this control are: BRBCR_EL1, BRBCR_EL2, BRBCR_EL12, BRBFCR_EL1, BRBIDR0_EL1, BRBINF<n>_EL1, BRBINFINJ_EL1, BRBSRC<n>_EL1, BRBSRCINJ_EL1, BRBTGT<n>_EL1, BRBTGTINJ_EL1, and BRBTS_EL1.

The Branch Record Buffer instructions trapped by this control are:

- BRB IALL.
- BRB INJ.

Note

If FEAT_BRBEv1p1 is not implemented, EL3 is a prohibited region.

If EL3 is not implemented, then the Effective value of this field is 0b11.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

PMSSE, bits [31:30]

When FEAT_PMUv3_SS is implemented:

Performance Monitors Snapshot Enable. Controls the generation of Capture events.

PMSSE	Meaning
`0b00`	Capture events are disabled.
`0b01`	Capture events are controlled by MDCR_EL2.PMSSE.
`0b10`	Capture events are enabled and prohibited.
`0b11`	Capture events are enabled and allowed.

If EL3 is not implemented, then the Effective value of this field is `0b01`.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'00'`.

Otherwise:

Reserved, RES0.

Bit [29]

Reserved, RES0.

MTPME, bit [28]

When FEAT_MTPMU is implemented:

Multi-threaded PMU Enable. Enables use of the PMEVTYPER<n>_EL0.MT bits.

MTPME	Meaning
`0b0`	FEAT_MTPMU is disabled. The Effective value of PMEVTYPER<n>_EL0.MT is 0.
`0b1`	PMEVTYPER<n>_EL0.MT bits not affected by this field.

If FEAT_MTPMU is disabled for any other PE in the system that has the same level 1 Affinity as the PE, it is IMPLEMENTATION DEFINED whether the PE behaves as if this field is 0.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'1'`.

Otherwise:

Reserved, RES0.

TDCC, bit [27]

When FEAT_FGT is implemented:

Trap DCC. Traps use of the Debug Comms Channel at EL2, EL1, and EL0 to EL3.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TDCC	Meaning
0b0	This control does not cause any register accesses to be trapped.
0b1	Accesses to the DCC registers at EL2, EL1, and EL0 generate a Trap exception to EL3, unless the access also generates a higher priority exception. Traps on the DCC data transfer registers are ignored when the PE is in Debug state.

The DCC registers trapped by this control are:

AArch64: OSDTRRX_EL1, OSDTRTX_EL1, MDCCSR_EL0, MDCCINT_EL1, and, when the PE is in Non-debug state, DBGDTR_EL0, DBGDTRRX_EL0, and DBGDTRTX_EL0.

AArch32: DBGDTRRXext, DBGDTRTXext, DBGDSCRint, DBGDCCINT, and, when the PE is in Non-debug state, DBGDTRRXint and DBGDTRTXint.

The traps are reported with EC syndrome value:

- 0x05 for trapped AArch32 MRC and MCR accesses with coproc == 0b1110.
- 0x06 for trapped AArch32 LDC to DBGDTRTXint and STC from DBGDTRRXint.
- 0x18 for trapped AArch64 MRS and MSR accesses.

When the PE is in Debug state, MDCR_EL3.TDCC does not trap any accesses to:

AArch64: DBGDTR_EL0, DBGDTRRX_EL0, and DBGDTRTX_EL0.

AArch32: DBGDTRRXint and DBGDTRTXint.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSTBE, bit [26]

When FEAT_TRBE is implemented and FEAT_RME is implemented:

Non-secure Trace Buffer Extended. Together with MDCR_EL3.NSTB, controls the trace buffer owning Security state and accesses to trace buffer System registers from EL2 and EL1.

For a description of the values derived by evaluating NSTB and NSTBE together, see MDCR_EL3.NSTB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSTB, bits [25:24]

When FEAT_TRBE is implemented and FEAT_RME is implemented:

Non-secure Trace Buffer. Together with MDCR_EL3.NSTBE, controls the trace buffer owning Security state and accesses to trace buffer System registers from EL2 and EL1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

NSTBE	NSTB	Meaning
0b0	0b00	Trace buffer owning Security state is Secure state. When TraceBufferEnabled()==TRUE, tracing is prohibited in Realm and Non-secure states. Accesses to trace buffer System registers at EL2 and EL1 are trapped to EL3, unless the access generates a higher priority exception. When Secure state is not implemented, this encoding is reserved.
0b0	0b01	Trace buffer owning Security state is Secure state. When TraceBufferEnabled()==TRUE, tracing is prohibited in Realm and Non-secure states. Accesses to trace buffer System registers at Realm and Non-secure EL2, and Realm and Non-secure EL1, are trapped to EL3, unless the access generates a higher priority exception. When Secure state is not implemented, this encoding is reserved.
0b0	0b10	Trace buffer owning Security state is Non-secure state. When TraceBufferEnabled()==TRUE, tracing is prohibited in Secure and Realm states. Accesses to trace buffer System registers at EL2 and EL1 are trapped to EL3, unless the access generates a higher priority exception.
0b0	0b11	Trace buffer owning Security state is Non-secure state. When TraceBufferEnabled()==TRUE, tracing is prohibited in Secure and Realm states. Accesses to trace buffer System registers at Secure and Realm EL2, and Secure and Realm EL1, are trapped to EL3, unless the access generates a higher priority exception.
0b1	0b10	Trace buffer owning Security state is Realm state. When TraceBufferEnabled()==TRUE, tracing is prohibited in Secure and Non-secure states. Accesses to trace buffer System registers at EL2 and EL1 are trapped to EL3, unless the access generates a highe priority exception.
0b1	0b11	Trace buffer owning Security state is Realm state. When TraceBufferEnabled()==TRUE, tracing is prohibited in Secure and Non-secure states. Accesses to trace buffer System registers at Secure and Non-secure EL2, and Secure and Non-secure EL1, are trapped to EL3, unless the access generates a higher priority exception.

All other combinations of MDCR_EL3.NSTBE and MDCR_EL3.NSTB are reserved.

In AArch64 state, the instructions affected by this control are:

- `MRS` and `MSR` accesses to TRBBASER_EL1, TRBLIMITR_EL1, TRBMAR_EL1, TRBPTR_EL1, TRBSR_EL1, and TRBTRG_EL1.
- If FEAT_TRBE_MPAM is implemented, `MRS` and `MSR` accesses to TRBMPAM_EL1.
- If FEAT_TRBE_EXC is implemented, `MRS` and `MSR` accesses to TRBSR_EL2 and TRBSR_EL12.

Unless the instruction generates a higher priority exception, trapped instructions generate an exception to EL3, reported using EC syndrome value `0x18`.

When `TraceBufferEnabled()`==FALSE, these controls have no effect on whether tracing is prohibited.

If the Trace Buffer Unit is enabled and using Self-hosted mode, and MDCR_EL3.{NSTB, NSTBE} selects a reserved value, then the behavior is CONSTRAINED UNPREDICTABLE, and the Trace Buffer Unit does one of:

- Behaves as if the Trace Buffer Unit is disabled.
- Selects an implemented Security state as the owning Security state.
- When trace data is received from the trace unit, it is not written to memory and the Trace Buffer Unit generates a trace buffer management event, as follows:
  - TRBSR_ELx.IRQ is set to 1.
  - If TRBSR_ELx.S is 0, then all of the following occur:
    - TRBSR_ELx.S is set to 1, Collection is stopped.
    - TRBSR_ELx.EC is set to `0x00`, Other buffer management event.
    - TRBSR_ELx.BSC is set to `0b000000`, Access not allowed.
  - The other fields in TRBSR_ELx are unchanged.

If the Trace Buffer Unit is enabled and using Self-hosted mode, and MDCR_EL3.{NSTB, NSTBE} selects a reserved value, then it is CONSTRAINED UNPREDICTABLE whether or not accesses to trace buffer System registers at EL2 and EL1 generate Trap exceptions to EL3.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 1, then the Effective value of this field is `0b11`.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0, then the Effective value of this field is `0b01`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_TRBE is implemented:

Non-secure Trace Buffer. Controls the trace buffer owning Security state and accesses to trace buffer System registers from EL2 and EL1.

NSTB	Meaning
`0b00`	Trace buffer owning Security state is Secure state. When `TraceBufferEnabled()`==TRUE, tracing is prohibited in Non-secure state. Accesses to trace buffer System registers at EL2 and EL1 are trapped to EL3, unless the access generates a higher priority exception.
`0b01`	Trace buffer owning Security state is Secure state. When `TraceBufferEnabled()`==TRUE, tracing is prohibited in Non-secure state. Accesses to trace buffer System registers at EL2 and EL1 in Non-secure state are trapped to EL3, unless the access generates a higher priority exception.
`0b10`	Trace buffer owning Security state is Non-secure state. When `TraceBufferEnabled()`==TRUE, tracing is prohibited in Secure state. Accesses to trace buffer System registers at EL2 and EL1 are trapped to EL3, unless the access generates a higher priority exception.
`0b11`	Trace buffer owning Security state is Non-secure state. When `TraceBufferEnabled()`==TRUE, tracing is prohibited in Secure state. Accesses to trace buffer System registers at EL2 and EL1 in Secure state are trapped to EL3, unless the access generates a higher priority exception.

In AArch64 state, the instructions affected by this control are:

- `MRS` and `MSR` accesses to TRBBASER_EL1, TRBLIMITR_EL1, TRBMAR_EL1, TRBPTR_EL1, TRBSR_EL1, and TRBTRG_EL1.
- If FEAT_TRBE_MPAM is implemented, `MRS` and `MSR` accesses to TRBMPAM_EL1.
- If FEAT_TRBE_EXC is implemented, `MRS` and `MSR` accesses to TRBSR_EL2 and TRBSR_EL12.

Unless the instruction generates a higher priority exception, trapped instructions generate an exception to EL3, reported using EC syndrome value `0x18`.

When `TraceBufferEnabled()`==FALSE, this control has no effect on whether tracing is prohibited.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 1, then the Effective value of this field is `0b11`.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0, then the Effective value of this field is `0b01`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SCCD, bit [23]

When FEAT_PMUv3p5 is implemented:

Secure Cycle Counter Disable. Prohibits PMCCNTR_EL0 from counting in Secure state and EL3.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SCCD	Meaning
0b0	Cycle counting by PMCCNTR_EL0 is not affected by this mechanism.
0b1	Cycle counting by PMCCNTR_EL0 is prohibited in Secure state and EL3.

This field does not affect the CPU_CYCLES event or any other event that counts cycles.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

ETAD, bit [22]

When FEAT_RME is implemented, FEAT_TRC_EXT is implemented, and FEAT_TRBE is implemented:

External Trace Access Disable. Together with MDCR_EL3.ETADE, controls access to trace unit registers by an external debugger.

ETADE	ETAD	Meaning
0b0	0b0	No accesses from an external debugger to trace unit registers are prohibited by this control.
0b0	0b1	Realm and Non-secure accesses from an external debugger to trace unit registers are prohibited. Other accesses from an external debugger to trace unit registers are not affected by this control.
0b1	0b0	Secure and Non-secure accesses from an external debugger to trace unit registers are prohibited. Other accesses from an external debugger to trace unit registers are not affected by this control.
0b1	0b1	Secure, Non-secure, and Realm accesses from an external debugger to trace unit registers are prohibited. Other accesses from an external debugger to trace unit registers are not affected by this control.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

When FEAT_TRC_EXT is implemented and FEAT_TRBE is implemented:

External Trace Access Disable. Controls Non-secure access to trace unit registers by an external debugger.

ETAD	Meaning
0b0	No accesses from an external debugger to the trace unit registers are prohibited by this control.
0b1	Non-secure accesses from an external debugger to some trace unit registers are prohibited. See individual registers for the effect of this field.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EPMAD, bit [21]

When FEAT_RME is implemented and FEAT_PMUv3_EXT is implemented:

External Performance Monitors Access Disable. Together with MDCR_EL3.EPMADE, controls access to Performance Monitor registers by an external debugger.

External accesses of the following Performance Monitor registers are affected by this control:

- PMCCFILTR_EL0, PMCCNTR_EL0, PMCFGR, PMCNTENCLR_EL0, PMCNTENSET_EL0, PMCR_EL0, PMEVCNTR<n>_EL0, PMEVTYPER<n>_EL0, PMINTENCLR_EL1, PMINTENSET_EL1, PMOVSCLR_EL0, and PMOVSSET_EL0.
- If PMEVFILT2R<n> is implemented, PMEVFILT2R<n>.
- If implemented, PMIIDR.
- If PMSWINC_EL0 is implemented in the external view, PMSWINC_EL0.
- If FEAT_PMUv3p4 is implemented, PMMIR.
- If FEAT_PMUv3_EXT64 is implemented, PMCNTEN, PMINTEN, and PMOVS.
- If FEAT_PMUv3p9 is implemented, PMZR_EL0.
- If FEAT_PMUv3_ICNTR is implemented, PMCGCR0, PMICFILTR_EL0, and PMICNTR_EL0.
- If FEAT_PCSRv8p9 is implemented, PMPCSCTL.

EPMADE	EPMAD	Meaning
0b0	0b0	No accesses from an external debugger to affected Performance Monitor registers are prohibited by this control.
0b0	0b1	Realm and Non-secure accesses from an external debugger to affected Performance Monitor registers are prohibited. Other accesses from an external debugger to affected Performance Monitor registers are not affected by this control.
0b1	0b0	Secure and Non-secure accesses from an external debugger to affected Performance Monitor registers are prohibited. Other accesses from an external debugger to affected Performance Monitor registers are not affected by this control.
0b1	0b1	Secure, Non-secure, and Realm accesses from an external debugger to affected Performance Monitor registers are prohibited. Other accesses from an external debugger to affected Performance Monitor registers are not affected by this control.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

When FEAT_Debugv8p4 is implemented and FEAT_PMUv3_EXT is implemented:

External Performance Monitors Access Disable. Controls Non-secure access to Performance Monitor registers by an external debugger.

External accesses of the following Performance Monitor registers are affected by this control:

- PMCCFILTR_EL0, PMCCNTR_EL0, PMCFGR, PMCNTENCLR_EL0, PMCNTENSET_EL0, PMCR_EL0, PMEVCNTR<n>_EL0, PMEVTYPER<n>_EL0, PMINTENCLR_EL1, PMINTENSET_EL1, PMOVSCLR_EL0, and PMOVSSET_EL0.
- If PMEVFILT2R<n> is implemented, PMEVFILT2R<n>.
- If implemented, PMIIDR.
- If PMSWINC_EL0 is implemented in the external view, PMSWINC_EL0.
- If FEAT_PMUv3p4 is implemented, PMMIR.
- If FEAT_PMUv3_EXT64 is implemented, PMCNTEN, PMINTEN, and PMOVS.
- If FEAT_PMUv3p9 is implemented, PMZR_EL0.
- If FEAT_PMUv3_ICNTR is implemented, PMCGCR0, PMICFILTR_EL0, and PMICNTR_EL0.
- If FEAT_PCSRv8p9 is implemented, PMPCSCTL.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EPMAD	Meaning
0b0	No accesses from an external debugger to the Performance Monitor registers are prohibited by this control.
0b1	Non-secure accesses from an external debugger to the affected Performance Monitor registers are prohibited.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

When FEAT_PMUv3_EXT is implemented:

External Performance Monitors Access Disable. Controls access to Performance Monitor registers by an external debugger.

External accesses of the following Performance Monitor registers are affected by this control:

- PMCCFILTR_EL0, PMCCNTR_EL0, PMCFGR, PMCNTENCLR_EL0, PMCNTENSET_EL0, PMCR_EL0, PMEVCNTR<n>_EL0, PMEVTYPER<n>_EL0, PMINTENCLR_EL1, PMINTENSET_EL1, PMOVSCLR_EL0, and PMOVSSET_EL0.
- If PMEVFILT2R<n> is implemented, PMEVFILT2R<n>.
- If implemented, PMIIDR.
- If PMSWINC_EL0 is implemented in the external view, PMSWINC_EL0.
- If FEAT_PMUv3p4 is implemented, PMMIR.

EPMAD	Meaning
0b0	No accesses from an external debugger to the Performance Monitor registers are prohibited by this control.
0b1	If the IMPLEMENTATION DEFINED authentication interface function ExternalSecureInvasiveDebugEnabled() returns FALSE, then accesses from an external debugger to the affected Performance Monitor registers are prohibited.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

EDAD, bit [20]

When FEAT_RME is implemented:

External Debug Access Disable. Together with MDCR_EL3.EDADE, controls access to breakpoint registers, watchpoint registers, and OSLAR_EL1 by an external debugger.

External accesses of the following debug registers are affected by this control:

- DBGBCR<n>_EL1, DBGBVR<n>_EL1, DBGWCR<n>_EL1, DBGWVR<n>_EL1, and OSLAR_EL1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EDADE	EDAD	Meaning
0b0	0b0	No accesses from an external debugger to affected debug registers are prohibited by this control.
0b0	0b1	Realm and Non-secure accesses from an external debugger to affected debug registers are prohibited. Other accesses from an external debugger to affected debug registers are not affected by this control.
0b1	0b0	Secure and Non-secure accesses from an external debugger to affected debug registers are prohibited. Other accesses from an external debugger to affected debug registers are not affected by this control.
0b1	0b1	Secure, Non-secure, and Realm accesses from an external debugger to affected debug registers are prohibited. Other accesses from an external debugger to affected debug registers are not affected by this control.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

When FEAT_Debugv8p4 is implemented:

External Debug Access Disable. Controls Non-secure access to breakpoint registers, watchpoint registers, and OSLAR_EL1 by an external debugger.

External accesses of the following debug registers are affected by this control:

- DBGBCR<n>_EL1, DBGBVR<n>_EL1, DBGWCR<n>_EL1, DBGWVR<n>_EL1, and OSLAR_EL1.

EDAD	Meaning
0b0	No accesses from an external debugger to the debug registers are prohibited by this control.
0b1	Non-secure accesses from an external debugger to the affected debug registers are prohibited.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

When FEAT_Debugv8p2 is implemented:

External Debug Access Disable. Controls access to breakpoint registers, watchpoint registers, and OSLAR_EL1 by an external debugger.

External accesses of the following debug registers are affected by this control:

- DBGBCR<n>_EL1, DBGBVR<n>_EL1, DBGWCR<n>_EL1, DBGWVR<n>_EL1, and OSLAR_EL1.

EDAD	Meaning
0b0	No accesses from an external debugger to the debug registers are prohibited by this control.
0b1	If the IMPLEMENTATION DEFINED authentication interface function `ExternalSecureInvasiveDebugEnabled()` returns FALSE, then accesses from an external debugger to the affected debug registers are prohibited.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Otherwise:

External Debug Access Disable. Controls access to breakpoint registers, watchpoint registers, and optionally OSLAR_EL1 by an external debugger.

External accesses of the following debug registers are affected by this control:

- DBGBCR<n>_EL1, DBGBVR<n>_EL1, DBGWCR<n>_EL1, and DBGWVR<n>_EL1.
- It is IMPLEMENTATION DEFINED whether this control affects OSLAR_EL1.

EDAD	Meaning
`0b0`	No accesses from an external debugger to the debug registers are prohibited by this control.
`0b1`	If the IMPLEMENTATION DEFINED authentication interface function `ExternalSecureInvasiveDebugEnabled()` returns FALSE, then accesses from an external debugger to the affected debug registers are prohibited.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

TTRF, bit [19]

When FEAT_TRF is implemented:

Trap Trace Filter controls. Traps use of the Trace Filter control registers at EL2 and EL1 to EL3.

The Trace Filter registers trapped by this control are:

- TRFCR_EL2, TRFCR_EL12, TRFCR_EL1, reported using EC syndrome value `0x18`.
- HTRFCR and TRFCR, reported using EC syndrome value `0x03`.

TTRF	Meaning
`0b0`	Accesses to Trace Filter registers at EL2 and EL1 are not affected by this field.
`0b1`	Accesses to Trace Filter registers at EL2 and EL1 generate a Trap exception to EL3, unless the access generates a higher priority exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

STE, bit [18]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_TRF is implemented and Secure state is implemented:

Secure Trace enable. Enables tracing in Secure state.

STE	Meaning
0b0	Trace prohibited in Secure state unless overridden by the IMPLEMENTATION DEFINED authentication interface.
0b1	Trace in Secure state is not affected by this field.

This field also controls the level of authentication required by an external debugger to enable external tracing. See 'Register controls to enable self-hosted trace'.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0, the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

SPME, bit [17]

When FEAT_PMUv3 is implemented and FEAT_PMUv3p7 is implemented:

Secure Performance Monitors Enable. Controls PMU operation in Secure state and at EL3 when MDCR_EL3.MPMX is 0.

SPME	Meaning
0b0	Affected counters are prohibited from counting in Secure state and at EL3. If PMCR_EL0.DP is 1, PMCCNTR_EL0 is disabled in Secure state and at EL3. Otherwise, PMCCNTR_EL0 is not affected by this mechanism.
0b1	Counters are not affected by this mechanism.

When MDCR_EL3.MPMX is 0, the counters affected by this field are:

- Event counters in the first and second ranges. For more information about event counter ranges, see MDCR_EL2.HPMN.
- If FEAT_PMUv3_ICNTR is implemented, the instruction counter, PMICNTR_EL0.
- If PMCR_EL0.DP is 1, the cycle counter, PMCCNTR_EL0.

When PMCR_EL0.DP is 0, PMCCNTR_EL0 is not affected by this field.

When MDCR_EL3.MPMX is 1, this field controls which event counters are affected by MDCR_EL3.MPMX at EL3. See MDCR_EL3.MPMX for more information.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

When FEAT_PMUv3 is implemented and FEAT_Debugv8p2 is implemented:

Secure Performance Monitors Enable. Controls PMU operation in Secure state.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SPME	Meaning
0b0	Event counting is prohibited in Secure state. If PMCR_EL0.DP is 1, PMCCNTR_EL0 is disabled in Secure state. Otherwise, PMCCNTR_EL0 is not affected by this mechanism.
0b1	Event counting and PMCCNTR_EL0 are not affected by this mechanism.

The counters affected by this field are:

- All event counters.
- If PMCR_EL0.DP is 1, the cycle counter, PMCCNTR_EL0.

When PMCR_EL0.DP is 0, PMCCNTR_EL0 is not affected by this field.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

When FEAT_PMUv3 is implemented:

Secure Performance Monitors Enable. Controls PMU operation in Secure state.

SPME	Meaning
0b0	If `ExternalSecureNoninvasiveDebugEnabled()` is FALSE, then all the following apply: • Event counting is prohibited in Secure state. • If PMCR_EL0.DP is 1, PMCCNTR_EL0 is disabled in Secure state. Otherwise, PMCCNTR_EL0 is not affected by this mechanism.
0b1	Event counting and PMCCNTR_EL0 are not affected by this mechanism.

If `ExternalSecureNoninvasiveDebugEnabled()` is TRUE, then the event counters and PMCCNTR_EL0 are not affected by this field.

Otherwise, the counters affected by this field are:

- All event counters.
- If PMCR_EL0.DP is 1, the cycle counter, PMCCNTR_EL0.

When PMCR_EL0.DP is 0, PMCCNTR_EL0 is not affected by this field.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Otherwise:

Reserved, RES0.

SDD, bit [16]

When Secure state is implemented:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

AArch64 Secure Self-hosted invasive debug disable. Disables Software debug exceptions in Secure state, other than Breakpoint Instruction exceptions.

SDD	Meaning
0b0	Debug exceptions in Secure state are not affected by this field.
0b1	Debug exceptions, other than Breakpoint Instruction exceptions, are disabled from all Exception levels in Secure state.

The SDD bit is ignored unless both of the following are true:

- The PE is in Secure state.
- The Effective value of SCR_EL3.RW is 0.

If Secure EL2 is implemented and enabled, and Secure EL1 is using AArch32, then:

- If debug exceptions from Secure EL1 are enabled, debug exceptions from Secure EL0 are also enabled.
- Otherwise, debug exceptions from Secure EL0 are enabled only if the value of SDER32_EL3.SUIDEN is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SPD32, bits [15:14]

When FEAT_AA32EL1 is implemented:

AArch32 Secure self-hosted privileged debug. Enables or disables debug exceptions from Secure EL1 using AArch32, other than Breakpoint Instruction exceptions.

SPD32	Meaning
0b00	Legacy mode. Debug exceptions from Secure EL1 are enabled by the IMPLEMENTATION DEFINED authentication interface.
0b10	Secure privileged debug disabled. Debug exceptions from Secure EL1 are disabled.
0b11	Secure privileged debug enabled. Debug exceptions from Secure EL1 are enabled.

Other values are reserved, and have the CONSTRAINED UNPREDICTABLE behavior that they must have the same behavior as 0b00. Software must not rely on this property as the behavior of reserved values might change in a future revision of the architecture.

This field has no effect on Breakpoint Instruction exceptions. These are always enabled.

This field is ignored unless both of the following are true:

- The PE is in Secure state.
- The Effective value of SCR_EL3.RW is 0.

If Secure EL1 is using AArch32, then:

- If debug exceptions from Secure EL1 are enabled, then debug exceptions from Secure EL0 are also enabled.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- Otherwise, debug exceptions from Secure EL0 are enabled only if the value of SDER32_EL3.SUIDEN is 1.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0, then the Effective value of this field is `0b11`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSPB, bits [13:12]

When FEAT_SPE is implemented and FEAT_RME is implemented:

Non-secure Profiling Buffer. Together with MDCR_EL3.NSPBE, controls the Profiling Buffer owning Security state and accesses to Statistical Profiling and Profiling Buffer System registers from EL2 and EL1.

NSPBE	NSPB	Meaning
`0b0`	`0b00`	Profiling Buffer owning Security state is Secure state. Profiling is disabled in Realm and Non-secure states. Accesses to Statistical Profiling and Profiling Buffer System registers at EL2 and EL1 are trapped to EL3, unless the access generates a higher priority exception. When Secure state is not implemented, this encoding is reserved.
`0b0`	`0b01`	Profiling Buffer owning Security state is Secure state. Profiling is disabled in Realm and Non-secure states. Accesses to Statistical Profiling and Profiling Buffer System registers at Realm and Non-secure EL2, and Realm and Non-secure EL1, are trapped to EL3, unless the access generates a higher priority exception. When Secure state is not implemented, this encoding is reserved.
`0b0`	`0b10`	Profiling Buffer owning Security state is Non-secure state. Profiling is disabled in Secure and Realm states. Accesses to Statistical Profiling and Profiling Buffer System registers at EL2 and EL1 are trapped to EL3, unless the access generates a higher priority exception.
`0b0`	`0b11`	Profiling Buffer owning Security state is Non-secure state. Profiling is disabled in Secure and Realm states. Accesses to Statistical Profiling and Profiling Buffer System registers at Secure and Realm EL2, and Secure and Realm EL1, are trapped to EL3, unless the access generates a higher priority exception.
`0b1`	`0b10`	Profiling Buffer owning Security state is Realm state. Profiling is disabled in Secure and Non-secure states. Accesses to Statistical Profiling and Profiling Buffer System registers at EL2 and EL1 are trapped to EL3, unless the access generates a highe priority exception.
`0b1`	`0b11`	Profiling Buffer owning Security state is Realm state. Profiling is disabled in Secure and Non-secure states. Accesses to Statistical Profiling and Profiling Buffer System registers at Secure and Non-secure EL2, and Secure and Non-secure EL1, are trapped to EL3, unless the access generates a higher priority exception.

All other combinations of MDCR_EL3.NSPBE and MDCR_EL3.NSPB are reserved.

In AArch64 state, the instructions affected by this control are:

- `MRS` and `MSR` accesses to PMBLIMITR_EL1, PMBPTR_EL1, PMBSR_EL1, PMSCR_EL1, PMSCR_EL2, PMSCR_EL12, PMSEVFR_EL1, PMSFCR_EL1, PMSICR_EL1, PMSIRR_EL1, and PMSLATFR_EL1.
- `MRS` accesses to PMSIDR_EL1.
- If FEAT_SPE_FnE is implemented, `MRS` and `MSR` accesses to PMSNEVFR_EL1.
- If FEAT_SPE_FDS is implemented, `MRS` and `MSR` accesses to PMSDSFR_EL1.
- If FEAT_SPE_EXC is implemented, `MRS` and `MSR` accesses to PMBSR_EL2 and PMBSR_EL12.
- If FEAT_SPE_nVM is implemented, `MRS` and `MSR` accesses to PMBMAR_EL1.

Unless the instruction generates a higher priority exception, trapped instructions generate an exception to EL3, reported using EC syndrome value `0x18`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

If profiling is enabled and MDCR_EL3.{NSPB, NSPBE} selects a reserved value, then the behavior is CONSTRAINED UNPREDICTABLE, and the Statistical Profiling Unit does one of:

- Behaves as if profiling is disabled.
- Selects an implemented Security state as the owning Security state.
- When profiling data is generated, it is not written to memory and the Statistical Profiling Unit generates a Profiling Buffer management event, as follows:
  - If PMBSR_ELx.S is 0, then all of the following occur:
    - PMBSR_ELx.S is set to 1.
    - PMBSR_ELx.DL is set to 1.
    - PMBSR_ELx.EC is set to `0b000000`, Other buffer management event.
    - PMBSR_ELx.BSC is set to `0b000000`, Access not allowed.
  - The other fields in PMBSR_ELx are unchanged.

If profiling is enabled and MDCR_EL3.{NSPB, NSPBE} selects a reserved value, then it is CONSTRAINED UNPREDICTABLE whether or not accesses to the Statistical Profiling and Profiling Buffer System registers at EL2 and EL1 generate Trap exceptions to EL3.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 1, then the Effective value of this field is `0b11`.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0, then the Effective value of this field is `0b01`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_SPE is implemented:

Non-secure Profiling Buffer. Controls the Profiling Buffer owning Security state and accesses to Statistical Profiling and Profiling Buffer System registers from EL2 and EL1.

NSPB	Meaning
`0b00`	Profiling Buffer owning Security state is Secure state. Profiling is disabled in Non-secure state. Accesses to Statistical Profiling and Profiling Buffer System registers at EL2 and EL1 are trapped to EL3, unless the access generates a higher priority exception.
`0b01`	Profiling Buffer owning Security state is Secure state. Profiling is disabled in Non-secure state. Accesses to Statistical Profiling and Profiling Buffer System registers at EL2 and EL1 in Non-secure state are trapped to EL3, unless the access generates a higher priority exception.
`0b10`	Profiling Buffer owning Security state is Non-secure state. Profiling is disabled in Secure state. Accesses to Statistical Profiling and Profiling Buffer System registers at EL2 and EL1 are trapped to EL3, unless the access generates a higher priority exception.
`0b11`	Profiling Buffer owning Security state is Non-secure state. Profiling is disabled in Secure state. Accesses to Statistical Profiling and Profiling Buffer System registers at EL2 and EL1 in Secure state are trapped to EL3, unless the access generates a higher priority exception.

In AArch64 state, the instructions affected by this control are:

- `MRS` and `MSR` accesses to PMBLIMITR_EL1, PMBPTR_EL1, PMBSR_EL1, PMSCR_EL1, PMSCR_EL2, PMSCR_EL12, PMSEVFR_EL1, PMSFCR_EL1, PMSICR_EL1, PMSIRR_EL1, and PMSLATFR_EL1.
- `MRS` accesses to PMSIDR_EL1.
- If FEAT_SPE_FnE is implemented, `MRS` and `MSR` accesses to PMSNEVFR_EL1.
- If FEAT_SPE_FDS is implemented, `MRS` and `MSR` accesses to PMSDSFR_EL1.
- If FEAT_SPE_EXC is implemented, `MRS` and `MSR` accesses to PMBSR_EL2 and PMBSR_EL12.
- If FEAT_SPE_nVM is implemented, `MRS` and `MSR` accesses to PMBMAR_EL1.

Unless the instruction generates a higher priority exception, trapped instructions generate an exception to EL3, reported using EC syndrome value `0x18`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 1, then the Effective value of this field is `0b11`.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0, then the Effective value of this field is `0b01`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSPBE, bit [11]

When FEAT_SPE is implemented and FEAT_RME is implemented:

Non-secure Profiling Buffer Extended. Together with MDCR_EL3.NSPB, controls the Profiling Buffer owning Security state and accesses to Statistical Profiling and Profiling Buffer System registers from EL2 and EL1.

For a description of the values derived by evaluating NSPB and NSPBE together, see MDCR_EL3.NSPB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TDOSA, bit [10]

When FEAT_DoubleLock is implemented:

Trap debug OS-related register access. Traps EL2 and EL1 System register accesses to the powerdown debug registers to EL3.

Accesses to the registers are trapped as follows:

- Accesses from AArch64 state, OSLAR_EL1, OSLSR_EL1, OSDLR_EL1, DBGPRCR_EL1, and any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this field, are trapped to EL3 and reported using EC syndrome value `0x18`.
- Accesses using MCR or MRC to DBGOSLAR, DBGOSLSR, DBGOSDLR, and DBGPRCR, are trapped to EL3 and reported using EC syndrome value `0x05`.
- Accesses to any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this field.

TDOSA	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	EL2 and EL1 System register accesses to the powerdown debug registers are trapped to EL3, unless it is trapped by HDCR.TDOSA or MDCR_EL2.TDOSA.

Note

The powerdown debug registers are not accessible at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Trap debug OS-related register access. Traps EL2 and EL1 System register accesses to the powerdown debug registers to EL3.

The following registers are affected by this trap:

- AArch64: OSLAR_EL1, OSLSR_EL1, and DBGPRCR_EL1.
- AArch32: DBGOSLAR, DBGOSLSR, and DBGPRCR.
- AArch64 and AArch32: Any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this field.
- It is IMPLEMENTATION DEFINED whether accesses to OSDLR_EL1 and DBGOSDLR are trapped.

TDOSA	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL2 and EL1 System register accesses to the powerdown debug registers are trapped to EL3, unless it is trapped by HDCR.TDOSA or MDCR_EL2.TDOSA.

Note

The powerdown debug registers are not accessible at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TDA, bit [9]

Trap accesses of debug System registers. Enables a trap to EL3 on accesses of debug System registers.

TDA	Meaning
0b0	Accesses of the specified debug System registers are not trapped by this mechanism.
0b1	Accesses of the specified debug System registers at EL2, EL1, and EL0 are trapped to EL3, unless the instruction generates a higher priority exception.

In AArch64 state, the instructions affected by this control are:

- MRS and MSR accesses to DBGAUTHSTATUS_EL1, DBGBCR<n>_EL1, DBGBVR<n>_EL1, DBGCLAIMCLR_EL1, DBGCLAIMSET_EL1, DBGVCR32_EL2, DBGWCR<n>_EL1, DBGWVR<n>_EL1, MDCCINT_EL1, MDCCSR_EL0, MDCR_EL2, MDRAR_EL1, MDSCR_EL1, OSDTRRX_EL1, OSDTRTX_EL1, and OSECCR_EL1.
- If FEAT_Debugv8p9 is implemented, MRS and MSR accesses to MDSELR_EL1.
- If FEAT_STEP2 is implemented, MRS and MSR accesses to MDSTEPOP_EL1.
- In Non-debug state, MRS accesses to DBGDTRRX_EL0 and DBGDTR_EL0 and MSR accesses to DBGDTRTX_EL0 and DBGDTR_EL0.

In AArch32 state, the instructions affected by this control are:

- MRC and MCR accesses to DBGAUTHSTATUS, DBGBCR<n>, DBGBVR<n>, DBGBXVR<n>, DBGCLAIMCLR, DBGCLAIMSET, DBGDCCINT, DBGDEVID, DBGDEVID1, DBGDEVID2, DBGDIDR, DBGDRAR, DBGDSAR, DBGDSCRext, DBGDSCRint, DBGDTRRXext, DBGDTRTXext, DBGOSECCR, DBGVCR, DBGWCR<n>, DBGWFAR, DBGWVR<n>, HDCR, and SDER.
- MRRC accesses to DBGDRAR and DBGDSAR.
- STC accesses to DBGDTRRXint and LDC accesses to DBGDTRTXint.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- In Non-debug state, MRC accesses to DBGDTRRXint and MCR accesses to DBGDTRTXint.

Trapped AArch64 instructions are reported using EC syndrome value 0x18.

Trapped AArch32 instructions are reported using EC syndrome value 0x03 for MRC and MCR accesses with coproc == 0b1111, 0x05 for MCR and MCR accesses with coproc == 0b1110, 0x06 for LDC and STC accesses, and 0x0C for MRRC accesses.

The following instructions are not trapped in Debug state:

- AArch64 MRS accesses to DBGDTRRX_EL0 and DBGDTR_EL0 and MSR accesses to DBGDTRTX_EL0 and DBGDTR_EL0.
- AArch32 MRC accesses to DBGDTRRXint and MCR accesses to DBGDTRTXint.

If 16 or fewer breakpoints and 16 or fewer watchpoints are implemented, and MDSELR_EL1 is implemented as RAZ/WI, then it is IMPLEMENTATION DEFINED whether AArch64 accesses to MDSELR_EL1 are trapped to EL3 when MDCR_EL3.TDA is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [8]

Reserved, RES0.

EnPM2, bit [7]

When FEAT_PMUv3p9 is implemented, or FEAT_SPMU is implemented, or FEAT_EBEP is implemented, or FEAT_PMUv3_SS is implemented, or FEAT_SPMU2 is implemented:

Enable access to PMU registers. When disabled, accesses to PMU registers generate a trap to EL3.

EnPM2	Meaning
0b0	Accesses of the specified PMU registers at EL2, EL1, and EL0 are trapped to EL3, unless the instruction generates a higher priority exception. If FEAT_PMUv3_ICNTR is implemented, then: • PMCNTENCLR_EL0.F0, PMCNTENSET_EL0.F0, PMOVSCLR_EL0.F0, PMOVSSET_EL0.F0, and PMZR_EL0.F0 read-as-zero and ignore writes at EL2, EL1, and EL0. • PMINTENCLR_EL1.F0 and PMINTENSET_EL1.F0 read-as-zero and ignore writes at EL2 and EL1.
0b1	Accesses of the specified PMU registers are not trapped by this mechanism.

In AArch64 state, the instructions affected by this control are:

- If FEAT_EBEP is implemented or FEAT_PMUv3_SS is implemented, MRS and MSR accesses to PMECR_EL1.
- If FEAT_PMUv3_ICNTR is implemented, MRS and MSR accesses to PMICFILTR_EL0 and PMICNTR_EL0.
- If FEAT_PMUv3p9 is implemented, MRS and MSR accesses to PMUACR_EL1.
- If FEAT_SEBEP is implemented, MRS and MSR accesses to PMIAR_EL1.
- If FEAT_SPMU is implemented, MRS and MSR accesses to SPMACCESSR_EL1, SPMACCESSR_EL2, SPMACCESSR_EL12, SPMCFGR_EL1, SPMCGCR<n>_EL1, SPMCNTENCLR_EL0, SPMCNTENSET_EL0, SPMCR_EL0, SPMDEVAFF_EL1, SPMDEVARCH_EL1, SPMEVCNTR<n>_EL0, SPMEVFILT2R<n>_EL0, SPMEVFILTR<n>_EL0, SPMEVTYPER<n>_EL0, SPMIIDR_EL1, SPMINTENCLR_EL1, SPMINTENSET_EL1, SPMOVSCLR_EL0, SPMOVSSET_EL0, SPMSCR_EL1, and SPMSELR_EL0.
- If FEAT_SPMU2 is implemented, MSR writes to SPMZR_EL0.

Trapped instructions are reported using EC syndrome value 0x18.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TPM, bit [6]

When FEAT_PMUv3 is implemented:

Trap accesses of PMU registers. Enables a trap to EL3 on accesses of PMU registers.

TPM	Meaning
0b0	Accesses of the specified PMU registers are not trapped by this mechanism.
0b1	Accesses of the specified PMU registers at EL2, EL1, and EL0 are trapped to EL3, unless the instruction generates a higher priority exception.

In AArch64 state, the instructions affected by this control are:

- MRS and MSR accesses to PMCCFILTR_EL0, PMCCNTR_EL0, PMCNTENCLR_EL0, PMCNTENSET_EL0, PMCR_EL0, PMEVCNTR<n>_EL0, PMEVTYPER<n>_EL0, PMINTENCLR_EL1, PMINTENSET_EL1, PMOVSCLR_EL0, PMOVSSET_EL0, PMSELR_EL0, PMSWINC_EL0, PMUSERENR_EL0, PMXEVCNTR_EL0, and PMXEVTYPER_EL0.
- MRS accesses to PMCEID0_EL0 and PMCEID1_EL0.
- If FEAT_PMUv3p4 is implemented, MRS accesses to PMMIR_EL1.
- If FEAT_PMUv3p9 is implemented, MSR accesses to PMZR_EL0.
- If FEAT_PMUv3_ICNTR is implemented, MRS and MSR accesses to PMICFILTR_EL0 and PMICNTR_EL0.
- If FEAT_EBEP is implemented or FEAT_PMUv3_SS is implemented, MRS and MSR accesses to PMECR_EL1.
- If FEAT_SEBEP is implemented, MRS and MSR accesses to PMIAR_EL1.

In AArch32 state, the instructions affected by this control are:

- MRC and MCR accesses to PMCCFILTR, PMCCNTR, PMCEID0, PMCEID1, PMCNTENCLR, PMCNTENSET, PMCR, PMEVCNTR<n>, PMEVTYPER<n>, PMINTENCLR, PMINTENSET, PMOVSR, PMOVSSET, PMSELR, PMSWINC, PMUSERENR, PMXEVCNTR, and PMXEVTYPER.
- MRRC and MCRR accesses to PMCCNTR.
- If FEAT_PMUv3p1 is implemented, MRC accesses to PMCEID2 and PMCEID3.
- If FEAT_PMUv3p4 is implemented, MRC accesses to PMMIR.

Trapped AArch64 instructions are reported using EC syndrome value 0x18.

Trapped AArch32 instructions are reported using EC syndrome value 0x03 for MRC and MCR accesses, and 0x04 for MRRC and MCRR accesses.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [5]

Reserved, RES0.

EDADE, bit [4]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_RME is implemented:

External Debug Access Disable Extended. Together with MDCR_EL3.EDAD, controls access to breakpoint registers, watchpoint registers, and OSLAR_EL1 by an external debugger.

For a description of the values derived by evaluating MDCR_EL3.EDAD and MDCR_EL3.EDADE together, see MDCR_EL3.EDAD.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Otherwise:

Reserved, RES0.

ETADE, bit [3]

When FEAT_RME is implemented, FEAT_TRC_EXT is implemented, and FEAT_TRBE is implemented:

External Trace Access Disable Extended. Together with MDCR_EL3.ETAD, controls access to trace unit registers by an external debugger.

For a description of the values derived by evaluating MDCR_EL3.ETAD and MDCR_EL3.ETADE together, see MDCR_EL3.ETAD.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Otherwise:

Reserved, RES0.

EPMADE, bit [2]

When FEAT_RME is implemented and FEAT_PMUv3_EXT is implemented:

External Performance Monitors Access Disable Extended. Together with MDCR_EL3.EPMAD, controls access to Performance Monitor registers by an external debugger.

For a description of the values derived by evaluating MDCR_EL3.EPMAD and MDCR_EL3.EPMADE together, see MDCR_EL3.EPMAD.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Otherwise:

Reserved, RES0.

Bit [1]

Reserved, RES0.

RLTE, bit [0]

When FEAT_RME is implemented and FEAT_TRF is implemented:

Realm Trace enable. Enables tracing in Realm state.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

RLTE	Meaning
0b0	Trace prohibited in Realm state, unless overridden by the IMPLEMENTATION DEFINED authentication interface.
0b1	Trace in Realm state is not affected by this field.

This field also controls the level of authentication that is required by an external debugger to enable external tracing.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

Accessing MDCR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MDCR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0011	0b001

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MDCR_EL3;
```

```
MSR MDCR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0011	0b001

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.MDCR_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MDCR_EL3 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.3.19 MDRAR_EL1, Monitor Debug ROM Address Register

The MDRAR_EL1 characteristics are:

Purpose

Defines the base physical address of a 4KB-aligned memory-mapped debug component, usually a ROM table that locates and describes the memory-mapped debug components in the system. Use of this register is deprecated.

Configuration

AArch64 System register MDRAR_EL1 bits [63:0] are architecturally mapped to AArch32 System register DBGDRAR[63:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to MDRAR_EL1 are UNDEFINED.

Attributes

MDRAR_EL1 is a 64-bit register.

Field descriptions

Bits [63:56]

Reserved, RES0.

ROMADDR, bits [55:12]

When FEAT_D128 is implemented and MDRAR_EL1.Valid != '00':

ROMADDR, bits [43:0]

Bits [55:12] of the ROM table physical address.

Bits [11:0] of the ROM table physical address are zero.

This field has an IMPLEMENTATION DEFINED value.

For implementations with fewer than 56 physical address bits, the corresponding upper bits of this field are RES0

In an implementation that includes EL3, ROMADDR is an address in Non-secure PA space. It is IMPLEMENTATION DEFINED whether the ROM table is also accessible in Secure PA space. If FEAT_RME is

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

implemented, it is IMPLEMENTATION DEFINED whether the ROM table is also accessible in the Root or Realm PA spaces.

Arm strongly recommends that bits ROMADDR[(PAsize-1):32] are zero in any system where the implementation only supports execution in AArch32 state.

Access to this field is RO.

When FEAT_D128 is not implemented, FEAT_LPA is implemented, and MDRAR_EL1.Valid != '00':

Bits [43:40]

Reserved, RES0.

ROMADDR, bits [39:0]

Bits [51:12] of the ROM table physical address.

Bits [11:0] of the ROM table physical address are zero.

This field has an IMPLEMENTATION DEFINED value.

For implementations with fewer than 52 physical address bits, the corresponding upper bits of this field are RES0

In an implementation that includes EL3, ROMADDR is an address in Non-secure PA space. It is IMPLEMENTATION DEFINED whether the ROM table is also accessible in Secure PA space. If FEAT_RME is implemented, it is IMPLEMENTATION DEFINED whether the ROM table is also accessible in the Root or Realm PA spaces.

Arm strongly recommends that bits ROMADDR[(PAsize-1):32] are zero in any system where the implementation only supports execution in AArch32 state.

Access to this field is RO.

When FEAT_D128 is not implemented, FEAT_LPA is not implemented, and MDRAR_EL1.Valid != '00':

Bits [43:36]

Reserved, RES0.

ROMADDR, bits [35:0]

Bits [39:12] of the ROM table physical address.

Bits [11:0] of the ROM table physical address are zero.

This field has an IMPLEMENTATION DEFINED value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

For implementations with fewer than 48 physical address bits, the corresponding upper bits of this field are RES0

In an implementation that includes EL3, ROMADDR is an address in Non-secure PA space. It is IMPLEMENTATION DEFINED whether the ROM table is also accessible in Secure PA space. If FEAT_RME is implemented, it is IMPLEMENTATION DEFINED whether the ROM table is also accessible in Root or Realm PA spaces.

Arm strongly recommends that bits ROMADDR[(PAsize-1):32] are zero in any system where the implementation only supports execution in AArch32 state.

Access to this field is RO.

When MDRAR_EL1.Valid == '00':

Bits [43:0]

Reserved, UNKNOWN.

Bits [11:2]

Reserved, RES0.

Valid, bits [1:0]

This field indicates whether the ROM Table address is valid.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Valid	Meaning
`0b00`	ROM Table address is not valid. Software must ignore ROMADDR.
`0b11`	ROM Table address is valid.

Other values are reserved.

Arm recommends implementations set this field to zero.

Access to this field is RO.

Accessing MDRAR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MDRAR_EL1
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b10	0b000	0b0001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDRA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MDRAR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MDRAR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MDRAR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.3.20 MDSCR_EL1, Monitor Debug System Control Register

The MDSCR_EL1 characteristics are:

Purpose

Main control register for the debug implementation.

Configuration

AArch64 System register MDSCR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGDSCRext[31:0].

AArch64 System register MDSCR_EL1 bit [15] is architecturally mapped to AArch32 System register DBGDSCRint[15].

AArch64 System register MDSCR_EL1 bit [12] is architecturally mapped to AArch32 System register DBGDSCRint[12].

AArch64 System register MDSCR_EL1 bits [5:2] are architecturally mapped to AArch32 System register DBGDSCRint[5:2].

AArch64 System register MDSCR_EL1 bits [31:29, 27:26, 23:21, 19, 14, 6] are architecturally mapped to External register EDSCR[31:29, 27:26, 23:21, 19, 14, 6].

AArch64 System register MDSCR_EL1 bits [35, 33] are architecturally mapped to External register EDSCR2[3, 1].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to MDSCR_EL1 are UNDEFINED.

Attributes

MDSCR_EL1 is a 64-bit register.

Field descriptions

Bits [63:51]

Reserved, RES0.

EnSTEPOP, bit [50]

When FEAT_STEP2 is implemented:

Software step control bit. Enable execution from MDSTEPOP_EL1. Permitted values are:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EnSTEPOP	Meaning
0b0	Execution from MDSTEPOP_EL1 is disabled.
0b1	Execution from MDSTEPOP_EL1 is not disabled by this control.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [49:36]

Reserved, RES0.

EHBWE, bit [35]

When FEAT_Debugv8p9 is implemented:

Extended Halting Breakpoint and Watchpoint Enable. Used for save/restore of EDSCR2.EHBWE.

When OSLSR_EL1.OSLK is 0, software must treat this field as UNK/SBZP.

When OSLSR_EL1.OSLK is 1, this field holds the value of EDSCR2.EHBWE. Reads and writes of this field are indirect accesses to EDSCR2.EHBWE.

It is IMPLEMENTATION DEFINED whether this field is implemented or is RES0 when 16 or fewer breakpoints are implemented, 16 or fewer watchpoints are implemented, and MDSELR_EL1 is implemented as RAZ/WI.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- When OSLSR_EL1.OSLK == ‘0’, access to this field is RO.
- Otherwise, access to this field is RW.

Otherwise:

Reserved, RES0.

EnSPM, bit [34]

When FEAT_SPMU is implemented:

Enable access to System PMU registers. When disabled, accesses to System PMU registers generate a trap to EL1.

EnSPM	Meaning
0b0	Accesses of the specified System PMU registers at EL0 are trapped to EL1, unless the instruction generates a higher priority exception.
0b1	Accesses of the specified System PMU registers are not trapped by this mechanism.

In AArch64 state, the instructions affected by this control are: `MRS` and `MSR` accesses to SPMCNTENCLR_EL0, SPMCNTENSET_EL0, SPMCR_EL0, SPMEVCNTR<n>_EL0, SPMEVFILT2R<n>_EL0, SPMEVFILTR<n>_EL0, SPMEVTYPER<n>_EL0, SPMOVSCLR_EL0, SPMOVSSET_EL0, and SPMSELR_EL0.

Unless the instruction generates a higher priority exception:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- If EL2 is implemented and enabled in the current Security state, and HCR_EL2.TGE is 1, then trapped instructions generate an exception to EL2.
- Otherwise, trapped instructions generate an exception to EL1.

Trapped instructions are reported using EC syndrome value `0x18`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TTA, bit [33]

When FEAT_TRBE_EXT is implemented or FEAT_ETEv1p3 is implemented:

Trap Trace Accesses. Used for save/restore of EDSCR2.TTA.

When OSLSR_EL1.OSLK is 0, software must treat this field as UNK/SBZP.

When OSLSR_EL1.OSLK is 1, this field holds the value of EDSCR2.TTA. Reads and writes of this field are indirect accesses to EDSCR2.TTA.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- When OSLSR_EL1.OSLK == ‘0’, access to this field is RO.
- Otherwise, access to this field is RW.

Otherwise:

Reserved, RES0.

EMBWE, bit [32]

When FEAT_Debugv8p9 is implemented:

Extended Monitor Breakpoint and Watchpoint Enable. Enables use of additional breakpoints or watchpoints.

EMBWE	Meaning
`0b0`	Breakpoint and Watchpoint exceptions are disabled for each breakpoint <n> and watchpoint <n>, where n is greater than or equal to 16. The Effective value of MDSELR_EL1.BANK is zero at EL1.
`0b1`	Breakpoint and Watchpoint exceptions are not affected by this mechanism. The Effective value of MDSELR_EL1.BANK is not affected by this field.

It is IMPLEMENTATION DEFINED whether this field is implemented or is RES0 when 16 or fewer breakpoints are implemented, 16 or fewer watchpoints are implemented, and MDSELR_EL1 is implemented as RAZ/WI.

This field is ignored by the PE and treated as zero when any of the following are true:

- EL3 is implemented and MDCR_EL3.EBWE is 0.
- EL2 is implemented and enabled in the current Security state, and MDCR_EL2.EBWE is 0.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'0'`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TFO, bit [31]

When FEAT_TRF is implemented:

Trace Filter override. Used for save/restore of EDSCR.TFO.

When OSLSR_EL1.OSLK == 0, software must treat this bit as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this bit holds the value of EDSCR.TFO. Reads and writes of this bit are indirect accesses to EDSCR.TFO.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- When OSLSR_EL1.OSLK == ‘1’, access to this field is RW.
- When OSLSR_EL1.OSLK == ‘0’, access to this field is RO.

Otherwise:

Reserved, RES0.

RXfull, bit [30]

Used for save/restore of EDSCR.RXfull.

When OSLSR_EL1.OSLK == 0, software must treat this bit as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this bit holds the value of EDSCR.RXfull. Reads and writes of this bit are indirect accesses to EDSCR.RXfull.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- When OSLSR_EL1.OSLK == ‘1’, access to this field is RW.
- When OSLSR_EL1.OSLK == ‘0’, access to this field is RO.

TXfull, bit [29]

Used for save/restore of EDSCR.TXfull.

When OSLSR_EL1.OSLK == 0, software must treat this bit as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this bit holds the value of EDSCR.TXfull. Reads and writes of this bit are indirect accesses to EDSCR.TXfull.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- When OSLSR_EL1.OSLK == ‘1’, access to this field is RW.
- When OSLSR_EL1.OSLK == ‘0’, access to this field is RO.

Bit [28]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

RXO, bit [27]

Used for save/restore of EDSCR.RXO.

When OSLSR_EL1.OSLK == 0, software must treat this bit as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this bit holds the value of EDSCR.RXO. Reads and writes of this bit are indirect accesses to EDSCR.RXO.

When OSLSR_EL1.OSLK == 1, if bits [27,6] of the value written to MDSCR_EL1 are {1,0}, that is, the RXO bit is 1 and the ERR bit is 0, the PE sets EDSCR.{RXO,ERR} to UNKNOWN values.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- When OSLSR_EL1.OSLK == ‘1’, access to this field is RW.
- When OSLSR_EL1.OSLK == ‘0’, access to this field is RO.

TXU, bit [26]

Used for save/restore of EDSCR.TXU.

When OSLSR_EL1.OSLK == 0, software must treat this bit as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this bit holds the value of EDSCR.TXU. Reads and writes of this bit are indirect accesses to EDSCR.TXU.

When OSLSR_EL1.OSLK == 1, if bits [26,6] of the value written to MDSCR_EL1 are {1,0}, that is, the TXU bit is 1 and the ERR bit is 0, the PE sets EDSCR.{TXU,ERR} to UNKNOWN values.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- When OSLSR_EL1.OSLK == ‘1’, access to this field is RW.
- When OSLSR_EL1.OSLK == ‘0’, access to this field is RO.

Bits [25:24]

Reserved, RES0.

INTdis, bits [23:22]

Used for save/restore of EDSCR.INTdis.

When OSLSR_EL1.OSLK == 0, software must treat this bit as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this field holds the value of EDSCR.INTdis. Reads and writes of this field are indirect accesses to EDSCR.INTdis.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'00'`.

Accessing this field has the following behavior:

- When OSLSR_EL1.OSLK == ‘1’, access to this field is RW.
- When OSLSR_EL1.OSLK == ‘0’, access to this field is RO.

TDA, bit [21]

Used for save/restore of EDSCR.TDA.

When OSLSR_EL1.OSLK == 0, software must treat this bit as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this bit holds the value of EDSCR.TDA. Reads and writes of this bit are indirect accesses to EDSCR.TDA.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- When OSLSR_EL1.OSLK == ‘1’, access to this field is RW.
- When OSLSR_EL1.OSLK == ‘0’, access to this field is RO.

Bit [20]

Reserved, RES0.

SC2, bit [19]

When FEAT_PCSRv8 is implemented, FEAT_VHE is implemented, and FEAT_PCSRv8p2 is not implemented:

Used for save/restore of EDSCR.SC2.

When OSLSR_EL1.OSLK == 0, software must treat this bit as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this bit holds the value of EDSCR.SC2. Reads and writes of this bit are indirect accesses to EDSCR.SC2.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- When OSLSR_EL1.OSLK == ‘1’, access to this field is RW.
- When OSLSR_EL1.OSLK == ‘0’, access to this field is RO.

Otherwise:

Reserved, RES0.

Bits [18:16]

Reserved, RAZ/WI.

Hardware must implement this field as RAZ/WI. Software must not rely on the register reading as zero, and must use a read-modify-write sequence to write to the register.

MDE, bit [15]

Monitor debug events. Enable Breakpoint, Watchpoint, and Vector Catch exceptions.

MDE	Meaning
`0b0`	Breakpoint, Watchpoint, and Vector Catch exceptions disabled.
`0b1`	Breakpoint, Watchpoint, and Vector Catch exceptions enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

HDE, bit [14]

Used for save/restore of EDSCR.HDE.

When OSLSR_EL1.OSLK == 0, software must treat this bit as UNK/SBZP.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When OSLSR_EL1.OSLK == 1, this bit holds the value of EDSCR.HDE. Reads and writes of this bit are indirect accesses to EDSCR.HDE.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- When OSLSR_EL1.OSLK == '1', access to this field is RW.
- When OSLSR_EL1.OSLK == '0', access to this field is RO.

KDE, bit [13]

Local (kernel) debug enable. If EL_D is using AArch64, enable debug exceptions within EL_D. Permitted values are:

KDE	Meaning
`0b0`	Debug exceptions, other than Breakpoint Instruction exceptions, disabled within EL_D.
`0b1`	All debug exceptions enabled within EL_D.

RES0 if EL_D is using AArch32.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TDCC, bit [12]

Traps EL0 accesses to the Debug Communication Channel (DCC) registers to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, from both Execution states, as follows:

- In AArch64 state, MRS or MSR accesses to the following DCC registers are trapped, reported using EC syndrome value `0x18`:
  - MDCCSR_EL0.
  - If not in Debug state, DBGDTR_EL0, DBGDTRTX_EL0, and DBGDTRRX_EL0.
- In AArch32 state, MRC or MCR accesses to the following registers are trapped, reported using EC syndrome value `0x05`.
  - DBGDSCRint, DBGDIDR, DBGDSAR, DBGDRAR.
  - If not in Debug state, DBGDTRRXint, and DBGDTRTXint.
- In AArch32 state, LDC access to DBGDTRRXint and STC access to DBGDTRTXint are trapped, reported using EC syndrome value `0x06`.
- In AArch32 state, MRRC accesses to DBGDSAR and DBGDRAR are trapped, reported using EC syndrome value `0x0C`.

TDCC	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	EL0 using AArch64: EL0 accesses to the AArch64 DCC registers are trapped. EL0 using AArch32: EL0 accesses to the AArch32 DCC registers are trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bits [11:7]

Reserved, RES0.

ERR, bit [6]

Used for save/restore of EDSCR.ERR.

When OSLSR_EL1.OSLK == 0, software must treat this bit as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this bit holds the value of EDSCR.ERR. Reads and writes of this bit are indirect accesses to EDSCR.ERR.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- When OSLSR_EL1.OSLK == ‘1’, access to this field is RW.
- When OSLSR_EL1.OSLK == ‘0’, access to this field is RO.

Bits [5:1]

Reserved, RES0.

SS, bit [0]

Software step control bit. If EL_D is using AArch64, enable Software step. Permitted values are:

SS	Meaning
0b0	Software step disabled
0b1	Software step enabled.

RES0 if EL_D is using AArch32.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MDSCR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MDSCR_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	0b0010	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.MDSCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x158];
    else
        X[t, 64] = MDSCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MDSCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MDSCR_EL1;
```

```
MSR MDSCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	0b0010	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.MDSCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x158] = X[t, 64];
    else
        MDSCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MDSCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MDSCR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.3.21 MDSELR_EL1, Breakpoint and Watchpoint Selection Register

The MDSELR_EL1 characteristics are:

Purpose

Selects the current breakpoints or watchpoints accessed by System register instructions.

Configuration

This register is present only when FEAT_Debugv8p9 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to MDSELR_EL1 are UNDEFINED.

Attributes

MDSELR_EL1 is a 64-bit register.

Field descriptions

Bits [63:6]

Reserved, RES0.

BANK, bits [5:4]

Breakpoint and watchpoint bank select.

BANK	Meaning	Applies when
0b00	Select 0 to 15.	
0b01	Select 16 to 31.	NUM_BREAKPOINTS > 16 or NUM_WATCHPOINTS > 16
0b10	Select 32 to 47.	NUM_BREAKPOINTS > 32 or NUM_WATCHPOINTS > 32
0b11	Select 48 to 63.	NUM_BREAKPOINTS > 48 or NUM_WATCHPOINTS > 48

Each of the following register names accesses a register for breakpoint or watchpoint <n>, where `n = UInt(MDSELR_EL1.BANK:m[3:0])`:

- DBGBCR<m>_EL1.
- DBGBVR<m>_EL1.
- DBGWCR<m>_EL1.
- DBGWVR<m>_EL1.

This field is ignored by the PE and treated as zeros when any of the following are true:

- Executing at EL3 and MDCR_EL3.EBWE is 0.
- Executing at EL2 and the Effective value of MDCR_EL2.EBWE is 0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- Executing at EL1 and the Effective value of MDSCR_EL1.EMBWE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is RES0 if all of the following are true:
  - NUM_BREAKPOINTS <= 16
  - NUM_WATCHPOINTS <= 16
- Otherwise, access to this field is RW.

Bits [3:0]

Reserved, RES0.

Accessing MDSELR_EL1

When 16 or fewer breakpoints are implemented, 16 or fewer watchpoints are implemented, and MDSELR_EL1 is implemented as RAZ/WI, it is IMPLEMENTATION DEFINED whether these trap controls have any effect on accesses to MDSELR_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MDSELR_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	0b0100	0b010

```
if !(IsFeatureImplemented(FEAT_Debugv8p9) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EBWE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGRTR2_EL2.nMDSELR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EBWE == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MDSELR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EBWE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif HaveEL(EL3) && MDCR_EL3.EBWE == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MDSELR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MDSELR_EL1;
```

```
MSR MDSELR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	0b0100	0b010

```
if !(IsFeatureImplemented(FEAT_Debugv8p9) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EBWE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGWTR2_EL2.nMDSELR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EBWE == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MDSELR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EBWE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EBWE == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MDSELR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MDSELR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.3.22 MDSTEPOP_EL1, Monitor Debug Step Opcode Register

The MDSTEPOP_EL1 characteristics are:

Purpose

Used to execute instructions while Software Step is in active-not-pending state.

Configuration

This register is present only when FEAT_STEP2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to MDSTEPOP_EL1 are UNDEFINED.

Attributes

MDSTEPOP_EL1 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

OPCODE, bits [31:0]

A64 instruction to be executed on the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MDSTEPOP_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MDSTEPOP_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	0b0101	0b010

```
if !(IsFeatureImplemented(FEAT_STEP2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnSTEPOP == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGRTR2_EL2.nMDSTEPOP_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnSTEPOP == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MDSTEPOP_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnSTEPOP == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnSTEPOP == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MDSTEPOP_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MDSTEPOP_EL1;
```

MSR MDSTEPOP_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	0b0101	0b010

```
if !(IsFeatureImplemented(FEAT_STEP2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnSTEPOP == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGWTR2_EL2.nMDSTEPOP_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnSTEPOP == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MDSTEPOP_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnSTEPOP == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnSTEPOP == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MDSTEPOP_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MDSTEPOP_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.3.23 OSDLR_EL1, OS Double Lock Register

The OSDLR_EL1 characteristics are:

Purpose

Used to control the OS Double Lock.

Configuration

AArch64 System register OSDLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGOSDLR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to OSDLR_EL1 are UNDEFINED.

Attributes

OSDLR_EL1 is a 64-bit register.

Field descriptions

Bits [63:1]

Reserved, RES0.

DLK, bit [0]

When FEAT_DoubleLock is implemented:

OS Double Lock control bit.

DLK	Meaning
0b0	OS Double Lock unlocked.
0b1	OS Double Lock locked, if DBGPRCR_EL1.CORENPDRQ (Core no powerdown request) bit is set to 0 and the PE is in Non-debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Otherwise:

Reserved, RAZ/WI.

Accessing OSDLR_EL1

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, OSDLR_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0001	0b0011	0b100

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDOSA == '1' &&
         ↪(IsFeatureImplemented(FEAT_DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by
         ↪MDCR_EL3.TDOSA") then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪IsFeatureImplemented(FEAT_DoubleLock) && HDFGRTR_EL2.OSDLR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDOSA> != '00' && (IsFeatureImplemented(FEAT_DoubleLock) ||
         ↪boolean IMPLEMENTATION_DEFINED "Trapped by MDCR_EL2.TDOSA") then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' && (IsFeatureImplemented(FEAT_DoubleLock) || boolean
         ↪IMPLEMENTATION_DEFINED "Trapped by MDCR_EL3.TDOSA") then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = OSDLR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDOSA == '1' &&
         ↪(IsFeatureImplemented(FEAT_DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by
         ↪MDCR_EL3.TDOSA") then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' && (IsFeatureImplemented(FEAT_DoubleLock) || boolean
         ↪IMPLEMENTATION_DEFINED "Trapped by MDCR_EL3.TDOSA") then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = OSDLR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = OSDLR_EL1;
```

```
MSR OSDLR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0001	0b0011	0b100

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDOSA == '1' &&
         ↪(IsFeatureImplemented(FEAT_DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by
         ↪MDCR_EL3.TDOSA") then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪IsFeatureImplemented(FEAT_DoubleLock) && HDFGWTR_EL2.OSDLR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDOSA> != '00' && (IsFeatureImplemented(FEAT_DoubleLock) ||
         ↪boolean IMPLEMENTATION_DEFINED "Trapped by MDCR_EL2.TDOSA") then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' && (IsFeatureImplemented(FEAT_DoubleLock) || boolean
         ↪IMPLEMENTATION_DEFINED "Trapped by MDCR_EL3.TDOSA") then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        OSDLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDOSA == '1' &&
         ↪(IsFeatureImplemented(FEAT_DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by
         ↪MDCR_EL3.TDOSA") then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' && (IsFeatureImplemented(FEAT_DoubleLock) || boolean
         ↪IMPLEMENTATION_DEFINED "Trapped by MDCR_EL3.TDOSA") then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        OSDLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    OSDLR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.3.24 OSDTRRX_EL1, OS Lock Data Transfer Register, Receive

The OSDTRRX_EL1 characteristics are:

Purpose

Used for save and restore of DBGDTRRX_EL0. It is a component of the Debug Communications Channel.

Configuration

AArch64 System register OSDTRRX_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGDTRRXext[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to OSDTRRX_EL1 are UNDEFINED.

Attributes

OSDTRRX_EL1 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

DTRRX, bits [31:0]

Update DTRRX without side-effect.

Writes to this register update the value in DTRRX and do not change RXfull.

Reads of this register return the last value written to DTRRX and do not change RXfull.

For the full behavior of the Debug Communications Channel, see 'The Debug Communication Channel and Instruction Transfer Register'.

Accessing OSDTRRX_EL1

Arm deprecates reads and writes of OSDTRRX_EL1 when the OS Lock is unlocked.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, OSDTRRX_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	0b0000	0b010

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    X[t, 64] = OSDTRRX_EL1;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1'
        ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = OSDTRRX_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1'
        ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = OSDTRRX_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = OSDTRRX_EL1;
```

```
MSR OSDTRRX_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    OSDTRRX_EL1 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1'
        ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        OSDTRRX_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1'
        ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        OSDTRRX_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    OSDTRRX_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.3.25 OSDTRTX_EL1, OS Lock Data Transfer Register, Transmit

The OSDTRTX_EL1 characteristics are:

Purpose

Used for save/restore of DBGDTRTX_EL0. It is a component of the Debug Communications Channel.

Configuration

AArch64 System register OSDTRTX_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGDTRTXext[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to OSDTRTX_EL1 are UNDEFINED.

Attributes

OSDTRTX_EL1 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

DTRTX, bits [31:0]

Return DTRTX without side-effect.

Reads of this register return the value in DTRTX and do not change TXfull.

Writes of this register update the value in DTRTX and do not change TXfull.

For the full behavior of the Debug Communications Channel, see 'The Debug Communication Channel and Instruction Transfer Register'.

Accessing OSDTRTX_EL1

Arm deprecates reads and writes of OSDTRTX_EL1 when the OS Lock is unlocked.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, OSDTRTX_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	0b0011	0b010

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    X[t, 64] = OSDTRTX_EL1;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1'
        ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = OSDTRTX_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1'
        ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = OSDTRTX_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = OSDTRTX_EL1;
```

```
MSR OSDTRTX_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	0b0011	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    OSDTRTX_EL1 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1'
        ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        OSDTRTX_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1'
        ↪then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_FGT) && MDCR_EL3.TDCC == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        OSDTRTX_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    OSDTRTX_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.3.26 OSECCR_EL1, OS Lock Exception Catch Control Register

The OSECCR_EL1 characteristics are:

Purpose

Provides a mechanism for an operating system to access the contents of EDECCR that are otherwise invisible to software, so it can save/restore the contents of EDECCR over powerdown on behalf of the external debugger.

Configuration

If OSLSR_EL1.OSLK == 0, then OSECCR_EL1 returns an UNKNOWN value on reads and ignores writes.

AArch64 System register OSECCR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGOSECCR[31:0].

AArch64 System register OSECCR_EL1 bits [31:0] are architecturally mapped to External register EDECCR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to OSECCR_EL1 are UNDEFINED.

Attributes

OSECCR_EL1 is a 64-bit register.

Field descriptions

When OSLSR_EL1.OSLK == '1':

Bits [63:32]

Reserved, RES0.

EDECCR, bits [31:0]

Used for save/restore to EDECCR over powerdown.

Reads or writes to this field are indirect accesses to EDECCR.

The reset behavior of this field is:

- On a Cold reset, this field resets to `0x00000000`.

Accessing OSECCR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, OSECCR_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	0b0110	0b010

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.OSECCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' then
        X[t, 64] = bits(64) UNKNOWN;
    else
        X[t, 64] = OSECCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' then
        X[t, 64] = bits(64) UNKNOWN;
    else
        X[t, 64] = OSECCR_EL1;
elsif PSTATE.EL == EL3 then
    if OSLSR_EL1.OSLK == '0' then
        X[t, 64] = bits(64) UNKNOWN;
    else
        X[t, 64] = OSECCR_EL1;
```

MSR OSECCR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b10	0b000	0b0000	0b0110	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.OSECCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' then
        return;
    else
        OSECCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' then
        return;
    else
        OSECCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if OSLSR_EL1.OSLK == '0' then
        return;
    else
        OSECCR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.3.27 OSLAR_EL1, OS Lock Access Register

The OSLAR_EL1 characteristics are:

Purpose

Used to lock or unlock the OS Lock.

Configuration

The OS Lock can also be locked or unlocked using DBGOSLAR.

AArch64 System register OSLAR_EL1 bits [31:0] are architecturally mapped to External register OSLAR_EL1[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to OSLAR_EL1 are UNDEFINED.

Attributes

OSLAR_EL1 is a 64-bit register.

Field descriptions

Bits [63:1]

Reserved, RES0.

OSLK, bit [0]

On writes to OSLAR_EL1, bit[0] is copied to the OS Lock.

Use OSLSR_EL1.OSLK to check the current status of the lock.

Accessing OSLAR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MSR OSLAR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0001	0b0000	0b100

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDOSA == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.OSLAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDOSA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        OSLAR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        OSLAR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    OSLAR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.3.28 OSLSR_EL1, OS Lock Status Register

The OSLSR_EL1 characteristics are:

Purpose

Provides the status of the OS Lock.

Configuration

AArch64 System register OSLSR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGOSLSR[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to OSLSR_EL1 are UNDEFINED.

Attributes

OSLSR_EL1 is a 64-bit register.

Field descriptions

Bits [63:4]

Reserved, RES0.

OSLM, bits [3, 0]

OS Lock model implemented. Identifies the form of OS save and restore mechanism implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

OSLM	Meaning
0b00	OS Lock not implemented.
0b10	OS Lock implemented.

All other values are reserved. In an Armv8 implementation the value 0b00 is not permitted.

Access to this field is RO.

nTT, bit [2]

Not 32-bit access. This bit is always RAZ. It indicates that a 32-bit access is needed to write the key to the OS Lock Access Register.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

OSLK, bit [1]

OS Lock Status.

OSLK	Meaning
0b0	OS Lock unlocked.
0b1	OS Lock locked.

The OS Lock is locked and unlocked by writing to the OS Lock Access Register.

The reset behavior of this field is:

- On a Cold reset, this field resets to '1'.

Accessing OSLSR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, OSLSR_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b0001	0b0001	0b100

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.OSLSR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDOSA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = OSLSR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = OSLSR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = OSLSR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.3.29 SDER32_EL2, AArch32 Secure Debug Enable Register

The SDER32_EL2 characteristics are:

Purpose

Allows access to the AArch32 register SDER from Secure EL2 and EL3 only.

Configuration

When EL3 is implemented, AArch64 System register SDER32_EL2 bits [63:0] are architecturally mapped to AArch64 System register SDER32_EL3[63:0].

AArch64 System register SDER32_EL2 bits [31:0] are architecturally mapped to AArch32 System register SDER[31:0].

This register is present only when EL2 is implemented, FEAT_SEL2 is implemented, FEAT_AA32EL1 is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to SDER32_EL2 are UNDEFINED.

Attributes

SDER32_EL2 is a 64-bit register.

Field descriptions

Bits [63:2]

Reserved, RES0.

SUNIDEN, bit [1]

Secure User Non-Invasive Debug Enable.

SUNIDEN	Meaning
0b0	This bit has no effect on non-invasive debug.
0b1	Non-invasive debug is allowed in Secure EL0 using AArch32.

When Secure EL1 is using AArch32, the forms of non-invasive debug affected by this control are:

- The PC Sample-based Profiling Extension. See About the PC Sample-based Profiling Extension.
- When SelfHostedTraceEnabled() == FALSE, processor trace.
- When EL3 is implemented, Performance Monitors.

When Secure EL1 is using AArch64, this bit has no effect.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SUIDEN, bit [0]

When EL3 is implemented:

Secure User Invasive Debug Enable.

SUIDEN	Meaning
0b0	This bit does not affect the generation of debug exceptions at Secure EL0.
0b1	If EL1 is using AArch32, debug exceptions from Secure EL0 are enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing SDER32_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SDER32_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0011	0b001

```
if !(HaveEL(EL2) && IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_AA32EL1) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !HaveEL(EL2) || !IsFeatureImplemented(FEAT_SEL2) || !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SDER32_EL2;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        X[t, 64] = SDER32_EL2;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MSR SDER32_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0011	0b001

```
if !(HaveEL(EL2) && IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_AA32EL1) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !HaveEL(EL2) || !IsFeatureImplemented(FEAT_SEL2) || !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SDER32_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        SDER32_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.3.30 SDER32_EL3, AArch32 Secure Debug Enable Register

The SDER32_EL3 characteristics are:

Purpose

Allows access to the AArch32 register SDER from AArch64 state only. Its value has no effect on execution in AArch64 state.

Configuration

When EL2 is implemented and FEAT_SEL2 is implemented, AArch64 System register SDER32_EL3 bits [63:0] are architecturally mapped to AArch64 System register SDER32_EL2[63:0].

AArch64 System register SDER32_EL3 bits [31:0] are architecturally mapped to AArch32 System register SDER[31:0].

This register is present only when EL3 is implemented, FEAT_AA32EL1 is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to SDER32_EL3 are UNDEFINED.

Attributes

SDER32_EL3 is a 64-bit register.

Field descriptions

Bits [63:2]

Reserved, RES0.

SUNIDEN, bit [1]

Secure User Non-Invasive Debug Enable.

SUNIDEN	Meaning
0b0	This bit has no effect on non-invasive debug.
0b1	Non-invasive debug is allowed in Secure EL0 using AArch32.

When Secure EL1 is using AArch32, the forms of non-invasive debug affected by this control are:

- The PC Sample-based Profiling Extension. See About the PC Sample-based Profiling Extension.
- When SelfHostedTraceEnabled() == FALSE, processor trace.
- Performance Monitors.

When Secure EL1 is using AArch64, this bit has no effect.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SUIDEN, bit [0]

Secure User Invasive Debug Enable.

SUIDEN	Meaning
0b0	This bit does not affect the generation of debug exceptions at Secure EL0.
0b1	If EL1 is using AArch32, debug exceptions from Secure EL0 are enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SDER32_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SDER32_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0001	0b001

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !HaveEL(EL3) || !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SDER32_EL3;
```

```
MSR SDER32_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0001	0b0001	0b001

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA32EL1) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif !HaveEL(EL3) || !IsFeatureImplemented(FEAT_AA32EL1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    SDER32_EL3 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4 Trace registers

This section lists the Trace registers in AArch64.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.1 TRBBASER_EL1, Trace Buffer Base Address Register

The TRBBASER_EL1 characteristics are:

Purpose

Defines the base address for the trace buffer.

Configuration

When FEAT_TRBE_EXT is implemented, AArch64 System register TRBBASER_EL1 bits [63:0] are architecturally mapped to External register TRBBASER_EL1[63:0].

This register is present only when FEAT_TRBE is implemented. Otherwise, direct accesses to TRBBASER_EL1 are UNDEFINED.

Attributes

TRBBASER_EL1 is a 64-bit register.

Field descriptions

BASE, bits [63:12]

Trace Buffer Base pointer address. (TRBBASER_EL1.BASE « 12) is the address of the first byte in the trace buffer. Bits [11:0] of the Base pointer address are always zero. If the smallest implemented translation granule is not 4KB, then TRBBASER_EL1[N-1:12] are RES0, where N is the IMPLEMENTATION DEFINED value Log_2(smallest implemented translation granule).

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [11:0]

Reserved, RES0.

Accessing TRBBASER_EL1

The PE might ignore a write to TRBBASER_EL1 if any of the following apply:

- TRBLIMITR_EL1.E == `0b1`, and either FEAT_TRBE_EXT is not implemented or the Trace Buffer Unit is using Self-hosted mode.
- TRBLIMITR_EL1.XE == `0b1`, FEAT_TRBE_EXT is implemented, and the Trace Buffer Unit is using External mode.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRBBASER_EL1
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b010

```
if !IsFeatureImplemented(FEAT_TRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRBBASER_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBBASER_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBBASER_EL1;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA ==
         ↪'1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBBASER_EL1;
```

```
MSR TRBBASER_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b010

```
if !IsFeatureImplemented(FEAT_TRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRBBASER_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBBASER_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBBASER_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA ==
         ↪'1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBBASER_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.2 TRBIDR_EL1, Trace Buffer ID Register

The TRBIDR_EL1 characteristics are:

Purpose

Describes constraints on using the Trace Buffer Unit to software, including whether the Trace Buffer Unit can be programmed at the current Exception level.

Configuration

When FEAT_TRBE_EXT is implemented, AArch64 System register TRBIDR_EL1 bits [63:0] are architecturally mapped to External register TRBIDR_EL1[63:0].

This register is present only when FEAT_TRBE is implemented. Otherwise, direct accesses to TRBIDR_EL1 are UNDEFINED.

Attributes

TRBIDR_EL1 is a 64-bit register.

Field descriptions

Bits [63:48]

Reserved, RES0.

MaxBuffSize, bits [47:32]

Maximum supported trace buffer size. Reserved for software use.

Reads as `0x0000`

The only permitted value is `0x0000`, indicating there is no limit to the maximum buffer size.

Note

Permitted values relate to the values an implementation is permitted to set this field to. A hypervisor might trap accesses to this register and use other values to describe limitations of its virtualization support to a guest operating system, as follows:

- MaxBuffSize bits[8:0] encodes a mantissa value, `M`.
- MaxBuffSize bits[13:9] encodes an exponent value, `E`.
- MaxBuffSize bits[15:14] are reserved.

The maximum buffer size, in bytes, is expressed using the following function:

```
if IsZero(E), then UInt(M:Zeros(12)) else UInt('1':M:Zeros(UInt(E)+11))
```

For example:

- A value of `0x0001` means a maximum buffer size of 4KB.
- A value of `0x3FFF` means a maximum buffer size of 4092TB.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bits [31:16]

Reserved, RES0.

MPAM, bits [15:12]

When FEAT_TRBE_EXT is implemented:

FEAT_MPAMv0p1 or FEAT_MPAMv1p0 extensions. Indicates Memory Partitioning and Monitoring (MPAM) support in the Trace Buffer Unit when using External mode.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MPAM	Meaning
0b0000	Trace Buffer External Mode is not implemented or the version of MPAM supported by this field is not implemented by the PE.
0b0001	FEAT_MPAMv0p1 or FEAT_MPAMv1p0 is implemented by the Trace Buffer Unit, using default PARTID and PMG values in External mode.
0b0010	Trace Buffer MPAM extensions implemented, using FEAT_MPAMv0p1 or FEAT_MPAMv1p0.

When FEAT_TRBE_EXT is not implemented by the PE, or if FEAT_MPAMv0p1 and FEAT_MPAMv1p0 are not implemented by the PE, the only permitted value is 0b0000.

When FEAT_TRBE_EXT is implemented by the PE, and at least one of FEAT_MPAMv0p1 and FEAT_MPAMv1p0 are implemented by the PE, the value 0b0000 is not permitted.

FEAT_TRBE_MPAM implements the functionality identified by the value 0b0010.

Access to this field is RO.

Otherwise:

Reserved, RES0.

EA, bits [11:8]

External Abort handling. Describes how the PE manages External aborts on writes made by the Trace Buffer Unit to the trace buffer.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EA	Meaning
0b0000	Not described.
0b0001	The PE ignores External aborts on writes made by the Trace Buffer Unit.
0b0010	An External abort on a write made by the Trace Buffer Unit generates an asynchronous SError exception at the PE.

All other values are reserved.

From Armv9.3, the value 0b0000 is not permitted.

The behavior described by this field does not apply for External aborts on a translation table walk, translation table update, or GPT walk made by the Trace Buffer Unit that are reported as MMU faults using TRBSR_ELx. For more information, see ‘External aborts’.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

AddrMode, bits [7:6]

Address Modes. Describes the addressing modes available for the trace buffer.

AddrMode	Meaning
`0b00`	Virtual and physical address modes are supported.
`0b01`	Only virtual address mode is supported.
`0b10`	Reserved for software use under virtualization, to show that only physical address mode is supported.

Other values are reserved.

If the Effective value of TRFCR_EL2.DnVM is 1 and the value returned for TRBIDR_EL1.P is 0, then this field reads as `0b01`. Otherwise, this field reads as `0b00`.

Note

A hypervisor might trap accesses to this register to describe limitations of its virtualization support to a guest operating system.

F, bit [5]

Flag updates. Describes how address translations performed by the Trace Buffer Unit manage the Access flag and dirty state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F	Meaning
`0b0`	Hardware management of the Access flag and dirty state for accesses made by the Trace Buffer Unit is always disabled for all translation stages.
`0b1`	Hardware management of the Access flag and dirty state for accesses made by the Trace Buffer Unit is controlled in the same way as explicit memory accesses in the trace buffer owning translation regime.

Note

If hardware management of the Access flag is disabled for a stage of translation, an access to a Page or Block with the Access flag bit not set in the descriptor will generate an Access Flag fault.
If hardware management of the dirty state is disabled for a stage of translation, an access to a Page or Block will ignore the Dirty Bit Modifier in the descriptor and might generate a Permission fault, depending on the values of the access permission bits in the descriptor.

From Armv9.3, the value 0 is not permitted.

Access to this field is RO.

P, bit [4]

Programming not allowed. When read at EL3, this field reads as zero. Otherwise, indicates that the trace buffer owning Exception level is a higher Exception level or the trace buffer owning Security state is not the current Security state.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

P	Meaning
0b0	Programming is allowed.
0b1	Programming not allowed.

The value read from this field depends on the current Exception level and the Effective values of MDCR_EL3.NSTB, MDCR_EL3.NSTBE, and MDCR_EL2.E2TB:

- If EL3 is implemented, MDCR_EL3.NSTB is 0b0x, and either FEAT_RME is not implemented, or Secure state is implemented and MDCR_EL3.NSTBE is 0, then this field reads as one from:
  - Non-secure EL1 and Non-secure EL2.
  - If FEAT_RME is implemented, Realm EL1 and Realm EL2.
  - If Secure EL2 is implemented and enabled, and MDCR_EL2.E2TB is 0b00, Secure EL1.
- If EL3 is implemented, MDCR_EL3.NSTB is 0b1x and either FEAT_RME is not implemented or MDCR_EL3.NSTBE is 0, then this field reads as one from:
  - If Secure state is implemented, Secure EL1.
  - If Secure EL2 is implemented, Secure EL2.
  - If EL2 is implemented and MDCR_EL2.E2TB is 0b00, Non-secure EL1.
  - If FEAT_RME is implemented, Realm EL1 and Realm EL2.
- If FEAT_RME is implemented, and MDCR_EL3.{NSTB, NSTBE} is {0b1x, 1}, then this field reads as one from:
  - Non-secure EL1 and Non-secure EL2.
  - If Secure state is implemented, Secure EL1 and Secure EL2.
  - If MDCR_EL2.E2TB is 0b00, Realm EL1.
- If EL3 is not implemented, EL2 is implemented, and MDCR_EL2.E2TB is 0b00, then this field reads as one from EL1.

Otherwise, this field reads as zero.

Align, bits [3:0]

Defines the minimum alignment constraint for writes to TRBPTR_EL1 and TRBTRG_EL1.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Align	Meaning
0b0000	Byte.
0b0001	Halfword.
0b0010	Word.
0b0011	Doubleword.
0b0100	16 bytes.
0b0101	32 bytes.
0b0110	64 bytes.
0b0111	128 bytes.
0b1000	256 bytes.
0b1001	512 bytes.
0b1010	1KB.
0b1011	2KB.

All other values are reserved.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing TRBIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRBIDR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b111

```
if !IsFeatureImplemented(FEAT_TRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        ↪HDFGRTR_EL2.TRBIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
        ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBIDR_EL1;
elsif PSTATE.EL == EL2 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA ==
        ↪'1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBIDR_EL1;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA ==
        ↪'1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBIDR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.3 TRBLIMITR_EL1, Trace Buffer Limit Address Register

The TRBLIMITR_EL1 characteristics are:

Purpose

Defines the top address for the trace buffer, and controls the trace buffer modes and enable.

Configuration

When FEAT_TRBE_EXT is implemented, AArch64 System register TRBLIMITR_EL1 bits [63:0] are architecturally mapped to External register TRBLIMITR_EL1[63:0].

This register is present only when FEAT_TRBE is implemented. Otherwise, direct accesses to TRBLIMITR_EL1 are UNDEFINED.

Attributes

TRBLIMITR_EL1 is a 64-bit register.

Field descriptions

LIMIT, bits [63:12]

Trace Buffer Limit pointer address. (TRBLIMITR_EL1.LIMIT « 12) is the address of the last byte in the trace buffer plus one. Bits [11:0] of the Limit pointer address are always zero. If the smallest implemented translation granule is not 4KB, then TRBLIMITR_EL1[N-1:12] are RES0, where N is the IMPLEMENTATION DEFINED value Log_2(smallest implemented translation granule).

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [11:7]

Reserved, RES0.

XE, bit [6]

When FEAT_TRBE_EXT is implemented:

Trace Buffer Unit External mode enable. Used for save/restore of TRBLIMITR_EL1.XE.

XE	Meaning
0b0	Trace Buffer Unit is not enabled by this control.
0b1	If SelfHostedTraceEnabled() is FALSE, the Trace Buffer Unit is enabled.

Software must treat this field as UNK/SBZP when the OS Lock is unlocked.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Cold reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- When !OSLockStatus(), access to this field is RO.
- Otherwise, access to this field is RW.

Otherwise:

Reserved, RES0.

nVM, bit [5]

Address mode.

nVM	Meaning
`0b0`	The trace buffer pointers are virtual addresses.
`0b1`	The trace buffer pointers are: • Physical address in the owning security state if the owning translation regime has no stage 2 translation. • Intermediate physical addresses in the owning security state if the owning translation regime has stage 2 translations.

If FEAT_TRBE_EXT is implemented and `SelfHostedTraceEnabled()` == FALSE, then the PE ignores the value of this field and the trace buffer pointers are always physical addresses.

If FEAT_TRBEv1p1 is implemented, `SelfHostedTraceEnabled()` == TRUE, and the Effective value of TRFCR_EL2.DnVM is 1, then the PE ignores the value of this field, and the trace buffer pointers are always virtual addresses.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is RES1 if all of the following are true:
  - FEAT_TRBE_EXT is implemented
  - !SelfHostedTraceEnabled()
- Otherwise, access to this field is RW.

TM, bits [4:3]

Trigger mode.

TM	Meaning
`0b00`	Stop on trigger. Flush then stop collection and raise maintenance interrupt on Trigger Event.
`0b01`	IRQ on trigger. Continue collection and raise maintenance interrupt on Trigger Event.
`0b11`	Ignore trigger. Continue collection and do not raise maintenance interrupt on Trigger Event.

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FM, bits [2:1]

Trace buffer mode.

FM	Meaning
`0b00`	Fill mode. Stop collection and raise maintenance interrupt on current write pointer wrap.
`0b01`	Wrap mode. Continue collection and raise maintenance interrupt on current write pointer wrap.
`0b11`	Circular Buffer mode. Continue collection and do not raise maintenance interrupt on current write pointer wrap.

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

E, bit [0]

Trace Buffer Unit enable. Controls whether the Trace Buffer Unit is enabled when `SelfHostedTraceEnabled()` == TRUE.

E	Meaning
`0b0`	Trace Buffer Unit is not enabled by this control.
`0b1`	If `SelfHostedTraceEnabled()` is TRUE, the Trace Buffer Unit is enabled.

If FEAT_TRBE_EXT is implemented and `SelfHostedTraceEnabled()` == FALSE, then TRBLIMITR_EL1.XE controls whether the Trace Buffer Unit is enabled.

If FEAT_TRBE_EXT is not implemented, then the Trace Buffer Unit is disabled when `SelfHostedTraceEnabled()` == FALSE.

All output is discarded by the Trace Buffer Unit when the Trace Buffer Unit is disabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Accessing TRBLIMITR_EL1

The PE might ignore a write to TRBLIMITR_EL1 if all the following are true:

- TRBLIMITR_EL1.E == `0b1`.
- Either FEAT_TRBE_EXT is not implemented or the Trace Buffer Unit is using Self-hosted mode.
- The write does not set TRBLIMITR_EL1.E to 0.

If FEAT_TRBE_EXT is implemented, the PE might ignore a write to TRBLIMITR_EL1 if all the following are true:

- TRBLIMITR_EL1.XE == `0b1`.
- The Trace Buffer Unit is using External mode.
- The write does not set TRBLIMITR_EL1.XE to 0.

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, TRBLIMITR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b000

```
if !IsFeatureImplemented(FEAT_TRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRBLIMITR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBLIMITR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBLIMITR_EL1;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA ==
         ↪'1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBLIMITR_EL1;
```

```
MSR TRBLIMITR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b000

```
if !IsFeatureImplemented(FEAT_TRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRBLIMITR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBLIMITR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBLIMITR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA ==
         ↪'1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBLIMITR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.4 TRBMAR_EL1, Trace Buffer Memory Attribute Register

The TRBMAR_EL1 characteristics are:

Purpose

Controls Trace Buffer Unit accesses to memory.

If the trace buffer pointers specify virtual addresses, the address properties are defined by the translation tables and this register is ignored.

Configuration

When FEAT_TRBE_EXT is implemented, AArch64 System register TRBMAR_EL1 bits [63:0] are architecturally mapped to External register TRBMAR_EL1[63:0].

This register is present only when FEAT_TRBE is implemented. Otherwise, direct accesses to TRBMAR_EL1 are UNDEFINED.

Attributes

TRBMAR_EL1 is a 64-bit register.

Field descriptions

Bits [63:12]

Reserved, RES0.

PAS, bits [11:10]

When FEAT_TRBE_EXT is implemented:

Physical address specifier. Defines the PAS attribute for memory addressed by the buffer in External mode.

PAS	Meaning	Applies when
0b00	Secure.	FEAT_Secure is implemented
0b01	Non-secure.	
0b10	Root.	FEAT_RME is implemented
0b11	Realm.	FEAT_RME is implemented

All other values are reserved.

If the Trace Buffer Unit is using external mode and either TRBMAR_EL1.PAS is set to a reserved value, or the IMPLEMENTATION DEFINED authentication interface prohibits invasive debug of the Security state corresponding to the physical address space selected by TRBMAR_EL1.PAS, then when the Trace Buffer Unit receives trace data from the trace unit, it does not write the trace data to memory and generates a trace buffer management event. That is, if any of the following apply:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- `ExternalInvasiveDebugEnabled()` == FALSE.
- TRBMAR_EL1.PAS is `0b00`, and either `ExternalSecureInvasiveDebugEnabled()` == FALSE or Secure state is not implemented.
- TRBMAR_EL1.PAS is `0b10`, and either `ExternalRootInvasiveDebugEnabled()` == FALSE or FEAT_RME is not implemented.
- TRBMAR_EL1.PAS is `0b11`, and either `ExternalRealmInvasiveDebugEnabled()` == FALSE or FEAT_RME is not implemented.

This field is ignored by the PE when `SelfHostedTraceEnabled()` == TRUE.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SH, bits [9:8]

Trace buffer shareability domain. Defines the shareability domain for Normal memory used by the trace buffer.

SH	Meaning
`0b00`	Non-shareable.
`0b10`	Outer Shareable.
`0b11`	Inner Shareable.

All other values are reserved.

This field is ignored when TRBMAR_EL1.Attr specifies any of the following memory types:

- Any Device memory type.
- Normal memory, Inner Non-cacheable, Outer Non-cacheable.

All Device and Normal Inner Non-cacheable Outer Non-cacheable memory regions are always treated as Outer Shareable.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_TRBE_EXT is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_TRBE_EXT is not implemented, this field resets to an architecturally UNKNOWN value.

Attr, bits [7:0]

Trace buffer memory type and attributes. Defines the memory type and, for Normal memory, the cacheability attributes, for memory addressed by the trace buffer.

The encoding of this field is the same as that of a MAIR_ELx.Attr<n> field, as follows:

Attr	Meaning
`0b0000`dd00	Device memory. See encoding of ‘dd’ for the type of Device memory.
`0b0000`dd01	If FEAT_XS is implemented: Device memory with the XS attribute set to 0. See encoding of ‘dd’ for the type of Device memory. Otherwise, UNPREDICTABLE.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Attr		Meaning
0b0000dd1x		UNPREDICTABLE.
0booooiiii	where oooo != 0000 and iiii != 0000	Normal memory. See encoding of ‘oooo’ and ‘iiii’ for the type of Normal memory.
0b01000000		If FEAT_XS is implemented: Normal Inner Non-cacheable, Outer Non-cacheable memory with the XS attribute set to 0. Otherwise, UNPREDICTABLE.
0b10100000		If FEAT_XS is implemented: Normal Inner Write-through Cacheable, Outer Write-through Cacheable, Read-Allocate, No-Write Allocate, Non-transient memory with the XS attribute set to 0. Otherwise, UNPREDICTABLE.
0b11110000		If FEAT_MTE2 is implemented: Tagged Normal Inner Write-Back, Outer Write-Back, Read-Allocate, Write-Allocate Non-transient memory. Otherwise, UNPREDICTABLE.
0bxxxx0000	where xxxx != 0000 and xxxx != 0100 and xxxx != 1010 and xxxx != 1111	UNPREDICTABLE.

dd is encoded as follows:

‘dd’	Meaning
0b00	Device-nGnRnE memory.
0b01	Device-nGnRE memory.
0b10	Device-nGRE memory.
0b11	Device-GRE memory.

oooo is encoded as follows:

‘oooo’		Meaning
0b0000		See encoding of Attr.
0b00RW	where RW != 00	Normal memory, Outer Write-Through Transient.
0b0100		Normal memory, Outer Non-cacheable.
0b01RW	where RW != 00	Normal memory, Outer Write-Back Transient.
0b10RW		Normal memory, Outer Write-Through Non-transient.
0b11RW		Normal memory, Outer Write-Back Non-transient.

R encodes the Outer Read-Allocate policy and W encodes the Outer Write-Allocate policy.

iiii is encoded as follows:

‘iiii’		Meaning
0b0000		See encoding of Attr.
0b00RW	where RW != 00	Normal memory, Inner Write-Through Transient.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

'iiii'		Meaning
0b0100		Normal memory, Inner Non-cacheable.
0b01RW	where RW != 00	Normal memory, Inner Write-Back Transient.
0b10RW		Normal memory, Inner Write-Through Non-transient.
0b11RW		Normal memory, Inner Write-Back Non-transient.

R encodes the Inner Read-Allocate policy and W encodes the Inner Write-Allocate policy.

In oooo and iiii, R and W are encoded as follows:

'R' or 'W'	Meaning
0b0	No Allocate.
0b1	Allocate.

When FEAT_XS is implemented, stage 1 Inner Write-Back Cacheable, Outer Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_TRBE_EXT is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_TRBE_EXT is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing TRBMAR_EL1

The PE might ignore a write to TRBMAR_EL1 if any of the following apply:

- TRBLIMITR_EL1.E == 0b1, and either FEAT_TRBE_EXT is not implemented or the Trace Buffer Unit is using Self-hosted mode.
- TRBLIMITR_EL1.XE == 0b1, FEAT_TRBE_EXT is implemented, and the Trace Buffer Unit is using External mode.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRBMAR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b100

```
if !IsFeatureImplemented(FEAT_TRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRBMAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif EL2Enabled() && MDCR_EL2.E2TB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBMAR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBMAR_EL1;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA ==
         ↪'1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBMAR_EL1;
```

MSR TRBMAR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b100

```
if !IsFeatureImplemented(FEAT_TRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRBMAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBMAR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBMAR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA ==
         ↪'1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBMAR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.5 TRBMPAM_EL1, Trace Buffer MPAM Configuration Register

The TRBMPAM_EL1 characteristics are:

Purpose

Defines the PARTID, PMG, and MPAM_SP values used by the trace buffer unit in external mode.

Configuration

AArch64 System register TRBMPAM_EL1 bits [63:0] are architecturally mapped to External register TRBMPAM_EL1[63:0].

This register is present only when FEAT_TRBE_MPAM is implemented. Otherwise, direct accesses to TRBMPAM_EL1 are UNDEFINED.

Attributes

TRBMPAM_EL1 is a 64-bit register.

Field descriptions

Bits [63:27]

Reserved, RES0.

EN, bit [26]

Enable. Enables use of non-default MPAM values.

EN	Meaning
0b0	Use default MPAM values.
0b1	Use TRBMPAM_EL1.{PARTID, PMG, MPAM_SP}.

This field is ignored by the PE when `SelfHostedTraceEnabled()` == TRUE.

The reset behavior of this field is:

- On a Cold reset, this field resets to `'0'`.

MPAM_SP, bits [25:24]

Partition Identifier space. Selects the PARTID space.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

MPAM_SP	Meaning	Applies when
`0b00`	PARTID is in the Secure PARTID space.	FEAT_Secure is implemented
`0b01`	PARTID is in the Non-secure PARTID space.	
`0b10`	PARTID is in the Root PARTID space.	FEAT_RME is implemented
`0b11`	PARTID is in the Realm PARTID space.	FEAT_RME is implemented

If the Trace Buffer Unit is using external mode and either TRBMPAM_EL1.MPAM_SP is set to reserved value, or the IMPLEMENTATION DEFINED authentication interface prohibits invasive debug of the Security state corresponding to the Partition Identifier space selected by TRBMPAM_EL1.MPAM_SP, then when the Trace Buffer Unit receives trace data from the trace unit, it does not write the trace data to memory and generates a trace buffer management event.

The interface prohibits invasive debug of the Security state if any of the following apply:

- `ExternalInvasiveDebugEnabled()` == FALSE.
- Secure state is implemented, `ExternalSecureInvasiveDebugEnabled()` == FALSE and TRBMPAM_EL1.MPAM_SP is `0b00`.
- FEAT_RME is implemented, `ExternalRootInvasiveDebugEnabled()` == FALSE, and TRBMPAM_EL1.MPAM_SP is `0b10`.
- FEAT_RME is implemented, `ExternalRealmInvasiveDebugEnabled()` == FALSE, and TRBMPAM_EL1.MPAM_SP is `0b11`.

This field is ignored by the PE when `SelfHostedTraceEnabled()` == TRUE.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

PMG, bits [23:16]

Performance Monitoring Group. Selects the PMG.

Only sufficient low-order bits are required to represent the TRBDEVID1.PMG_MAX. Higher-order bits are RES0.

This field is ignored by the PE when `SelfHostedTraceEnabled()` == TRUE.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

PARTID, bits [15:0]

Partition Identifier. Selects the PARTID.

Only sufficient low-order bits are required to represent the TRBDEVID1.PARTID_MAX. Higher-order bits are RES0.

This field is ignored by the PE when `SelfHostedTraceEnabled()` == TRUE.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing TRBMPAM_EL1

The PE might ignore a write to TRBMPAM_EL1 if any of the following apply:

- TRBLIMITR_EL1.E == `0b1`, and either FEAT_TRBE_EXT is not implemented or the Trace Buffer Unit is using Self-hosted mode.
- TRBLIMITR_EL1.XE == `0b1`, FEAT_TRBE_EXT is implemented, and the Trace Buffer Unit is using External mode.

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, TRBMPAM_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b101

```
if !IsFeatureImplemented(FEAT_TRBE_MPAM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnTB2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGRTR2_EL2.nTRBMPAM_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnTB2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBMPAM_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnTB2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnTB2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBMPAM_EL1;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA ==
         ↪'1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBMPAM_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MSR TRBMPAM_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b101

```
if !IsFeatureImplemented(FEAT_TRBE_MPAM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnTB2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGWTR2_EL2.nTRBMPAM_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnTB2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBMPAM_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnTB2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnTB2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBMPAM_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA ==
         ↪'1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBMPAM_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.6 TRBPTR_EL1, Trace Buffer Write Pointer Register

The TRBPTR_EL1 characteristics are:

Purpose

Defines the current write pointer for the trace buffer.

Configuration

When FEAT_TRBE_EXT is implemented, AArch64 System register TRBPTR_EL1 bits [63:0] are architecturally mapped to External register TRBPTR_EL1[63:0].

This register is present only when FEAT_TRBE is implemented. Otherwise, direct accesses to TRBPTR_EL1 are UNDEFINED.

Attributes

TRBPTR_EL1 is a 64-bit register.

Field descriptions

PTR, bits [63:0]

Trace Buffer current write pointer address.

Defines the virtual address of the next entry to be written to the trace buffer.

If TRBIDR_EL1.Align is not zero, then it is IMPLEMENTATION DEFINED whether bits [M-1:0] are RES0 or read/write, where M is an integer between 1 and TRBIDR_EL1.Align inclusive.

The architecture places restrictions on the values that software can write to the pointer. For more information, see Restrictions on Programming the Trace Buffer Unit.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing TRBPTR_EL1

The PE might ignore a write to TRBPTR_EL1 if any of the following apply:

- TRBLIMITR_EL1.E == `0b1`, and either FEAT_TRBE_EXT is not implemented or the Trace Buffer Unit is using Self-hosted mode.
- TRBLIMITR_EL1.XE == `0b1`, FEAT_TRBE_EXT is implemented, and the Trace Buffer Unit is using External mode.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRBPTR_EL1
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b001

```
if !IsFeatureImplemented(FEAT_TRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRBPTR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBPTR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBPTR_EL1;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA ==
         ↪'1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBPTR_EL1;
```

```
MSR TRBPTR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b001

```
if !IsFeatureImplemented(FEAT_TRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRBPTR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBPTR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBPTR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA ==
         ↪'1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBPTR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.7 TRBSR_EL1, Trace Buffer Status/syndrome Register (EL1)

The TRBSR_EL1 characteristics are:

Purpose

Provides syndrome information to software for a trace buffer management event.

Configuration

AArch64 System register TRBSR_EL1 bits [63:0] are architecturally mapped to External register TRBSR_EL1[63:0].

This register is present only when FEAT_TRBE is implemented. Otherwise, direct accesses to TRBSR_EL1 are UNDEFINED.

Attributes

TRBSR_EL1 is a 64-bit register.

Field descriptions

Bits [63:56]

Reserved, RES0.

MSS2, bits [55:32]

Management event Specific Syndrome 2. Contains syndrome specific to the management event.

The syndrome contents for each management event are described in the following sections.

MSS2 encoding for other trace buffer management events

Bits [23:0]

Reserved, RES0.

MSS2 encoding for stage 1 or stage 2 Data Aborts on write to trace buffer

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bits [23:9]

Reserved, RES0.

TopLevel, bit [8]

When FEAT_THE is implemented:

TopLevel. Indicates if the fault was due to TopLevel.

TopLevel	Meaning
0b0	Fault is not due to TopLevel.
0b1	Fault is due to TopLevel.

Otherwise:

Reserved, RES0.

AssuredOnly, bit [7]

When FEAT_THE is implemented, TRBSR_EL1.EC == '100101', and GetTRBSR_EL1_FSC() IN {'0011xx'}:

AssuredOnly flag. If a memory access generates a stage 2 Data Abort, then this field holds information about the fault.

AssuredOnly	Meaning
0b0	Data Abort is not due to AssuredOnly.
0b1	Data Abort is due to AssuredOnly.

Otherwise:

Reserved, RES0.

Overlay, bit [6]

When (FEAT_S1POE is implemented or FEAT_S2POE is implemented) and GetTRBSR_EL1_FSC() IN {'0011xx'}:

Overlay flag. If a memory access generates a Data Abort for a Permission fault, then this field holds information about the fault.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Overlay	Meaning
0b0	Data Abort is not due to Overlay Permissions.
0b1	Data Abort is due to Overlay Permissions.

Otherwise:

Reserved, RES0.

DirtyBit, bit [5]

When (FEAT_S1PIE is implemented or FEAT_S2PIE is implemented) and GetTRBSR_EL1_FSC() IN {'0011xx'}:

DirtyBit flag. If a write access to memory generates a Data Abort for a Permission fault using Indirect Permission, then this field holds information about the fault.

DirtyBit	Meaning
0b0	Permission Fault is not due to dirty state.
0b1	Permission Fault is due to dirty state.

Otherwise:

Reserved, RES0.

Bits [4:0]

Reserved, RES0.

MSS2 encoding for Granule Protection Check faults on write to trace buffer

23 … 0
RES0

Bits [23:0]

Reserved, RES0.

MSS2 encoding for trace buffer management event for an IMPLEMENTATION DEFINED reason

23 … 0
IMPLEMENTATION DEFINED

IMPLEMENTATION DEFINED, bits [23:0]

IMPLEMENTATION DEFINED.

EC, bits [31:26]

Event class. Top-level description of the cause of the trace buffer management event.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EC	Meaning	MSS	MSS2	Applies when
0b000000	Other trace buffer management event. All trace buffer management events other than those described by the other defined Event class codes.	MSS encoding for other trace buffer management events	MSS2 encoding for other trace buffer management events	
0b011110	Granule Protection Check fault on write to trace buffer, other than Granule Protection Fault (GPF). That is, any of the following: • Granule Protection Table (GPT) address size fault. • GPT walk fault. • Synchronous External abort on GPT fetch. A GPF on translation table walk or update is reported as either a Stage 1 or Stage 2 Data Abort, as appropriate. Other GPFs are reported as a Stage 1 Data Abort.	MSS encoding for Granule Protection Check faults on write to trace buffer	MSS2 encoding for Granule Protection Check faults on write to trace buffer	FEAT_RME is implemented
0b011111	Trace buffer management event for an IMPLEMENTATION DEFINED reason.	MSS encoding for trace buffer management event for an IMPLEMENTATION DEFINED reason	MSS2 encoding for trace buffer management event for an IMPLEMENTATION DEFINED reason	
0b100100	Stage 1 Data Abort on write to trace buffer.	MSS encoding for stage 1 or stage 2 Data Aborts on write to trace buffer	MSS2 encoding for stage 1 or stage 2 Data Aborts on write to trace buffer	
0b100101	Stage 2 Data Abort on write to trace buffer.	MSS encoding for stage 1 or stage 2 Data Aborts on write to trace buffer	MSS2 encoding for stage 1 or stage 2 Data Aborts on write to trace buffer	

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [25:24]

Reserved, RES0.

Bit [23]

When FEAT_TRBE_EXT is implemented:

Reserved, UNKNOWN.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

IRQ, bit [22]

Maintenance status. Indicates that a trace buffer management event has been recorded.

IRQ	Meaning
0b0	No trace buffer management event for EL1 has been recorded.
0b1	A trace buffer management event for EL1 has been recorded.

When FEAT_TRBE_EXC is implemented, this field indicates a management event for EL1.

If FEAT_TRBE_EXC is implemented and the TRBE Profiling exception for EL1 is enabled, then when this field is 1, a TRBE Profiling exception for EL1 is pending

If FEAT_TRBE_EXC is not implemented or the TRBE Profiling exception for EL1 is disabled, then this field drives the **TRBIRQ** trace buffer interrupt request signal.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

TRG, bit [21]

Triggered.

TRG	Meaning
0b0	No Detected Trigger has been observed since this field was last cleared to zero.
0b1	A Detected Trigger has been observed since this field was last cleared to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

WRAP, bit [20]

Wrapped.

WRAP	Meaning
0b0	The current write pointer has not wrapped since this field was last cleared to zero.
0b1	The current write pointer has wrapped since this field was last cleared to zero.

For each byte of trace the Trace Buffer Unit Accepts and writes to the trace buffer at the address in the current write pointer, if the current write pointer is equal to the Limit pointer minus one, the current write pointer is wrapped by setting it to the Base pointer, and this field is set to 1.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bit [19]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EA, bit [18]

When Armv9.3:

Reserved, RES0.

When the PE sets this bit as the result of an External abort:

External Abort.

EA	Meaning
0b0	An External abort has not been asserted.
0b1	An External abort has been asserted and detected by the Trace Buffer Unit.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

S, bit [17]

Stopped.

S	Meaning
0b0	Collection has not been stopped.
0b1	Collection is stopped.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bit [16]

Reserved, RES0.

MSS, bits [15:0]

Management Event Specific Syndrome. Contains syndrome specific to the trace buffer management event.

The syndrome contents for each trace buffer management event are described in the following sections.

MSS encoding for other trace buffer management events

Bits [15:6]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

BSC, bits [5:0]

Trace buffer status code

BSC	Meaning	Applies when
0b000000	Collection not stopped, or access not allowed.	
0b000001	Trace buffer filled. Collection stopped because the current write pointer wrapped to the base pointer and the trace buffer mode is Fill mode.	
0b000010	Trigger Event. Collection stopped because of a Trigger Event. See TRBTRG_EL1 for more information.	
0b000011	Manual Stop. Collection stopped because of a Manual Stop event. See TRBCR.ManStop for more information.	FEAT_TRBE_EXT is implemented
0b000100	Buffer size. The requested trace buffer size was too large.	

All other values are reserved.

MSS encoding for stage 1 or stage 2 Data Aborts on write to trace buffer

Bits [15:6]

Reserved, RES0.

FSC, bits [5:0]

Fault status code

FSC	Meaning	Applies when
0b000000	Address size fault, level 0 of translation or translation table base register.	
0b000001	Address size fault, level 1.	
0b000010	Address size fault, level 2.	
0b000011	Address size fault, level 3.	
0b000100	Translation fault, level 0.	
0b000101	Translation fault, level 1.	
0b000110	Translation fault, level 2.	
0b000111	Translation fault, level 3.	
0b001001	Access flag fault, level 1.	
0b001010	Access flag fault, level 2.	
0b001011	Access flag fault, level 3.	

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FSC	Meaning	Applies when
0b001000	Access flag fault, level 0.	FEAT_LPA2 is implemented
0b001100	Permission fault, level 0.	FEAT_LPA2 is implemented
0b001101	Permission fault, level 1.	
0b001110	Permission fault, level 2.	
0b001111	Permission fault, level 3.	
0b010000	Synchronous External abort, not on translation table walk or hardware update of translation table.	
0b010001	Asynchronous External abort.	
0b010010	Synchronous External abort on translation table walk or hardware update of translation table, level -2.	FEAT_D128 is implemented
0b010011	Synchronous External abort on translation table walk or hardware update of translation table, level -1.	FEAT_LPA2 is implemented
0b010100	Synchronous External abort on translation table walk or hardware update of translation table, level 0.	
0b010101	Synchronous External abort on translation table walk or hardware update of translation table, level 1.	
0b010110	Synchronous External abort on translation table walk or hardware update of translation table, level 2.	
0b010111	Synchronous External abort on translation table walk or hardware update of translation table, level 3.	
0b011011	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1.	FEAT_LPA2 is implemented and FEAT_RAS is not implemented
0b100001	Alignment fault.	
0b100010	Granule Protection Fault on translation table walk or hardware update of translation table, level -2.	FEAT_D128 is implemented and FEAT_RME is implemented
0b100011	Granule Protection Fault on translation table walk or hardware update of translation table, level -1.	FEAT_RME is implemented and FEAT_LPA2 is implemented
0b100100	Granule Protection Fault on translation table walk or hardware update of translation table, level 0.	FEAT_RME is implemented
0b100101	Granule Protection Fault on translation table walk or hardware update of translation table, level 1.	FEAT_RME is implemented
0b100110	Granule Protection Fault on translation table walk or hardware update of translation table, level 2.	FEAT_RME is implemented
0b100111	Granule Protection Fault on translation table walk or hardware update of translation table, level 3.	FEAT_RME is implemented

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FSC	Meaning	Applies when
0b101000	Granule Protection Fault, not on translation table walk or hardware update of translation table.	FEAT_RME is implemented
0b101001	Address size fault, level -1.	FEAT_LPA2 is implemented
0b101010	Translation fault, level -2.	FEAT_D128 is implemented
0b101011	Translation fault, level -1.	FEAT_LPA2 is implemented
0b101100	Address Size fault, level -2.	FEAT_D128 is implemented
0b110000	TLB conflict abort.	
0b110001	Unsupported atomic hardware update fault.	FEAT_HAFDBS is implemented

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

MSS encoding for Granule Protection Check faults on write to trace buffer

15 … 0
RES0

Bits [15:0]

Reserved, RES0.

MSS encoding for trace buffer management event for an IMPLEMENTATION DEFINED reason

15 … 0
IMPLEMENTATION DEFINED

IMPLEMENTATION DEFINED, bits [15:0]

IMPLEMENTATION DEFINED.

Accessing TRBSR_EL1

The PE might ignore a write to TRBSR_EL1 if any of the following apply:

- TRBLIMITR_EL1.E == 0b1, and either FEAT_TRBE_EXT is not implemented or the Trace Buffer Unit is using Self-hosted mode.
- TRBLIMITR_EL1.XE == 0b1, FEAT_TRBE_EXT is implemented, and the Trace Buffer Unit is using External mode.

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name TRBSR_EL1 or TRBSR_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, TRBSR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b011

```
if !IsFeatureImplemented(FEAT_TRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRBSR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} && (EffectiveTRFCR_EL2_EE() != '00' && TRFCR_EL1.EE != '00')
         ↪then
        X[t, 64] = NVMem[0x860];
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBSR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    elsif EffectiveTRFCR_EL2_EE() != '00' && ELIsInHost(EL2) then
        X[t, 64] = TRBSR_EL2;
    else
        X[t, 64] = TRBSR_EL1;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA ==
         ↪'1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBSR_EL1;
```

```
MSR TRBSR_EL1, <Xt>
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b011

```
if !IsFeatureImplemented(FEAT_TRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRBSR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} && (EffectiveTRFCR_EL2_EE() != '00' && TRFCR_EL1.EE != '00')
         ↪then
        NVMem[0x860] = X[t, 64];
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    elsif EffectiveTRFCR_EL2_EE() != '00' && ELIsInHost(EL2) then
        TRBSR_EL2 = X[t, 64];
    else
        TRBSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA ==
         ↪'1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBSR_EL1 = X[t, 64];
```

When FEAT_TRBE_EXC is implemented and FEAT_VHE is implemented MRS <Xt>, TRBSR_EL12

op0	op1	CRn	CRm	op2
0b11	0b101	0b1001	0b1011	0b011

```
if !(IsFeatureImplemented(FEAT_TRBE_EXC) && IsFeatureImplemented(FEAT_VHE)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        UNDEFINED;
    elsif PSTATE.EL == EL1 then
        if EffectiveHCR_EL2_NVx() == '101' then
            X[t, 64] = NVMem[0x860];
        elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if ELIsInHost(EL2) then
            if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
                UNDEFINED;
            elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
                if EL3SDDUndef() then
                    UNDEFINED;
                else
                    AArch64.SystemAccessTrap(EL3, 0x18);
            elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
                 ↪EDSCR2.TTA == '1' then
                Halt(DebugHalt_SoftwareAccess);
            else
                X[t, 64] = TRBSR_EL1;
        else
            UNDEFINED;
    elsif PSTATE.EL == EL3 then
        if ELIsInHost(EL2) then
            if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
                 ↪EDSCR2.TTA == '1' then
                Halt(DebugHalt_SoftwareAccess);
            else
                X[t, 64] = TRBSR_EL1;
        else
            UNDEFINED;
```

When FEAT_TRBE_EXC is implemented and FEAT_VHE is implemented MSR TRBSR_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b1001	0b1011	0b011

```
    if !(IsFeatureImplemented(FEAT_TRBE_EXC) && IsFeatureImplemented(FEAT_VHE)) then
        UNDEFINED;
    elsif PSTATE.EL == EL0 then
        UNDEFINED;
    elsif PSTATE.EL == EL1 then
        if EffectiveHCR_EL2_NVx() == '101' then
            NVMem[0x860] = X[t, 64];
        elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if ELIsInHost(EL2) then
            if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
                UNDEFINED;
            elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
                if EL3SDDUndef() then
                    UNDEFINED;
                else
                    AArch64.SystemAccessTrap(EL3, 0x18);
            elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
                 ↪EDSCR2.TTA == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            Halt(DebugHalt_SoftwareAccess);
        else
            TRBSR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() &&
             ↪EDSCR2.TTA == '1' then
            Halt(DebugHalt_SoftwareAccess);
        else
            TRBSR_EL1 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.8 TRBSR_EL2, Trace Buffer Syndrome Register (EL2)

The TRBSR_EL2 characteristics are:

Purpose

Provides syndrome information to software for a trace buffer management event.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_TRBE_EXC is implemented. Otherwise, direct accesses to TRBSR_EL2 are UNDEFINED.

Attributes

TRBSR_EL2 is a 64-bit register.

Field descriptions

Bits [63:56]

Reserved, RES0.

MSS2, bits [55:32]

Management event Specific Syndrome 2. Contains syndrome specific to the management event.

The syndrome contents for each management event are described in the following sections.

MSS2 encoding for other trace buffer management events

Bits [23:0]

Reserved, RES0.

MSS2 encoding for stage 1 or stage 2 Data Aborts on write to trace buffer

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bits [23:9]

Reserved, RES0.

TopLevel, bit [8]

When FEAT_THE is implemented:

TopLevel. Indicates if the fault was due to TopLevel.

TopLevel	Meaning
0b0	Fault is not due to TopLevel.
0b1	Fault is due to TopLevel.

Otherwise:

Reserved, RES0.

AssuredOnly, bit [7]

When FEAT_THE is implemented, TRBSR_EL2.EC == '100101', and GetTRBSR_EL2_FSC() IN {'0011xx'}:

AssuredOnly flag. If a memory access generates a stage 2 Data Abort, then this field holds information about the fault.

AssuredOnly	Meaning
0b0	Data Abort is not due to AssuredOnly.
0b1	Data Abort is due to AssuredOnly.

Otherwise:

Reserved, RES0.

Overlay, bit [6]

When (FEAT_S1POE is implemented or FEAT_S2POE is implemented) and GetTRBSR_EL2_FSC() IN {'0011xx'}:

Overlay flag. If a memory access generates a Data Abort for a Permission fault, then this field holds information about the fault.

Overlay	Meaning
0b0	Data Abort is not due to Overlay Permissions.
0b1	Data Abort is due to Overlay Permissions.

Otherwise:

Reserved, RES0.

DirtyBit, bit [5]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When (FEAT_S1PIE is implemented or FEAT_S2PIE is implemented) and GetTRBSR_EL2_FSC() IN {'0011xx'}:

DirtyBit flag. If a write access to memory generates a Data Abort for a Permission fault using Indirect Permission, then this field holds information about the fault.

DirtyBit	Meaning
`0b0`	Permission Fault is not due to dirty state.
`0b1`	Permission Fault is due to dirty state.

Otherwise:

Reserved, RES0.

Bits [4:0]

Reserved, RES0.

MSS2 encoding for Granule Protection Check faults on write to trace buffer

23 … 0
RES0

Bits [23:0]

Reserved, RES0.

MSS2 encoding for trace buffer management event for an IMPLEMENTATION DEFINED reason

23 … 0
IMPLEMENTATION DEFINED

IMPLEMENTATION DEFINED, bits [23:0]

IMPLEMENTATION DEFINED.

EC, bits [31:26]

Event class. Top-level description of the cause of the trace buffer management event.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EC	Meaning	MSS	MSS2	Applies when
0b000000	Other trace buffer management event. All trace buffer management events other than those described by the other defined Event class codes.	MSS encoding for other trace buffer management events	MSS2 encoding for other trace buffer management events	
0b011110	Granule Protection Check fault on write to trace buffer, other than Granule Protection Fault (GPF). That is, any of the following: • Granule Protection Table (GPT) address size fault. • GPT walk fault. • Synchronous External abort on GPT fetch. A GPF on translation table walk or update is reported as either a Stage 1 or Stage 2 Data Abort, as appropriate. Other GPFs are reported as a Stage 1 Data Abort.	MSS encoding for Granule Protection Check faults on write to trace buffer	MSS2 encoding for Granule Protection Check faults on write to trace buffer	FEAT_RME is implemented
0b011111	Trace buffer management event for an IMPLEMENTATION DEFINED reason.	MSS encoding for trace buffer management event for an IMPLEMENTATION DEFINED reason	MSS2 encoding for trace buffer management event for an IMPLEMENTATION DEFINED reason	
0b100100	Stage 1 Data Abort on write to trace buffer.	MSS encoding for stage 1 or stage 2 Data Aborts on write to trace buffer	MSS2 encoding for stage 1 or stage 2 Data Aborts on write to trace buffer	
0b100101	Stage 2 Data Abort on write to trace buffer.	MSS encoding for stage 1 or stage 2 Data Aborts on write to trace buffer	MSS2 encoding for stage 1 or stage 2 Data Aborts on write to trace buffer	

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [25:23]

Reserved, RES0.

IRQ, bit [22]

Maintenance status. Indicates that a trace buffer management event has been recorded.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

IRQ	Meaning
0b0	No trace buffer management event for EL2 has been recorded.
0b1	A trace buffer management event for EL2 has been recorded.

When FEAT_TRBE_EXC is implemented, this field indicates a management event for EL2.

If the TRBE Profiling exception for EL2 is enabled, then when this field is 1, a TRBE Profiling exception for EL2 is pending

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

TRG, bit [21]

Triggered.

TRG	Meaning
0b0	No Detected Trigger has been observed since this field was last cleared to zero.
0b1	A Detected Trigger has been observed since this field was last cleared to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

WRAP, bit [20]

Wrapped.

WRAP	Meaning
0b0	The current write pointer has not wrapped since this field was last cleared to zero.
0b1	The current write pointer has wrapped since this field was last cleared to zero.

For each byte of trace the Trace Buffer Unit Accepts and writes to the trace buffer at the address in the current write pointer, if the current write pointer is equal to the Limit pointer minus one, the current write pointer is wrapped by setting it to the Base pointer, and this field is set to 1.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bit [19]

Reserved, RES0.

EA, bit [18]

When Armv9.3:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When the PE sets this bit as the result of an External abort:

External Abort.

EA	Meaning
`0b0`	An External abort has not been asserted.
`0b1`	An External abort has been asserted and detected by the Trace Buffer Unit.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

S, bit [17]

Stopped.

S	Meaning
`0b0`	Collection has not been stopped.
`0b1`	Collection is stopped.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bit [16]

Reserved, RES0.

MSS, bits [15:0]

Management Event Specific Syndrome. Contains syndrome specific to the trace buffer management event.

The syndrome contents for each trace buffer management event are described in the following sections.

MSS encoding for other trace buffer management events

Bits [15:6]

Reserved, RES0.

BSC, bits [5:0]

Trace buffer status code

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

BSC	Meaning	Applies when
0b000000	Collection not stopped, or access not allowed.	
0b000001	Trace buffer filled. Collection stopped because the current write pointer wrapped to the base pointer and the trace buffer mode is Fill mode.	
0b000010	Trigger Event. Collection stopped because of a Trigger Event. See TRBTRG_EL1 for more information.	
0b000011	Manual Stop. Collection stopped because of a Manual Stop event. See TRBCR.ManStop for more information.	FEAT_TRBE_EXT is implemented
0b000100	Buffer size. The requested trace buffer size was too large.	

All other values are reserved.

MSS encoding for stage 1 or stage 2 Data Aborts on write to trace buffer

Bits [15:6]

Reserved, RES0.

FSC, bits [5:0]

Fault status code

FSC	Meaning	Applies when
0b000000	Address size fault, level 0 of translation or translation table base register.	
0b000001	Address size fault, level 1.	
0b000010	Address size fault, level 2.	
0b000011	Address size fault, level 3.	
0b000100	Translation fault, level 0.	
0b000101	Translation fault, level 1.	
0b000110	Translation fault, level 2.	
0b000111	Translation fault, level 3.	
0b001001	Access flag fault, level 1.	
0b001010	Access flag fault, level 2.	
0b001011	Access flag fault, level 3.	
0b001000	Access flag fault, level 0.	FEAT_LPA2 is implemented
0b001100	Permission fault, level 0.	FEAT_LPA2 is implemented
0b001101	Permission fault, level 1.	
0b001110	Permission fault, level 2.	

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FSC	Meaning	Applies when
0b001111	Permission fault, level 3.	
0b010000	Synchronous External abort, not on translation table walk or hardware update of translation table.	
0b010001	Asynchronous External abort.	
0b010010	Synchronous External abort on translation table walk or hardware update of translation table, level -2.	FEAT_D128 is implemented
0b010011	Synchronous External abort on translation table walk or hardware update of translation table, level -1.	FEAT_LPA2 is implemented
0b010100	Synchronous External abort on translation table walk or hardware update of translation table, level 0.	
0b010101	Synchronous External abort on translation table walk or hardware update of translation table, level 1.	
0b010110	Synchronous External abort on translation table walk or hardware update of translation table, level 2.	
0b010111	Synchronous External abort on translation table walk or hardware update of translation table, level 3.	
0b011011	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1.	FEAT_LPA2 is implemented and FEAT_RAS is not implemented
0b100001	Alignment fault.	
0b100010	Granule Protection Fault on translation table walk or hardware update of translation table, level -2.	FEAT_D128 is implemented and FEAT_RME is implemented
0b100011	Granule Protection Fault on translation table walk or hardware update of translation table, level -1.	FEAT_RME is implemented and FEAT_LPA2 is implemented
0b100100	Granule Protection Fault on translation table walk or hardware update of translation table, level 0.	FEAT_RME is implemented
0b100101	Granule Protection Fault on translation table walk or hardware update of translation table, level 1.	FEAT_RME is implemented
0b100110	Granule Protection Fault on translation table walk or hardware update of translation table, level 2.	FEAT_RME is implemented
0b100111	Granule Protection Fault on translation table walk or hardware update of translation table, level 3.	FEAT_RME is implemented
0b101000	Granule Protection Fault, not on translation table walk or hardware update of translation table.	FEAT_RME is implemented
0b101001	Address size fault, level -1.	FEAT_LPA2 is implemented
0b101010	Translation fault, level -2.	FEAT_D128 is implemented
0b101011	Translation fault, level -1.	FEAT_LPA2 is implemented

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FSC	Meaning	Applies when
0b101100	Address Size fault, level -2.	FEAT_D128 is implemented
0b110000	TLB conflict abort.	
0b110001	Unsupported atomic hardware update fault.	FEAT_HAFDBS is implemented

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

MSS encoding for Granule Protection Check faults on write to trace buffer

15	0
RES0	

Bits [15:0]

Reserved, RES0.

MSS encoding for trace buffer management event for an IMPLEMENTATION DEFINED reason

15	0
IMPLEMENTATION DEFINED	

IMPLEMENTATION DEFINED, bits [15:0]

IMPLEMENTATION DEFINED.

Accessing TRBSR_EL2

The PE might ignore a write to TRBSR_EL2 if any of the following apply:

- TRBLIMITR_EL1.E == 0b1, and either FEAT_TRBE_EXT is not implemented or the Trace Buffer Unit is using Self-hosted mode.
- TRBLIMITR_EL1.XE == 0b1, FEAT_TRBE_EXT is implemented, and the Trace Buffer Unit is using External mode.

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name TRBSR_EL2 or TRBSR_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRBSR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1001	0b1011	0b011

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_TRBE_EXC) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TRBEE == '00' then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TRBEE == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = TRBSR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TRBSR_EL2;
```

MSR TRBSR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1001	0b1011	0b011

```
if !IsFeatureImplemented(FEAT_TRBE_EXC) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TRBEE == '00' then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TRBEE == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        TRBSR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TRBSR_EL2 = X[t, 64];
```

MRS <Xt>, TRBSR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b011

```
if !IsFeatureImplemented(FEAT_TRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRBSR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} && (EffectiveTRFCR_EL2_EE() != '00' && TRFCR_EL1.EE != '00')
         ↪then
        X[t, 64] = NVMem[0x860];
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBSR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    elsif EffectiveTRFCR_EL2_EE() != '00' && ELIsInHost(EL2) then
        X[t, 64] = TRBSR_EL2;
    else
        X[t, 64] = TRBSR_EL1;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA ==
         ↪'1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBSR_EL1;
```

MSR TRBSR_EL1, <Xt>

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b011

```
if !IsFeatureImplemented(FEAT_TRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRBSR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} && (EffectiveTRFCR_EL2_EE() != '00' && TRFCR_EL1.EE != '00')
         ↪then
        NVMem[0x860] = X[t, 64];
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    elsif EffectiveTRFCR_EL2_EE() != '00' && ELIsInHost(EL2) then
        TRBSR_EL2 = X[t, 64];
    else
        TRBSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA ==
         ↪'1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBSR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.9 TRBSR_EL3, Trace Buffer Syndrome Register (EL3)

The TRBSR_EL3 characteristics are:

Purpose

Provides syndrome information to software for a trace buffer management event.

Configuration

This register is present only when FEAT_TRBE_EXC is implemented and EL3 is implemented. Otherwise, direct accesses to TRBSR_EL3 are UNDEFINED.

Attributes

TRBSR_EL3 is a 64-bit register.

Field descriptions

Bits [63:56]

Reserved, RES0.

MSS2, bits [55:32]

Management event Specific Syndrome 2. Contains syndrome specific to the management event.

The syndrome contents for each management event are described in the following sections.

MSS2 encoding for other trace buffer management events

Bits [23:0]

Reserved, RES0.

MSS2 encoding for stage 1 or stage 2 Data Aborts on write to trace buffer

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bits [23:9]

Reserved, RES0.

TopLevel, bit [8]

When FEAT_THE is implemented:

TopLevel. Indicates if the fault was due to TopLevel.

TopLevel	Meaning
0b0	Fault is not due to TopLevel.
0b1	Fault is due to TopLevel.

Otherwise:

Reserved, RES0.

AssuredOnly, bit [7]

When FEAT_THE is implemented, TRBSR_EL3.EC == '100101', and GetTRBSR_EL3_FSC() IN {'0011xx'}:

AssuredOnly flag. If a memory access generates a stage 2 Data Abort, then this field holds information about the fault.

AssuredOnly	Meaning
0b0	Data Abort is not due to AssuredOnly.
0b1	Data Abort is due to AssuredOnly.

Otherwise:

Reserved, RES0.

Overlay, bit [6]

When (FEAT_S1POE is implemented or FEAT_S2POE is implemented) and GetTRBSR_EL3_FSC() IN {'0011xx'}:

Overlay flag. If a memory access generates a Data Abort for a Permission fault, then this field holds information about the fault.

Overlay	Meaning
0b0	Data Abort is not due to Overlay Permissions.
0b1	Data Abort is due to Overlay Permissions.

Otherwise:

Reserved, RES0.

DirtyBit, bit [5]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When (FEAT_S1PIE is implemented or FEAT_S2PIE is implemented) and GetTRBSR_EL3_FSC() IN {'0011xx'}:

DirtyBit flag. If a write access to memory generates a Data Abort for a Permission fault using Indirect Permission, then this field holds information about the fault.

DirtyBit	Meaning
0b0	Permission Fault is not due to dirty state.
0b1	Permission Fault is due to dirty state.

Otherwise:

Reserved, RES0.

Bits [4:0]

Reserved, RES0.

MSS2 encoding for Granule Protection Check faults on write to trace buffer

23 … 0
RES0

Bits [23:0]

Reserved, RES0.

MSS2 encoding for trace buffer management event for an IMPLEMENTATION DEFINED reason

23 … 0
IMPLEMENTATION DEFINED

IMPLEMENTATION DEFINED, bits [23:0]

IMPLEMENTATION DEFINED.

EC, bits [31:26]

Event class. Top-level description of the cause of the trace buffer management event.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EC	Meaning	MSS	MSS2	Applies when
0b000000	Other trace buffer management event. All trace buffer management events other than those described by the other defined Event class codes.	MSS encoding for other trace buffer management events	MSS2 encoding for other trace buffer management events	
0b011110	Granule Protection Check fault on write to trace buffer, other than Granule Protection Fault (GPF). That is, any of the following: • Granule Protection Table (GPT) address size fault. • GPT walk fault. • Synchronous External abort on GPT fetch. A GPF on translation table walk or update is reported as either a Stage 1 or Stage 2 Data Abort, as appropriate. Other GPFs are reported as a Stage 1 Data Abort.	MSS encoding for Granule Protection Check faults on write to trace buffer	MSS2 encoding for Granule Protection Check faults on write to trace buffer	FEAT_RME is implemented
0b011111	Trace buffer management event for an IMPLEMENTATION DEFINED reason.	MSS encoding for trace buffer management event for an IMPLEMENTATION DEFINED reason	MSS2 encoding for trace buffer management event for an IMPLEMENTATION DEFINED reason	
0b100100	Stage 1 Data Abort on write to trace buffer.	MSS encoding for stage 1 or stage 2 Data Aborts on write to trace buffer	MSS2 encoding for stage 1 or stage 2 Data Aborts on write to trace buffer	
0b100101	Stage 2 Data Abort on write to trace buffer.	MSS encoding for stage 1 or stage 2 Data Aborts on write to trace buffer	MSS2 encoding for stage 1 or stage 2 Data Aborts on write to trace buffer	

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [25:23]

Reserved, RES0.

IRQ, bit [22]

Maintenance status. Indicates that a trace buffer management event has been recorded.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

IRQ	Meaning
0b0	No trace buffer management event for EL3 has been recorded.
0b1	A trace buffer management event for EL3 has been recorded.

When FEAT_TRBE_EXC is implemented, this field indicates a management event for EL3.

If the TRBE Profiling exception for EL3 is enabled, then when this field is 1, a TRBE Profiling exception for EL3 is pending

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

TRG, bit [21]

Triggered.

TRG	Meaning
0b0	No Detected Trigger has been observed since this field was last cleared to zero.
0b1	A Detected Trigger has been observed since this field was last cleared to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

WRAP, bit [20]

Wrapped.

WRAP	Meaning
0b0	The current write pointer has not wrapped since this field was last cleared to zero.
0b1	The current write pointer has wrapped since this field was last cleared to zero.

For each byte of trace the Trace Buffer Unit Accepts and writes to the trace buffer at the address in the current write pointer, if the current write pointer is equal to the Limit pointer minus one, the current write pointer is wrapped by setting it to the Base pointer, and this field is set to 1.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bit [19]

Reserved, RES0.

EA, bit [18]

When Armv9.3:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When the PE sets this bit as the result of an External abort:

External Abort.

EA	Meaning
0b0	An External abort has not been asserted.
0b1	An External abort has been asserted and detected by the Trace Buffer Unit.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

S, bit [17]

Stopped.

S	Meaning
0b0	Collection has not been stopped.
0b1	Collection is stopped.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bit [16]

Reserved, RES0.

MSS, bits [15:0]

Management Event Specific Syndrome. Contains syndrome specific to the trace buffer management event.

The syndrome contents for each trace buffer management event are described in the following sections.

MSS encoding for other trace buffer management events

Bits [15:6]

Reserved, RES0.

BSC, bits [5:0]

Trace buffer status code

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

BSC	Meaning	Applies when
0b000000	Collection not stopped, or access not allowed.	
0b000001	Trace buffer filled. Collection stopped because the current write pointer wrapped to the base pointer and the trace buffer mode is Fill mode.	
0b000010	Trigger Event. Collection stopped because of a Trigger Event. See TRBTRG_EL1 for more information.	
0b000011	Manual Stop. Collection stopped because of a Manual Stop event. See TRBCR.ManStop for more information.	FEAT_TRBE_EXT is implemented
0b000100	Buffer size. The requested trace buffer size was too large.	

All other values are reserved.

MSS encoding for stage 1 or stage 2 Data Aborts on write to trace buffer

Bits [15:6]

Reserved, RES0.

FSC, bits [5:0]

Fault status code

FSC	Meaning	Applies when
0b000000	Address size fault, level 0 of translation or translation table base register.	
0b000001	Address size fault, level 1.	
0b000010	Address size fault, level 2.	
0b000011	Address size fault, level 3.	
0b000100	Translation fault, level 0.	
0b000101	Translation fault, level 1.	
0b000110	Translation fault, level 2.	
0b000111	Translation fault, level 3.	
0b001001	Access flag fault, level 1.	
0b001010	Access flag fault, level 2.	
0b001011	Access flag fault, level 3.	
0b001000	Access flag fault, level 0.	FEAT_LPA2 is implemented
0b001100	Permission fault, level 0.	FEAT_LPA2 is implemented
0b001101	Permission fault, level 1.	
0b001110	Permission fault, level 2.	

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FSC	Meaning	Applies when
0b001111	Permission fault, level 3.	
0b010000	Synchronous External abort, not on translation table walk or hardware update of translation table.	
0b010001	Asynchronous External abort.	
0b010010	Synchronous External abort on translation table walk or hardware update of translation table, level -2.	FEAT_D128 is implemented
0b010011	Synchronous External abort on translation table walk or hardware update of translation table, level -1.	FEAT_LPA2 is implemented
0b010100	Synchronous External abort on translation table walk or hardware update of translation table, level 0.	
0b010101	Synchronous External abort on translation table walk or hardware update of translation table, level 1.	
0b010110	Synchronous External abort on translation table walk or hardware update of translation table, level 2.	
0b010111	Synchronous External abort on translation table walk or hardware update of translation table, level 3.	
0b011011	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1.	FEAT_LPA2 is implemented and FEAT_RAS is not implemented
0b100001	Alignment fault.	
0b100010	Granule Protection Fault on translation table walk or hardware update of translation table, level -2.	FEAT_D128 is implemented and FEAT_RME is implemented
0b100011	Granule Protection Fault on translation table walk or hardware update of translation table, level -1.	FEAT_RME is implemented and FEAT_LPA2 is implemented
0b100100	Granule Protection Fault on translation table walk or hardware update of translation table, level 0.	FEAT_RME is implemented
0b100101	Granule Protection Fault on translation table walk or hardware update of translation table, level 1.	FEAT_RME is implemented
0b100110	Granule Protection Fault on translation table walk or hardware update of translation table, level 2.	FEAT_RME is implemented
0b100111	Granule Protection Fault on translation table walk or hardware update of translation table, level 3.	FEAT_RME is implemented
0b101000	Granule Protection Fault, not on translation table walk or hardware update of translation table.	FEAT_RME is implemented
0b101001	Address size fault, level -1.	FEAT_LPA2 is implemented
0b101010	Translation fault, level -2.	FEAT_D128 is implemented
0b101011	Translation fault, level -1.	FEAT_LPA2 is implemented

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FSC	Meaning	Applies when
0b101100	Address Size fault, level -2.	FEAT_D128 is implemented
0b110000	TLB conflict abort.	
0b110001	Unsupported atomic hardware update fault.	FEAT_HAFDBS is implemented

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

MSS encoding for Granule Protection Check faults on write to trace buffer

15	0
RES0	

Bits [15:0]

Reserved, RES0.

MSS encoding for trace buffer management event for an IMPLEMENTATION DEFINED reason

15	0
IMPLEMENTATION DEFINED	

IMPLEMENTATION DEFINED, bits [15:0]

IMPLEMENTATION DEFINED.

Accessing TRBSR_EL3

The PE might ignore a write to TRBSR_EL3 if any of the following apply:

- TRBLIMITR_EL1.E == 0b1, and either FEAT_TRBE_EXT is not implemented or the Trace Buffer Unit is using Self-hosted mode.
- TRBLIMITR_EL1.XE == 0b1, FEAT_TRBE_EXT is implemented, and the Trace Buffer Unit is using External mode.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRBSR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1001	0b1011	0b011

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_TRBE_EXC) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TRBSR_EL3;
```

```
MSR TRBSR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1001	0b1011	0b011

```
if !(IsFeatureImplemented(FEAT_TRBE_EXC) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TRBSR_EL3 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.10 TRBTRG_EL1, Trace Buffer Trigger Counter Register

The TRBTRG_EL1 characteristics are:

Purpose

Specifies the number of bytes of trace to capture following a Detected Trigger before a Trigger Event.

Configuration

When FEAT_TRBE_EXT is implemented, AArch64 System register TRBTRG_EL1 bits [63:0] are architecturally mapped to External register TRBTRG_EL1[63:0].

This register is present only when FEAT_TRBE is implemented. Otherwise, direct accesses to TRBTRG_EL1 are UNDEFINED.

Attributes

TRBTRG_EL1 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

TRG, bits [31:0]

Trigger count.

Specifies the number of bytes of trace to capture following a Detected Trigger before a Trigger Event.

TRBTRG_EL1 decrements by 1 for every byte of trace written to the trace buffer when all of the following are true:

- TRBTRG_EL1 is nonzero.
- TRBSR_EL1.TRG is 1.

The architecture places restrictions on the values that software can write to the counter.

Note

As a result of the restrictions an implementation might treat some of TRG[M:0] as RES0, where M is defined by TRBIDR_EL1.Align.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing TRBTRG_EL1

The PE might ignore a write to TRBTRG_EL1 if any of the following apply:

- TRBLIMITR_EL1.E == `0b1`, and either FEAT_TRBE_EXT is not implemented or the Trace Buffer Unit is using Self-hosted mode.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- TRBLIMITR_EL1.XE == `0b1`, FEAT_TRBE_EXT is implemented, and the Trace Buffer Unit is using External mode.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRBTRG_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b110

```
if !IsFeatureImplemented(FEAT_TRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRBTRG_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBTRG_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBTRG_EL1;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA ==
         ↪'1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRBTRG_EL1;
```

```
MSR TRBTRG_EL1, <Xt>
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1011	0b110

```
if !IsFeatureImplemented(FEAT_TRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRBTRG_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBTRG_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSTBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSTBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBTRG_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA ==
         ↪'1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRBTRG_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.11 TRCACATR<n>, Trace Address Comparator Access Type Register <n>, n = 0 - 15

The TRCACATR<n> characteristics are:

Purpose

Defines the type of access for the corresponding TRCACVR<n> Register. This register configures the context type, Exception levels, alignment, masking that is applied by the Address Comparator, and how the Address Comparator behaves when it is one half of an Address Range Comparator.

Configuration

AArch64 System register TRCACATR<n> bits [63:0] are architecturally mapped to External register TRCACATR<n>[63:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and (UInt(TRCIDR4.NUMACPAIRS) * 2) > n. Otherwise, direct accesses to TRCACATR<n> are UNDEFINED.

Attributes

TRCACATR<n> is a 64-bit register.

Field descriptions

Bits [63:19]

Reserved, RES0.

EXLEVEL_RL_EL2, bit [18]

When FEAT_RME is implemented:

Realm EL2 address comparison control. Controls whether a comparison can occur at EL2 in Realm state.

EXLEVEL_RL_EL2	Meaning
0b0	When TRCACATR<n>.EXLEVEL_NS_EL2 is 0 the Address Comparator performs comparisons in Realm EL2. When TRCACATR<n>.EXLEVEL_NS_EL2 is 1 the Address Comparator does not perform comparisons in Realm EL2.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EXLEVEL_RL_EL2	Meaning
0b1	When TRCACATR<n>.EXLEVEL_NS_EL2 is 0 the Address Comparator does not perform comparisons in Realm EL2. When TRCACATR<n>.EXLEVEL_NS_EL2 is 1 the Address Comparator performs comparisons in Realm EL2.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_RL_EL1, bit [17]

When FEAT_RME is implemented:

Realm EL1 address comparison control. Controls whether a comparison can occur at EL1 in Realm state.

EXLEVEL_RL_EL1	Meaning
0b0	When TRCACATR<n>.EXLEVEL_NS_EL1 is 0 the Address Comparator performs comparisons in Realm EL1. When TRCACATR<n>.EXLEVEL_NS_EL1 is 1 the Address Comparator does not perform comparisons in Realm EL1.
0b1	When TRCACATR<n>.EXLEVEL_NS_EL1 is 0 the Address Comparator does not perform comparisons in Realm EL1. When TRCACATR<n>.EXLEVEL_NS_EL1 is 1 the Address Comparator performs comparisons in Realm EL1.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_RL_EL0, bit [16]

When FEAT_RME is implemented:

Realm EL0 address comparison control. Controls whether a comparison can occur at EL0 in Realm state.

EXLEVEL_RL_EL0	Meaning
0b0	When TRCACATR<n>.EXLEVEL_NS_EL0 is 0 the Address Comparator performs comparisons in Realm EL0. When TRCACATR<n>.EXLEVEL_NS_EL0 is 1 the Address Comparator does not perform comparisons in Realm EL0.
0b1	When TRCACATR<n>.EXLEVEL_NS_EL0 is 0 the Address Comparator does not perform comparisons in Realm EL0. When TRCACATR<n>.EXLEVEL_NS_EL0 is 1 the Address Comparator performs comparisons in Realm EL0.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [15]

Reserved, RES0.

EXLEVEL_NS_EL2, bit [14]

When Non-secure EL2 is implemented:

Non-secure EL2 address comparison control. Controls whether a comparison can occur at EL2 in Non-secure state.

EXLEVEL_NS_EL2	Meaning
0b0	The Address Comparator performs comparisons in Non-secure EL2.
0b1	The Address Comparator does not perform comparisons in Non-secure EL2.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_NS_EL1, bit [13]

When Non-secure EL1 is implemented:

Non-secure EL1 address comparison control. Controls whether a comparison can occur at EL1 in Non-secure state.

EXLEVEL_NS_EL1	Meaning
0b0	The Address Comparator performs comparisons in Non-secure EL1.
0b1	The Address Comparator does not perform comparisons in Non-secure EL1.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_NS_EL0, bit [12]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When Non-secure EL0 is implemented:

Non-secure EL0 address comparison control. Controls whether a comparison can occur at EL0 in Non-secure state.

EXLEVEL_NS_EL0	Meaning
0b0	The Address Comparator performs comparisons in Non-secure EL0.
0b1	The Address Comparator does not perform comparisons in Non-secure EL0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL3, bit [11]

When EL3 is implemented:

EL3 address comparison control. Controls whether a comparison can occur at EL3.

EXLEVEL_S_EL3	Meaning
0b0	The Address Comparator performs comparisons at EL3.
0b1	The Address Comparator does not perform comparisons at EL3.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL2, bit [10]

When Secure EL2 is implemented:

Secure EL2 address comparison control. Controls whether a comparison can occur at EL2 in Secure state.

EXLEVEL_S_EL2	Meaning
0b0	The Address Comparator performs comparisons in Secure EL2.
0b1	The Address Comparator does not perform comparisons in Secure EL2.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL1, bit [9]

When Secure EL1 is implemented:

Secure EL1 address comparison control. Controls whether a comparison can occur at EL1 in Secure state.

EXLEVEL_S_EL1	Meaning
0b0	The Address Comparator performs comparisons in Secure EL1.
0b1	The Address Comparator does not perform comparisons in Secure EL1.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL0, bit [8]

When Secure EL0 is implemented:

Secure EL0 address comparison control. Controls whether a comparison can occur at EL0 in Secure state.

EXLEVEL_S_EL0	Meaning
0b0	The Address Comparator performs comparisons in Secure EL0.
0b1	The Address Comparator does not perform comparisons in Secure EL0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [7]

Reserved, RES0.

CONTEXT, bits [6:4]

When TRCIDR4.NUMCIDC != '0000' or TRCIDR4.NUMVMIDC != '0000':

Selects a Context Identifier Comparator or Virtual Context Identifier Comparator:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CONTEXT	Meaning	Applies when
0b000	Comparator 0.	
0b001	Comparator 1.	UInt(TRCIDR4.NUMCIDC) > 1 or UInt(TRCIDR4.NUMVMIDC) > 1
0b010	Comparator 2.	UInt(TRCIDR4.NUMCIDC) > 2 or UInt(TRCIDR4.NUMVMIDC) > 2
0b011	Comparator 3.	UInt(TRCIDR4.NUMCIDC) > 3 or UInt(TRCIDR4.NUMVMIDC) > 3
0b100	Comparator 4.	UInt(TRCIDR4.NUMCIDC) > 4 or UInt(TRCIDR4.NUMVMIDC) > 4
0b101	Comparator 5.	UInt(TRCIDR4.NUMCIDC) > 5 or UInt(TRCIDR4.NUMVMIDC) > 5
0b110	Comparator 6.	UInt(TRCIDR4.NUMCIDC) > 6 or UInt(TRCIDR4.NUMVMIDC) > 6
0b111	Comparator 7.	UInt(TRCIDR4.NUMCIDC) > 7 or UInt(TRCIDR4.NUMVMIDC) > 7

The width of this field is dependent on the maximum number of Context Identifier Comparators or Virtual Context Identifier Comparators implemented. Unimplemented bits are RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CONTEXTTYPE, bits [3:2]

When TRCIDR4.NUMCIDC != '0000' or TRCIDR4.NUMVMIDC != '0000':

Controls whether the Address Comparator is dependent on a Context Identifier Comparator, a Virtual Context Identifier Comparator, or both comparisons.

CONTEXTTYPE	Meaning	Applies when
0b00	The Address Comparator is not dependent on the Context Identifier Comparators or Virtual Context Identifier Comparators.	
0b01	The Address Comparator is dependent on the Context Identifier Comparator that TRCACATR<n>.CONTEXT specifies. The Address Comparator signals a match only if both the Context Identifier Comparator and the address comparison match.	TRCIDR4.NUMCIDC != ‘0000’
0b10	The Address Comparator is dependent on the Virtual Context Identifier Comparator that TRCACATR<n>.CONTEXT specifies. The Address Comparator signals a match only if both the Virtual Context Identifier Comparator and the address comparison match.	TRCIDR4.NUMVMIDC != ‘0000’

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CONTEXTTYPE	Meaning	Applies when
0b11	The Address Comparator is dependent on the Context Identifier Comparator and Virtual Context Identifier Comparator that TRCACATR<n>.CONTEXT specifies. The Address Comparator signals a match only if the Context Identifier Comparator, the Virtual Context Identifier Comparator, and address comparison all match.	TRCIDR4.NUMCIDC != '0000' and TRCIDR4.NUMVMIDC != '0000'

If TRCIDR4.NUMCIDC == 0b0000, then bit [2] is RES0.

If TRCIDR4.NUMVMIDC == 0b0000, then bit [3] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [1:0]

Reserved, RES0.

Accessing TRCACATR<n>

Must be programmed if any of the following are true:

- TRCBBCTLR.RANGE[n/2] == 1.
- TRCRSCTLR<a>.GROUP == 0b0100 and TRCRSCTLR<a>.SAC[n] == 1.
- TRCRSCTLR<a>.GROUP == 0b0101 and TRCRSCTLR<a>.ARC[n/2] == 1.
- TRCVIIECTLR.EXCLUDE[n/2] == 1.
- TRCVIIECTLR.INCLUDE[n/2] == 1.
- TRCVISSCTLR.START[n] == 1.
- TRCVISSCTLR.STOP[n] == 1.
- TRCSSCCR<>.ARC[n/2] == 1.
- TRCSSCCR<>.SAC[n] == 1.
- TRCQCTLR.RANGE[n/2] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCACATR<m> ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0010	m[2:0]:0b0	0b01:m[3]

```
integer m = UInt(op2<0>:CRm<3:1>);

if m >= NUM_TRACE_ADDRESS_COMPARATOR_PAIRS * 2 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCACATR[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCACATR[m];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCACATR[m];
```

```
MSR TRCACATR<m>, <Xt> ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0010	m[2:0]:0b0	0b01:m[3]

```
integer m = UInt(op2<0>:CRm<3:1>);

if m >= NUM_TRACE_ADDRESS_COMPARATOR_PAIRS * 2 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRC == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCACATR[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCACATR[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCACATR[m] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.12 TRCACVR<n>, Trace Address Comparator Value Register <n>, n = 0 - 15

The TRCACVR<n> characteristics are:

Purpose

Contains the address value.

Configuration

AArch64 System register TRCACVR<n> bits [63:0] are architecturally mapped to External register TRCACVR<n>[63:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and (UInt(TRCIDR4.NUMACPAIRS) * 2) > n. Otherwise, direct accesses to TRCACVR<n> are UNDEFINED.

Attributes

TRCACVR<n> is a 64-bit register.

Field descriptions

ADDRESS, bits [63:0]

Address Value.

The Address Comparators can support implementations that use multiple address widths. When the trace unit compares the ADDRESS field with an address that has a width less than this field, then the address must be zero-extended to the ADDRESS field width. The trace unit then compares all implemented bits. For example, in a system that supports both 32-bit and 64-bit addresses, when the PE is in AArch32 state the comparator must zero-extend the 32-bit address and compare against the full 64 bits that are stored in TRCACVR.ADDRESS. This requires that the trace analyzer always programs all implemented bits of TRCACVR.ADDRESS.

The result of writing a value other than all zeros or all ones to ADDRESS at bits[63:P] is an UNKNOWN value, where P is defined as:

- 56, when FEAT_LVA3 is implemented.
- 52, when FEAT_LVA is implemented.
- 48, otherwise.

The result of writing a value of all zeros or all ones to ADDRESS at bits[63:P] is the written value, and a read of the register returns the written value.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCACVR<n>

Must be programmed if any of the following are true:

- TRCBBCTLR.RANGE[n/2] == 1.
- TRCRSCTLR<a>.GROUP == `0b0100` and TRCRSCTLR<a>.SAC[n] == 1.
- TRCRSCTLR<a>.GROUP == `0b0101` and TRCRSCTLR<a>.ARC[n/2] == 1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- TRCVIIECTLR.EXCLUDE[n/2] == 1.
- TRCVIIECTLR.INCLUDE[n/2] == 1.
- TRCVISSCTLR.START[n] == 1.
- TRCVISSCTLR.STOP[n] == 1.
- TRCSSCCR<>.ARC[n/2] == 1.
- TRCSSCCR<>.SAC[n] == 1.
- TRCQCTLR.RANGE[n/2] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCACVR<m> ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0010	m[2:0]:0b0	0b00:m[3]

```
integer m = UInt(op2<0>:CRm<3:1>);

if m >= NUM_TRACE_ADDRESS_COMPARATOR_PAIRS * 2 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCACVR[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCACVR[m];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
        ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCACVR[m];
```

MSR TRCACVR<m>, <Xt> ; Where m = 0-15

op0	op1	CRn	CRm	op2
0b10	0b001	0b0010	m[2:0]:0b0	0b00:m[3]

```
integer m = UInt(op2<0>:CRm<3:1>);

if m >= NUM_TRACE_ADDRESS_COMPARATOR_PAIRS * 2 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        ↪HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
        ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCACVR[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
        ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCACVR[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
        ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCACVR[m] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.13 TRCAUTHSTATUS, Trace Authentication Status Register

The TRCAUTHSTATUS characteristics are:

Purpose

Provides information about the state of the IMPLEMENTATION DEFINED authentication interface for debug.

For additional information, see the CoreSight Architecture Specification.

Configuration

AArch64 System register TRCAUTHSTATUS bits [31:0] are architecturally mapped to External register TRCAUTHSTATUS[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCAUTHSTATUS are UNDEFINED.

Attributes

TRCAUTHSTATUS is a 64-bit register.

Field descriptions

Bits [63:28]

Reserved, RES0.

RTNID, bits [27:26]

Root non-invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.RTNID.

RTID, bits [25:24]

Root invasive debug.

RTID	Meaning
0b00	Not implemented.

Bits [23:16]

Reserved, RES0.

RLNID, bits [15:14]

Realm non-invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.RLNID.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

RLID, bits [13:12]

Realm invasive debug.

RLID	Meaning
0b00	Not implemented.

HNID, bits [11:10]

Hyp Non-invasive Debug. Indicates whether a separate enable control for EL2 non-invasive debug features is implemented and enabled.

HNID	Meaning
0b00	Separate Hyp non-invasive debug enable not implemented, or EL2 non-invasive debug features not implemented.
0b10	Implemented and disabled.
0b11	Implemented and enabled.

All other values are reserved.

This field reads as 0b00.

HID, bits [9:8]

Hyp Invasive Debug. Indicates whether a separate enable control for EL2 invasive debug features is implemented and enabled.

HID	Meaning
0b00	Separate Hyp invasive debug enable not implemented, or EL2 invasive debug features not implemented.
0b10	Implemented and disabled.
0b11	Implemented and enabled.

All other values are reserved.

This field reads as 0b00.

SNID, bits [7:6]

Secure Non-invasive Debug. Indicates whether Secure non-invasive debug features are implemented and enabled.

SNID	Meaning
0b00	Secure non-invasive debug features not implemented.
0b10	Implemented and disabled.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SNID	Meaning
0b11	Implemented and enabled.

All other values are reserved.

When Secure state is implemented, this field reads as 0b10 or 0b11 depending whether Secure non-invasive debug is enabled.

When Secure state is not implemented, this field reads as 0b00.

SID, bits [5:4]

Secure Invasive Debug. Indicates whether Secure invasive debug features are implemented and enabled.

SID	Meaning
0b00	Secure invasive debug features not implemented.
0b10	Implemented and disabled.
0b11	Implemented and enabled.

All other values are reserved.

This field reads as 0b00.

NSNID, bits [3:2]

Non-secure Non-invasive Debug. Indicates whether Non-secure non-invasive debug features are implemented and enabled.

NSNID	Meaning
0b00	Non-secure non-invasive debug features not implemented.
0b10	Implemented and disabled.
0b11	Implemented and enabled.

All other values are reserved.

When Non-secure state is implemented, this field reads as 0b11.

When Non-secure state is not implemented, this field reads as 0b00.

NSID, bits [1:0]

Non-secure Invasive Debug. Indicates whether Non-secure invasive debug features are implemented and enabled.

NSID	Meaning
0b00	Non-secure invasive debug features not implemented.
0b10	Implemented and disabled.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

NSID	Meaning
0b11	Implemented and enabled.

All other values are reserved.

This field reads as 0b00.

Accessing TRCAUTHSTATUS

For implementations that support multiple access mechanisms, different access mechanisms can return different values for reads of TRCAUTHSTATUS if the authentication signals have changed and that change has not yet been synchronized by a Context synchronization event. This scenario can happen if, for example, the external debugger view is implemented separately from the system instruction view to allow for separate power domains, and so observes changes on the signals differently.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCAUTHSTATUS
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0111	0b1110	0b110

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRCAUTHSTATUS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCAUTHSTATUS;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    else
        X[t, 64] = TRCAUTHSTATUS;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCAUTHSTATUS;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.14 TRCAUXCTLR, Trace Auxiliary Control Register

The TRCAUXCTLR characteristics are:

Purpose

The function of this register is IMPLEMENTATION DEFINED.

Configuration

AArch64 System register TRCAUXCTLR bits [31:0] are architecturally mapped to External register TRCAUXCTLR[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCAUXCTLR are UNDEFINED.

Attributes

TRCAUXCTLR is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to `0x00000000`.

Accessing TRCAUXCTLR

If this register is nonzero then it might cause the behavior of a trace unit to contradict this architecture specification. See the documentation of the specific implementation for information about the IMPLEMENTATION DEFINED support for this register.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCAUXCTLR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0110	0b000

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRCAUXCTLR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCAUXCTLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCAUXCTLR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCAUXCTLR;
```

```
MSR TRCAUXCTLR, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0110	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRCAUXCTLR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCAUXCTLR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCAUXCTLR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCAUXCTLR = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.15 TRCBBCTLR, Trace Branch Broadcast Control Register

The TRCBBCTLR characteristics are:

Purpose

Controls the regions in the memory map where branch broadcasting is active.

Configuration

AArch64 System register TRCBBCTLR bits [31:0] are architecturally mapped to External register TRCBBCTLR[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, TRCIDR0.TRCBB == '1', and UInt(TRCIDR4.NUMACPAIRS) > 0. Otherwise, direct accesses to TRCBBCTLR are UNDEFINED.

Attributes

TRCBBCTLR is a 64-bit register.

Field descriptions

Bits [63:9]

Reserved, RES0.

MODE, bit [8]

Mode.

MODE	Meaning
0b0	Exclude Mode. Branch broadcasting is not active for instructions in the address ranges defined by TRCBBCTLR.RANGE. If TRCBBCTLR.RANGE == 0x00 then branch broadcasting is active for all instructions.
0b1	Include Mode. Branch broadcasting is active for instructions in the address ranges defined by TRCBBCTLR.RANGE. If TRCBBCTLR.RANGE == 0x00 then the behavior of the trace unit is CONSTRAINED UNPREDICTABLE. That is, the trace unit might or might not consider any instructions to be in a branch broadcasting region.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

RANGE[<m>] , bits [m], for m = 7 to 0

Selects whether Address Range Comparator <m> is used with branch broadcasting.

RANGE[<m>]	Meaning
0b0	The address range that Address Range Comparator <m> defines, is not selected.
0b1	The address range that Address Range Comparator <m> defines, is selected.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR4.NUMACPAIRS), access to this field is RES0.
- Otherwise, access to this field is RW.

Accessing TRCBBCTLR

Must be programmed if TRCCONFIGR.BB == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCBBCTLR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1111	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && TRCIDR0.TRCBB == '1' &&
     ↪UInt(TRCIDR4.NUMACPAIRS) > 0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCBBCTLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCBBCTLR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCBBCTLR;
```

```
MSR TRCBBCTLR, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1111	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && TRCIDR0.TRCBB == '1' &&
     ↪UInt(TRCIDR4.NUMACPAIRS) > 0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCBBCTLR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCBBCTLR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCBBCTLR = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.16 TRCCCCTLR, Trace Cycle Count Control Register

The TRCCCCTLR characteristics are:

Purpose

Set the threshold value for cycle counting.

Configuration

AArch64 System register TRCCCCTLR bits [31:0] are architecturally mapped to External register TRCCCCTLR[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and TRCIDR0.TRCCCI == '1'. Otherwise, direct accesses to TRCCCCTLR are UNDEFINED.

Attributes

TRCCCCTLR is a 64-bit register.

Field descriptions

Bits [63:12]

Reserved, RES0.

THRESHOLD, bits [11:0]

Sets the threshold value for instruction trace cycle counting.

The minimum threshold value that can be programmed into THRESHOLD is given in TRCIDR3.CCITMIN. If the THRESHOLD value is smaller than the value in TRCIDR3.CCITMIN then the behavior is CONSTRAINED UNPREDICTABLE. That is, cycle counts might or might not be included in the trace and the cycle count threshold is not known.

Writing a value of zero when TRCCONFIGR.CCI enables instruction trace cycle counting results in CONSTRAINED UNPREDICTABLE behavior. That is, cycle counts might or might not be included in the trace and the cycle count threshold is not known.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCCCCTLR

Must be programmed if TRCCONFIGR.CCI == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCCCCTLR
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1110	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && TRCIDR0.TRCCCI == '1') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCCCTLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCCCTLR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCCCTLR;
```

```
MSR TRCCCCTLR, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1110	0b000

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && TRCIDR0.TRCCCI == '1') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCCCTLR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCCCTLR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCCCTLR = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.17 TRCCIDCCTLR0, Trace Context Identifier Comparator Control Register 0

The TRCCIDCCTLR0 characteristics are:

Purpose

Contains Context identifier mask values for the TRCCIDCVR<n> registers, for n = 0 to 3.

Configuration

AArch64 System register TRCCIDCCTLR0 bits [31:0] are architecturally mapped to External register TRCCIDCCTLR0[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, UInt(TRCIDR4.NUMCIDC) > `0x0`, and UInt(TRCIDR2.CIDSIZE) > 0. Otherwise, direct accesses to TRCCIDCCTLR0 are UNDEFINED.

Attributes

TRCCIDCCTLR0 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

COMP3[<m>] , bits [m+24], for m = 7 to 0

When UInt(TRCIDR4.NUMCIDC) > 3:

TRCCIDCVR3 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR3. Each bit in this field corresponds to a byte in TRCCIDCVR3.

COMP3[<m>]	Meaning
`0b0`	The trace unit includes TRCCIDCVR3[(m×8+7):(m×8)] when it performs the Context identifier comparison.
`0b1`	The trace unit ignores TRCCIDCVR3[(m×8+7):(m×8)] when it performs the Context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.CIDSIZE), access to this field is RES0.
- Otherwise, access to this field is RW.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

COMP2[<m>] , bits [m+16], for m = 7 to 0

When UInt(TRCIDR4.NUMCIDC) > 2:

TRCCIDCVR2 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR2. Each bit in this field corresponds to a byte in TRCCIDCVR2.

COMP2[<m>]	Meaning
0b0	The trace unit includes TRCCIDCVR2[(m×8+7):(m×8)] when it performs the Context identifier comparison.
0b1	The trace unit ignores TRCCIDCVR2[(m×8+7):(m×8)] when it performs the Context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.CIDSIZE), access to this field is RES0.
- Otherwise, access to this field is RW.

Otherwise:

Reserved, RES0.

COMP1[<m>] , bits [m+8], for m = 7 to 0

When UInt(TRCIDR4.NUMCIDC) > 1:

TRCCIDCVR1 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR1. Each bit in this field corresponds to a byte in TRCCIDCVR1.

COMP1[<m>]	Meaning
0b0	The trace unit includes TRCCIDCVR1[(m×8+7):(m×8)] when it performs the Context identifier comparison.
0b1	The trace unit ignores TRCCIDCVR1[(m×8+7):(m×8)] when it performs the Context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.CIDSIZE), access to this field is RES0.
- Otherwise, access to this field is RW.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

COMP0[<m>] , bits [m], for m = 7 to 0

When UInt(TRCIDR4.NUMCIDC) > 0:

TRCCIDCVR0 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR0. Each bit in this field corresponds to a byte in TRCCIDCVR0.

COMP0[<m>]	Meaning
0b0	The trace unit includes TRCCIDCVR0[(m×8+7):(m×8)] when it performs the Context identifier comparison.
0b1	The trace unit ignores TRCCIDCVR0[(m×8+7):(m×8)] when it performs the Context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.CIDSIZE), access to this field is RES0.
- Otherwise, access to this field is RW.

Otherwise:

Reserved, RES0.

Accessing TRCCIDCCTLR0

If software uses the TRCCIDCVR<n> registers, for n = 0 to 3, then it must program this register.

If software sets a mask bit to 1 then it must program the relevant byte in TRCCIDCVR<n> to 0x00.

If any bit is 1 and the relevant byte in TRCCIDCVR<n> is not 0x00, the behavior of the Context Identifier Comparator is CONSTRAINED UNPREDICTABLE. In this scenario the comparator might match unexpectedly or might not match.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCCIDCCTLR0
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0011	0b0000	0b010

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && UInt(TRCIDR4.NUMCIDC) > 0x0
     ↪&& UInt(TRCIDR2.CIDSIZE) > 0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCIDCCTLR0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCIDCCTLR0;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCIDCCTLR0;
```

```
MSR TRCCIDCCTLR0, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0011	0b0000	0b010

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && UInt(TRCIDR4.NUMCIDC) > 0x0
     ↪&& UInt(TRCIDR2.CIDSIZE) > 0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCIDCCTLR0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCIDCCTLR0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCIDCCTLR0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.18 TRCCIDCCTLR1, Trace Context Identifier Comparator Control Register 1

The TRCCIDCCTLR1 characteristics are:

Purpose

Contains Context identifier mask values for the TRCCIDCVR<n> registers, for n = 4 to 7.

Configuration

AArch64 System register TRCCIDCCTLR1 bits [31:0] are architecturally mapped to External register TRCCIDCCTLR1[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, UInt(TRCIDR4.NUMCIDC) > `0x4`, and UInt(TRCIDR2.CIDSIZE) > 0. Otherwise, direct accesses to TRCCIDCCTLR1 are UNDEFINED.

Attributes

TRCCIDCCTLR1 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

COMP7[<m>] , bits [m+24], for m = 7 to 0

When UInt(TRCIDR4.NUMCIDC) > 7:

TRCCIDCVR7 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR7. Each bit in this field corresponds to a byte in TRCCIDCVR7.

COMP7[<m>]	Meaning
`0b0`	The trace unit includes TRCCIDCVR7[(m×8+7):(m×8)] when it performs the Context identifier comparison.
`0b1`	The trace unit ignores TRCCIDCVR7[(m×8+7):(m×8)] when it performs the Context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.CIDSIZE), access to this field is RES0.
- Otherwise, access to this field is RW.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

COMP6[<m>] , bits [m+16], for m = 7 to 0

When UInt(TRCIDR4.NUMCIDC) > 6:

TRCCIDCVR6 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR6. Each bit in this field corresponds to a byte in TRCCIDCVR6.

COMP6[<m>]	Meaning
0b0	The trace unit includes TRCCIDCVR6[(m×8+7):(m×8)] when it performs the Context identifier comparison.
0b1	The trace unit ignores TRCCIDCVR6[(m×8+7):(m×8)] when it performs the Context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.CIDSIZE), access to this field is RES0.
- Otherwise, access to this field is RW.

Otherwise:

Reserved, RES0.

COMP5[<m>] , bits [m+8], for m = 7 to 0

When UInt(TRCIDR4.NUMCIDC) > 5:

TRCCIDCVR5 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR5. Each bit in this field corresponds to a byte in TRCCIDCVR5.

COMP5[<m>]	Meaning
0b0	The trace unit includes TRCCIDCVR5[(m×8+7):(m×8)] when it performs the Context identifier comparison.
0b1	The trace unit ignores TRCCIDCVR5[(m×8+7):(m×8)] when it performs the Context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.CIDSIZE), access to this field is RES0.
- Otherwise, access to this field is RW.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

COMP4[<m>] , bits [m], for m = 7 to 0

When UInt(TRCIDR4.NUMCIDC) > 4:

TRCCIDCVR4 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR4. Each bit in this field corresponds to a byte in TRCCIDCVR4.

COMP4[<m>]	Meaning
0b0	The trace unit includes TRCCIDCVR4[(m×8+7):(m×8)] when it performs the Context identifier comparison.
0b1	The trace unit ignores TRCCIDCVR4[(m×8+7):(m×8)] when it performs the Context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.CIDSIZE), access to this field is RES0.
- Otherwise, access to this field is RW.

Otherwise:

Reserved, RES0.

Accessing TRCCIDCCTLR1

If software uses the TRCCIDCVR<n> registers, for n = 4 to 7, then it must program this register.

If software sets a mask bit to 1 then it must program the relevant byte in TRCCIDCVR<n> to 0x00.

If any bit is 1 and the relevant byte in TRCCIDCVR<n> is not 0x00, the behavior of the Context Identifier Comparator is CONSTRAINED UNPREDICTABLE. In this scenario the comparator might match unexpectedly or might not match.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCCIDCCTLR1
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0011	0b0001	0b010

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && UInt(TRCIDR4.NUMCIDC) > 0x4
    ↪&& UInt(TRCIDR2.CIDSIZE) > 0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCIDCCTLR1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCIDCCTLR1;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCIDCCTLR1;
```

```
MSR TRCCIDCCTLR1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0011	0b0001	0b010

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && UInt(TRCIDR4.NUMCIDC) > 0x4
     ↪&& UInt(TRCIDR2.CIDSIZE) > 0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCIDCCTLR1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCIDCCTLR1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCIDCCTLR1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.19 TRCCIDCVR<n>, Trace Context Identifier Comparator Value Registers <n>, n = 0 - 7

The TRCCIDCVR<n> characteristics are:

Purpose

Contains a Context identifier value.

Configuration

AArch64 System register TRCCIDCVR<n> bits [63:0] are architecturally mapped to External register TRCCIDCVR<n>[63:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and UInt(TRCIDR4.NUMCIDC) > n. Otherwise, direct accesses to TRCCIDCVR<n> are UNDEFINED.

Attributes

TRCCIDCVR<n> is a 64-bit register.

Field descriptions

VALUE, bits [63:0]

Context identifier value. The width of this field is indicated by TRCIDR2.CIDSIZE. Unimplemented bits are RES0. After a PE Reset, the trace unit assumes that the Context identifier is zero until the PE updates the Context identifier.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCCIDCVR<n>

Must be programmed if any of the following are true:

- TRCRSCTLR<a>.GROUP == `0b0110` and TRCRSCTLR<a>.CID[n] == 1.
- TRCACATR<a>.CONTEXTTYPE == `0b01` or `0b11` and TRCACATR<a>.CONTEXT == n.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCCIDCVR<m> ; Where m = 0-7
```

op0	op1	CRn	CRm	op2
`0b10`	`0b001`	`0b0011`	m[2:0]:`0b0`	`0b000`

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
integer m = UInt(CRm<3:1>);

if m >= NUM_TRACE_CONTEXT_IDENTIFIER_COMPARATORS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCIDCVR[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCIDCVR[m];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCIDCVR[m];
```

```
MSR TRCCIDCVR<m>, <Xt> ; Where m = 0-7
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0011	m[2:0]:0b0	0b000

```
integer m = UInt(CRm<3:1>);

if m >= NUM_TRACE_CONTEXT_IDENTIFIER_COMPARATORS then
    UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCIDCVR[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCIDCVR[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCIDCVR[m] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.20 TRCCLAIMCLR, Trace Claim Tag Clear Register

The TRCCLAIMCLR characteristics are:

Purpose

In conjunction with TRCCLAIMSET, provides Claim Tag bits that can be separately set and cleared to indicate whether functionality is in use by a debug agent.

For additional information, see the CoreSight Architecture Specification.

Configuration

AArch64 System register TRCCLAIMCLR bits [63:0] are architecturally mapped to AArch64 System register TRCCLAIMSET[63:0].

AArch64 System register TRCCLAIMCLR bits [31:0] are architecturally mapped to External register TRCCLAIMCLR[31:0].

AArch64 System register TRCCLAIMCLR bits [31:0] are architecturally mapped to External register TRCCLAIMSET[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCCLAIMCLR are UNDEFINED.

Attributes

TRCCLAIMCLR is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

CLR[<m>] , bits [m], for m = 31 to 0

Claim Tag Clear. Indicates the current status of Claim Tag bit <m>, and is used to clear Claim Tag bit <m> to 0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CLR[<m>]	Meaning
0b0	On a read: Claim Tag bit <m> is not set. On a write: Ignored.
0b1	On a read: Claim Tag bit <m> is set. On a write: Clear Claim tag bit <m> to 0.

The number of Claim Tag bits implemented is indicated in TRCCLAIMSET.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to '0'.

Accessing this field has the following behavior:

- When m >= NUM_TRC_CLAIM_TAGS, access to this field is RAZ/WI.
- Otherwise, access to this field is W1C.

Accessing TRCCLAIMCLR

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCCLAIMCLR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0111	0b1001	0b110

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRCCLAIM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCLAIMCLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCLAIMCLR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCLAIMCLR;
```

```
MSR TRCCLAIMCLR, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0111	0b1001	0b110

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRCCLAIM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCLAIMCLR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    else
        TRCCLAIMCLR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCLAIMCLR = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.21 TRCCLAIMSET, Trace Claim Tag Set Register

The TRCCLAIMSET characteristics are:

Purpose

In conjunction with TRCCLAIMCLR, provides Claim Tag bits that can be separately set and cleared to indicate whether functionality is in use by a debug agent.

For additional information, see the CoreSight Architecture Specification.

Configuration

The number of claim tag bits implemented is IMPLEMENTATION DEFINED. Arm recommends that implementations support a minimum of four claim tag bits, that is, SET[3:0] reads as `0b1111`.

AArch64 System register TRCCLAIMSET bits [63:0] are architecturally mapped to AArch64 System register TRCCLAIMCLR[63:0].

AArch64 System register TRCCLAIMSET bits [31:0] are architecturally mapped to External register TRCCLAIMSET[31:0].

AArch64 System register TRCCLAIMSET bits [31:0] are architecturally mapped to External register TRCCLAIMCLR[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCCLAIMSET are UNDEFINED.

Attributes

TRCCLAIMSET is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

SET[<m>] , bits [m], for m = 31 to 0

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Claim Tag Set. Indicates whether Claim Tag bit <m> is implemented, and is used to set Claim Tag bit <m> to 1.

SET[<m>]	Meaning
0b0	On a read: Claim Tag bit <m> is not implemented. On a write: Ignored.
0b1	On a read: Claim Tag bit <m> is implemented. On a write: Set Claim Tag bit <m> to 1.

Accessing this field has the following behavior:

- When m >= NUM_TRC_CLAIM_TAGS, access to this field is RAZ/WI.
- Otherwise, access to this field is RAO/W1S.

Accessing TRCCLAIMSET

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCCLAIMSET
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0111	0b1000	0b110

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRCCLAIM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCLAIMSET;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
             ↪== '1' then
            Halt(DebugHalt_SoftwareAccess);
        else
            X[t, 64] = TRCCLAIMSET;
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
             ↪== '1' then
            Halt(DebugHalt_SoftwareAccess);
        else
            X[t, 64] = TRCCLAIMSET;
```

```
MSR TRCCLAIMSET, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0111	0b1000	0b110

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRCCLAIM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCLAIMSET = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCLAIMSET = X[t, 64];
elsif PSTATE.EL == EL3 then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
     ↪== '1' then
    Halt(DebugHalt_SoftwareAccess);
else
    TRCCLAIMSET = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.22 TRCCNTCTLR<n>, Trace Counter Control Register <n>, n = 0 - 3

The TRCCNTCTLR<n> characteristics are:

Purpose

Controls the operation of Counter <n>.

Configuration

AArch64 System register TRCCNTCTLR<n> bits [31:0] are architecturally mapped to External register TRCCNTCTLR<n>[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and UInt(TRCIDR5.NUMCNTR) > n. Otherwise, direct accesses to TRCCNTCTLR<n> are UNDEFINED.

Attributes

TRCCNTCTLR<n> is a 64-bit register.

Field descriptions

Bits [63:18]

Reserved, RES0.

CNTCHAIN, bit [17]

When (n MOD 2) != 0:

For TRCCNTCTLR3 and TRCCNTCTLR1, this field controls whether the Counter decrements when a reload event occurs for Counter <n-1>.

CNTCHAIN	Meaning
0b0	The Counter does not decrement when a reload event for Counter <n-1> occurs.
0b1	Counter <n> decrements when a reload event for Counter <n-1> occurs. This concatenates Counter <n> and Counter <n-1>, to provide a larger count value.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

RLDSELF, bit [16]

Controls whether a reload event occurs for the Counter, when the Counter reaches zero.

RLDSELF	Meaning
0b0	Normal mode. The Counter is in Normal mode.
0b1	Self-reload mode. The Counter is in Self-reload mode.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

RLDEVENT_TYPE, bit [15]

Selects an event, that when it occurs causes a reload event for Counter

Chooses the type of Resource Selector.

RLDEVENT_TYPE	Meaning
0b0	A single Resource Selector. TRCCNTCTLR.RLDEVENT.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
0b1	A Boolean-combined pair of Resource Selectors. TRCCNTCTLR.RLDEVENT.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCCNTCTLR.RLDEVENT.SEL[4] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [14:13]

Reserved, RES0.

RLDEVENT_SEL, bits [12:8]

Selects an event, that when it occurs causes a reload event for Counter

Defines the selected Resource Selector or pair of Resource Selectors. TRCCNTCTLR.RLDEVENT.TYPE controls whether TRCCNTCTLR.RLDEVENT.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CNTEVENT_TYPE, bit [7]

Selects an event, that when it occurs causes Counter to decrement.

Chooses the type of Resource Selector.

CNTEVENT_TYPE	Meaning
0b0	A single Resource Selector. TRCCNTCTLR.CNTEVENT.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
0b1	A Boolean-combined pair of Resource Selectors. TRCCNTCTLR.CNTEVENT.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCCNTCTLR.CNTEVENT.SEL[4] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

CNTEVENT_SEL, bits [4:0]

Selects an event, that when it occurs causes Counter to decrement.

Defines the selected Resource Selector or pair of Resource Selectors. TRCCNTCTLR.CNTEVENT.TYPE controls whether TRCCNTCTLR.CNTEVENT.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCCNTCTLR<n>

Must be programmed if TRCRSCTLR<a>.GROUP == 0b0010 and TRCRSCTLR<a>.COUNTERS[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCCNTCTLR<m> ; Where m = 0-3
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b01:m[1:0]	0b101

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
integer m = UInt(CRm<1:0>);

if m >= NUM_TRACE_COUNTERS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCNTCTLR[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCNTCTLR[m];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCNTCTLR[m];
```

```
MSR TRCCNTCTLR<m>, <Xt> ; Where m = 0-3
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b01:m[1:0]	0b101

```
integer m = UInt(CRm<1:0>);

if m >= NUM_TRACE_COUNTERS then
    UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCNTCTLR[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCNTCTLR[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCNTCTLR[m] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.23 TRCCNTRLDVR<n>, Trace Counter Reload Value Register <n>, n = 0 - 3

The TRCCNTRLDVR<n> characteristics are:

Purpose

This sets or returns the reload count value for Counter <n>.

Configuration

AArch64 System register TRCCNTRLDVR<n> bits [31:0] are architecturally mapped to External register TRCCNTRLDVR<n>[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and UInt(TRCIDR5.NUMCNTR) > n. Otherwise, direct accesses to TRCCNTRLDVR<n> are UNDEFINED.

Attributes

TRCCNTRLDVR<n> is a 64-bit register.

Field descriptions

Bits [63:16]

Reserved, RES0.

VALUE, bits [15:0]

Contains the reload value for Counter <n>. When a reload event occurs for Counter <n> then the trace unit copies the VALUE<n> field into Counter <n>.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCCNTRLDVR<n>

Must be programmed if TRCRSCTLR<a>.GROUP == `0b0010` and TRCRSCTLR<a>.COUNTERS[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCCNTRLDVR<m> ; Where m = 0-3
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b00:m[1:0]	0b101

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
integer m = UInt(CRm<1:0>);

if m >= NUM_TRACE_COUNTERS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCNTRLDVR[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCNTRLDVR[m];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCNTRLDVR[m];
```

```
MSR TRCCNTRLDVR<m>, <Xt> ; Where m = 0-3
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b00:m[1:0]	0b101

```
integer m = UInt(CRm<1:0>);

if m >= NUM_TRACE_COUNTERS then
    UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCNTRLDVR[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCNTRLDVR[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCNTRLDVR[m] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.24 TRCCNTVR<n>, Trace Counter Value Register <n>, n = 0 - 3

The TRCCNTVR<n> characteristics are:

Purpose

This sets or returns the value of Counter <n>.

Configuration

AArch64 System register TRCCNTVR<n> bits [31:0] are architecturally mapped to External register TRCCNTVR<n>[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and UInt(TRCIDR5.NUMCNTR) > n. Otherwise, direct accesses to TRCCNTVR<n> are UNDEFINED.

Attributes

TRCCNTVR<n> is a 64-bit register.

Field descriptions

Bits [63:16]

Reserved, RES0.

VALUE, bits [15:0]

Contains the count value of Counter.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCCNTVR<n>

Must be programmed if TRCRSCTLR<a>.GROUP == `0b0010` and TRCRSCTLR<a>.COUNTERS[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCCNTVR<m> ; Where m = 0-3
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b10:m[1:0]	0b101

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
integer m = UInt(CRm<1:0>);

if m >= NUM_TRACE_COUNTERS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRCCNTVRn == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCNTVR[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCNTVR[m];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCNTVR[m];
```

```
MSR TRCCNTVR<m>, <Xt> ; Where m = 0-3
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b10:m[1:0]	0b101

```
integer m = UInt(CRm<1:0>);

if m >= NUM_TRACE_COUNTERS then
    UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRCCNTVRn == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCNTVR[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCNTVR[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCNTVR[m] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.25 TRCCONFIGR, Trace Configuration Register

The TRCCONFIGR characteristics are:

Purpose

Controls the tracing options.

Configuration

AArch64 System register TRCCONFIGR bits [31:0] are architecturally mapped to External register TRCCONFIGR[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCCONFIGR are UNDEFINED.

Attributes

TRCCONFIGR is a 64-bit register.

Field descriptions

Bits [63:19]

Reserved, RES0.

ITO, bit [18]

When TRCIDR0.ITE == '1':

Instrumentation Trace Override.

ITO	Meaning
0b0	Instrumentation Trace Override disabled.
0b1	Instrumentation Trace Override enabled.

This field is ignored when `SelfHostedTraceEnabled()` returns TRUE.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [17:16]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

VMIDOPT, bit [15]

When TRCIDR2.VMIDOPT == '01':

Virtual context identifier selection control.

VMIDOPT	Meaning
0b0	VTTBR_EL2.VMID is used as the Virtual context identifier.
0b1	CONTEXTIDR_EL2.PROCID is used as the Virtual context identifier.

When TRCIDR2.VMIDOPT == '00':

Reserved, RES0.

Virtual context identifier selection control.

VTTBR_EL2.VMID is used as the Virtual context identifier.

When TRCIDR2.VMIDOPT == '10':

Reserved, RES1.

Virtual context identifier selection control.

CONTEXTIDR_EL2.PROCID is used as the Virtual context identifier.

Otherwise:

Reserved, RES0.

QE, bits [14:13]

When TRCIDR0.QSUPP == '01':

Q element generation control.

QE	Meaning
0b00	Q elements are disabled.
0b01	Q elements with instruction counts are enabled. Q elements without instruction counts are disabled.

All other values are reserved.

When TRCIDR0.QSUPP == '10':

Q element generation control.

QE	Meaning
0b00	Q elements are disabled.
0b11	Q elements with instruction counts are enabled. Q elements without instruction counts are enabled.

All other values are reserved.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When TRCIDR0.QSUPP == '11':

Q element generation control.

QE	Meaning
`0b00`	Q elements are disabled.
`0b01`	Q elements with instruction counts are enabled. Q elements without instruction counts are disabled.
`0b11`	Q elements with instruction counts are enabled. Q elements without instruction counts are enabled.

All other values are reserved.

Otherwise:

Reserved, RES0.

RS, bit [12]

When TRCIDR0.RETSTACK == '1':

Return stack control.

RS	Meaning
`0b0`	Return stack is disabled.
`0b1`	Return stack is enabled.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TS, bit [11]

When TRCIDR0.TSSIZE != '00000':

Global timestamp tracing control.

TS	Meaning
`0b0`	Global timestamp tracing is disabled.
`0b1`	Global timestamp tracing is enabled.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

Bits [10:8]

Reserved, RES0.

VMID, bit [7]

When TRCIDR2.VMIDSIZE != '00000':

Virtual context identifier tracing control.

VMID	Meaning
0b0	Virtual context identifier tracing is disabled.
0b1	Virtual context identifier tracing is enabled.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CID, bit [6]

When TRCIDR2.CIDSIZE != '00000':

Context identifier tracing control.

CID	Meaning
0b0	Context identifier tracing is disabled.
0b1	Context identifier tracing is enabled.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [5]

Reserved, RES0.

CCI, bit [4]

When TRCIDR0.TRCCCI == '1':

Cycle counting instruction tracing control.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CCI	Meaning
0b0	Cycle counting instruction tracing is disabled.
0b1	Cycle counting instruction tracing is enabled.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BB, bit [3]

When TRCIDR0.TRCBB == '1':

Branch broadcasting control.

BB	Meaning
0b0	Branch broadcasting is disabled.
0b1	Branch broadcasting is enabled.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [2:1]

Reserved, RES0.

Bit [0]

Reserved, RES1.

Accessing TRCCONFIGR

Must always be programmed.

TRCCONFIGR.QE must be set to 0b00 if TRCCONFIGR.BB is not 0.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCCONFIGR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0100	0b000

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCONFIGR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCONFIGR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCCONFIGR;
```

```
MSR TRCCONFIGR, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCONFIGR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCONFIGR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCCONFIGR = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.26 TRCDEVARCH, Trace Device Architecture Register

The TRCDEVARCH characteristics are:

Purpose

Provides discovery information for the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

AArch64 System register TRCDEVARCH bits [31:0] are architecturally mapped to External register TRCDEVARCH[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCDEVARCH are UNDEFINED.

Attributes

TRCDEVARCH is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

ARCHITECT, bits [31:21]

Defines the architect of the component. For Trace, this is Arm Limited.

Bits [31:28] are the JEP106 continuation code, `0b0100`.

Bits [27:21] are the JEP106 identification code, `0b0111011`.

Reads as `0b01000111011`

Access to this field is RO.

PRESENT, bit [20]

DEVARCH present. Indicates that the TRCDEVARCH register is present.

Reads as `0b1`

Access to this field is RO.

REVISION, bits [19:16]

Revision. Defines the architecture revision of the component.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

REVISION	Meaning
0b0000	ETEv1.0, FEAT_ETE.
0b0001	ETEv1.1, FEAT_ETEv1p1.
0b0010	ETEv1.2, FEAT_ETEv1p2.
0b0011	ETEv1.3, FEAT_ETEv1p3.

All other values are reserved.

Access to this field is RO.

ARCHVER, bits [15:12]

Architecture Version. Defines the architecture version of the component.

ARCHVER	Meaning
0b0101	ETEv1.

ARCHVER and ARCHPART are also defined as a single field, ARCHID, so that ARCHVER is ARCHID[15:12].

This field reads as `0x5`.

Access to this field is RO.

ARCHPART, bits [11:0]

Architecture Part. Defines the architecture of the component.

ARCHPART	Meaning
0xA13	Arm PE trace architecture.

ARCHVER and ARCHPART are also defined as a single field, ARCHID, so that ARCHPART is ARCHID[11:0].

This field reads as `0xA13`.

Access to this field is RO.

Accessing TRCDEVARCH

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCDEVARCH
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0111	0b1111	0b110

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCDEVARCH;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCDEVARCH;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCDEVARCH;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.27 TRCDEVID, Trace Device Configuration Register

The TRCDEVID characteristics are:

Purpose

Provides discovery information for the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

AArch64 System register TRCDEVID bits [31:0] are architecturally mapped to External register TRCDEVID[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCDEVID are UNDEFINED.

Attributes

TRCDEVID is a 64-bit register.

Field descriptions

Bits [63:0]

Reserved, RES0.

Accessing TRCDEVID

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCDEVID
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0111	0b0010	0b111

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRCID == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCDEVID;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCDEVID;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCDEVID;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.28 TRCEVENTCTL0R, Trace Event Control 0 Register

The TRCEVENTCTL0R characteristics are:

Purpose

Controls the generation of ETEEvents.

Configuration

AArch64 System register TRCEVENTCTL0R bits [31:0] are architecturally mapped to External register TRCEVENTCTL0R[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and TRCIDR4.NUMRSPAIR != ‘0000’. Otherwise, direct accesses to TRCEVENTCTL0R are UNDEFINED.

Attributes

TRCEVENTCTL0R is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

EVENT3_TYPE, bit [31]

When TRCIDR4.NUMRSPAIR != '0000' and UInt(TRCIDR0.NUMEVENT) >= 3:

Chooses the type of Resource Selector.

EVENT3_TYPE	Meaning
0b0	A single Resource Selector. TRCEVENTCTL0R.EVENT3.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
0b1	A Boolean-combined pair of Resource Selectors. TRCEVENTCTL0R.EVENT3.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCEVENTCTL0R.EVENT3.SEL[4] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

Bits [30:29]

Reserved, RES0.

EVENT3_SEL, bits [28:24]

When TRCIDR4.NUMRSPAIR != '0000' and UInt(TRCIDR0.NUMEVENT) >= 3:

When any of the selected resource events occurs and TRCEVENTCTL1R.INSTEN[3] == 1, then Event element 3 is generated in the instruction trace element stream.

Defines the selected Resource Selector or pair of Resource Selectors. TRCEVENTCTL0R.EVENT3.TYPE controls whether TRCEVENTCTL0R.EVENT3.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EVENT2_TYPE, bit [23]

When TRCIDR4.NUMRSPAIR != '0000' and UInt(TRCIDR0.NUMEVENT) >= 2:

Chooses the type of Resource Selector.

EVENT2_TYPE	Meaning
`0b0`	A single Resource Selector. TRCEVENTCTL0R.EVENT2.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
`0b1`	A Boolean-combined pair of Resource Selectors. TRCEVENTCTL0R.EVENT2.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCEVENTCTL0R.EVENT2.SEL[4] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [22:21]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EVENT2_SEL, bits [20:16]

When TRCIDR4.NUMRSPAIR != '0000' and UInt(TRCIDR0.NUMEVENT) >= 2:

When any of the selected resource events occurs and TRCEVENTCTL1R.INSTEN[2] == 1, then Event element 2 is generated in the instruction trace element stream.

Defines the selected Resource Selector or pair of Resource Selectors. TRCEVENTCTL0R.EVENT2.TYPE controls whether TRCEVENTCTL0R.EVENT2.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EVENT1_TYPE, bit [15]

When TRCIDR4.NUMRSPAIR != '0000' and UInt(TRCIDR0.NUMEVENT) >= 1:

Chooses the type of Resource Selector.

EVENT1_TYPE	Meaning
0b0	A single Resource Selector. TRCEVENTCTL0R.EVENT1.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
0b1	A Boolean-combined pair of Resource Selectors. TRCEVENTCTL0R.EVENT1.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCEVENTCTL0R.EVENT1.SEL[4] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [14:13]

Reserved, RES0.

EVENT1_SEL, bits [12:8]

When TRCIDR4.NUMRSPAIR != '0000' and UInt(TRCIDR0.NUMEVENT) >= 1:

When any of the selected resource events occurs and TRCEVENTCTL1R.INSTEN[1] == 1, then Event element 1 is generated in the instruction trace element stream.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Defines the selected Resource Selector or pair of Resource Selectors. TRCEVENTCTL0R.EVENT1.TYPE controls whether TRCEVENTCTL0R.EVENT1.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EVENT0_TYPE, bit [7]

When TRCIDR4.NUMRSPAIR != '0000':

Chooses the type of Resource Selector.

EVENT0_TYPE	Meaning
0b0	A single Resource Selector. TRCEVENTCTL0R.EVENT0.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
0b1	A Boolean-combined pair of Resource Selectors. TRCEVENTCTL0R.EVENT0.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCEVENTCTL0R.EVENT0.SEL[4] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [6:5]

Reserved, RES0.

EVENT0_SEL, bits [4:0]

When TRCIDR4.NUMRSPAIR != '0000':

When any of the selected resource events occurs and TRCEVENTCTL1R.INSTEN[0] == 1, then Event element 0 is generated in the instruction trace element stream.

Defines the selected Resource Selector or pair of Resource Selectors. TRCEVENTCTL0R.EVENT0.TYPE controls whether TRCEVENTCTL0R.EVENT0.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TRCEVENTCTL0R

Must be programmed if implemented.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCEVENTCTL0R
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1000	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && TRCIDR4.NUMRSPAIR != '0000')
    then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
        == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCEVENTCTL0R;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
        ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCEVENTCTL0R;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
        ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCEVENTCTL0R;
```

```
MSR TRCEVENTCTL0R, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1000	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && TRCIDR4.NUMRSPAIR != '0000')
    ↪then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        ↪HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
        ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCEVENTCTL0R = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
        ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCEVENTCTL0R = X[t, 64];
elsif PSTATE.EL == EL3 then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
     ↪== '1' then
    Halt(DebugHalt_SoftwareAccess);
else
    TRCEVENTCTL0R = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.29 TRCEVENTCTL1R, Trace Event Control 1 Register

The TRCEVENTCTL1R characteristics are:

Purpose

Controls the behavior of the ETEEvents that TRCEVENTCTL0R selects.

Configuration

AArch64 System register TRCEVENTCTL1R bits [31:0] are architecturally mapped to External register TRCEVENTCTL1R[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCEVENTCTL1R are UNDEFINED.

Attributes

TRCEVENTCTL1R is a 64-bit register.

Field descriptions

Bits [63:14]

Reserved, RES0.

OE, bit [13]

When TRCIDR5.OE == '1':

ETE Trace Output Enable control.

OE	Meaning
0b0	Trace output to any IMPLEMENTATION DEFINED trace output interface is disabled.
0b1	Trace output to any IMPLEMENTATION DEFINED trace output interface is enabled.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to '0'.

Otherwise:

Reserved, RES0.

LPOVERRIDE, bit [12]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When TRCIDR5.LPOVERRIDE == '1':

Low-power Override Mode select.

LPOVERRIDE	Meaning
0b0	Trace unit Low-power Override Mode is not enabled. That is, the trace unit is permitted to enter low-power state.
0b1	Trace unit Low-power Override Mode is enabled. That is, entry to a low-power state does not affect the trace unit resources or trace generation.

Otherwise:

Reserved, RES0.

ATB, bit [11]

When TRCIDR5.ATBTRIG == '1':

AMBA Trace Bus (ATB) trigger enable.

If a CoreSight ATB interface is implemented then when ETEEvent 0 occurs the trace unit sets:

- ATID == 0x7D.
- ATDATA to the value of TRCTRACEIDR.

If the width of ATDATA is greater than the width of TRCTRACEIDR.TRACEID then the trace unit zeros the upper ATDATA bits.

If ETEEvent 0 is programmed to occur based on program execution, such as an Address Comparator, the ATB trigger might not be inserted into the ATB stream at the same time as any trace generated by that program execution is output by the trace unit. Typically, the generated trace might be buffered in a trace unit which means that the ATB trigger would be output before the associated trace is output.

If ETEEvent 0 is asserted multiple times in close succession, the trace unit is required to generate an ATB trigger for the first assertion, but might ignore one or more of the subsequent assertions. Arm recommends that the window in which ETEEvent 0 is ignored is limited only by the time taken to output an ATB trigger.

ATB	Meaning
0b0	ATB trigger is disabled.
0b1	ATB trigger is enabled.

Otherwise:

Reserved, RES0.

Bits [10:4]

Reserved, RES0.

INSTEN[<m>] , bits [m], for m = 3 to 0

Event element control.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

INSTEN[<m>]	Meaning
0b0	The trace unit does not generate an Event element <m>.
0b1	The trace unit generates an Event element <m> when ETEEvent <m> occurs.

Accessing this field has the following behavior:

- When TRCIDR4.NUMRSPAIR == ‘0000’, access to this field is RES0.
- Access to this field is RES0 if all of the following are true:
  - TRCIDR4.NUMRSPAIR != ‘0000’
  - m > UInt(TRCIDR0.NUMEVENT)
- Otherwise, access to this field is RW.

Accessing TRCEVENTCTL1R

Must be programmed.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCEVENTCTL1R
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1001	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCEVENTCTL1R;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCEVENTCTL1R;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCEVENTCTL1R;
```

```
MSR TRCEVENTCTL1R, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1001	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCEVENTCTL1R = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCEVENTCTL1R = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCEVENTCTL1R = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.30 TRCEXTINSELR<n>, Trace External Input Select Register <n>, n = 0 - 3

The TRCEXTINSELR<n> characteristics are:

Purpose

Use this to set, or read, which External Inputs are resources to the trace unit.

The name TRCEXTINSELR is an alias of TRCEXTINSELR0.

Configuration

AArch64 System register TRCEXTINSELR<n> bits [31:0] are architecturally mapped to External register TRCEXTINSELR<n>[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and UInt(TRCIDR5.NUMEXTINSEL) > n. Otherwise, direct accesses to TRCEXTINSELR<n> are UNDEFINED.

Attributes

TRCEXTINSELR<n> is a 64-bit register.

Field descriptions

Bits [63:16]

Reserved, RES0.

evtCount, bits [15:0]

PMU event to select.

The event number as defined by the Arm ARM.

Software must program this field with a PMU event that is supported by the PE being programmed.

There are three ranges of PMU event numbers:

- PMU event numbers in the range `0x0000` to `0x003F` are common architectural and microarchitectural events.
- PMU event numbers in the range `0x0040` to `0x00BF` are Arm recommended common architectural and microarchitectural PMU events.
- PMU event numbers in the range `0x00C0` to `0x03FF` are IMPLEMENTATION DEFINED PMU events.

If evtCount is programmed to a PMU event that is reserved or not supported by the PE, the behavior depends on the PMU event type:

- For the range `0x0000` to `0x003F`, then the PMU event is not active, and the value returned by a direct or external read of the evtCount field is the value written to the field.
- For IMPLEMENTATION DEFINED PMU events, it is UNPREDICTABLE what PMU event, if any, is counted, and the value returned by a direct or external read of the evtCount field is UNKNOWN.

UNPREDICTABLE means the PMU event must not expose privileged information.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Arm recommends that the behavior across a family of implementations is defined such that if a given implementation does not include a PMU event from a set of common IMPLEMENTATION DEFINED PMU events, then no PMU event is counted and the value read back on evtCount is the value written.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCEXTINSELR<n>

Must be programmed if any of the following is true: TRCRSCTLR<a>.GROUP == `0b0000` and TRCRSCTLR<a>.EXTIN[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCEXTINSELR<m> ; Where m = 0-3
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b10:m[1:0]	0b100

```
integer m = UInt(CRm<1:0>);

if m >= NUM_TRACE_EXTERNAL_INPUT_SELECTOR_RESOURCES then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCEXTINSELR[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        X[t, 64] = TRCEXTINSELR[m];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCEXTINSELR[m];
```

```
MSR TRCEXTINSELR<m>, <Xt> ; Where m = 0-3
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b10:m[1:0]	0b100

```
integer m = UInt(CRm<1:0>);

if m >= NUM_TRACE_EXTERNAL_INPUT_SELECTOR_RESOURCES then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCEXTINSELR[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCEXTINSELR[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
    ↪== '1' then
    Halt(DebugHalt_SoftwareAccess);
else
    TRCEXTINSELR[m] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.31 TRCIDR0, Trace ID Register 0

The TRCIDR0 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

AArch64 System register TRCIDR0 bits [31:0] are architecturally mapped to External register TRCIDR0[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCIDR0 are UNDEFINED.

Attributes

TRCIDR0 is a 64-bit register.

Field descriptions

Bits [63:31]

Reserved, RES0.

COMMTRANS, bit [30]

Transaction Start element behavior.

The value of this field is an IMPLEMENTATION DEFINED choice of:

COMMTRANS	Meaning
0b0	Transaction Start elements are P0 elements.
0b1	Transaction Start elements are not P0 elements.

Access to this field is RO.

COMMOPT, bit [29]

Indicates the contents and encodings of Cycle count packets.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

COMMOPT	Meaning
0b0	Commit mode 0.
0b1	Commit mode 1.

The Commit mode defines the contents and encodings of Cycle Count packets, in particular how Commit elements are indicated by these packets. See the descriptions of these packets for more details.

Accessing this field has the following behavior:

- Access to this field is RAO/WI if all of the following are true:
  - TRCIDR0.TRCCCI == '1'
  - UInt(TRCIDR8.MAXSPEC) == 0x0
- When TRCIDR0.TRCCCI == '0', access to this field is RAZ/WI.
- Otherwise, access to this field is RO.

TSSIZE, bits [28:24]

Indicates that the trace unit implements Global timestamping and the size of the timestamp value.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TSSIZE	Meaning
0b00000	Global timestamping not implemented.
0b01000	Global timestamping implemented with a 64-bit timestamp value.

All other values are reserved.

This field reads as 0b01000.

Access to this field is RO.

TSMARK, bit [23]

Indicates whether Timestamp Marker elements are generated.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TSMARK	Meaning
0b0	Timestamp Marker elements are not generated.
0b1	Timestamp Marker elements are generated.

Access to this field is RO.

ITE, bit [22]

Indicates whether Instrumentation Trace is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ITE	Meaning
0b0	Instrumentation Trace not implemented.
0b1	Instrumentation Trace implemented.

This field has the value 1 if FEAT_ITE is implemented.

Access to this field is RO.

Bits [21:18]

Reserved, RES0.

TRCEXDATA, bit [17]

When TRCIDR0.TRCDATA != '00':

Indicates if the trace unit implements tracing of data transfers for exceptions and exception returns. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TRCEXDATA	Meaning
0b0	Tracing of data transfers for exceptions and exception returns not implemented.
0b1	Tracing of data transfers for exceptions and exception returns implemented.

Access to this field is RO.

Otherwise:

Reserved, RES0.

QSUPP, bits [16:15]

Indicates that the trace unit implements Q element support.

The value of this field is an IMPLEMENTATION DEFINED choice of:

QSUPP	Meaning
0b00	Q element support is not implemented.
0b01	Q element support is implemented, and only supports Q elements with instruction counts.
0b10	Q element support is implemented, and only supports Q elements without instruction counts.
0b11	Q element support is implemented, and supports: • Q elements with instruction counts. • Q elements without instruction counts.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Access to this field is RO.

QFILT, bit [14]

Indicates if the trace unit implements Q element filtering.

The value of this field is an IMPLEMENTATION DEFINED choice of:

QFILT	Meaning
`0b0`	Q element filtering is not implemented.
`0b1`	Q element filtering is implemented.

If TRCIDR0.QSUPP == `0b00` then this field is 0.

Access to this field is RO.

CONDTYPE, bits [13:12]

When TRCIDR0.TRCCOND == '1':

Indicates how conditional instructions are traced. Conditional instruction tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CONDTYPE	Meaning
`0b00`	Conditional instructions are traced with an indication of whether they pass or fail their condition code check.
`0b01`	Conditional instructions are traced with an indication of the APSR condition flags.

All other values are reserved.

Access to this field is RO.

Otherwise:

Reserved, RES0.

NUMEVENT, bits [11:10]

When TRCIDR4.NUMRSPAIR == '0000':

Indicates the number of ETEEvents implemented.

NUMEVENT	Meaning
`0b00`	The trace unit supports 0 ETEEvents.

All other values are reserved.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When TRCIDR4.NUMRSPAIR != '0000':

Indicates the number of ETEEvents implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMEVENT	Meaning
0b00	The trace unit supports 1 ETEEvent.
0b01	The trace unit supports 2 ETEEvents.
0b10	The trace unit supports 3 ETEEvents.
0b11	The trace unit supports 4 ETEEvents.

Access to this field is RO.

Otherwise:

Reserved, RES0.

RETSTACK, bit [9]

Indicates if the trace unit supports the return stack.

The value of this field is an IMPLEMENTATION DEFINED choice of:

RETSTACK	Meaning
0b0	Return stack not implemented.
0b1	Return stack implemented.

Access to this field is RO.

Bit [8]

Reserved, RES0.

TRCCCI, bit [7]

Indicates if the trace unit implements cycle counting.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TRCCCI	Meaning
0b0	Cycle counting not implemented.
0b1	Cycle counting implemented.

This field reads as 1.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TRCCOND, bit [6]

Indicates if the trace unit implements conditional instruction tracing. Conditional instruction tracing is not implemented in ETE and this field is reserved for other trace architectures.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TRCCOND	Meaning
`0b0`	Conditional instruction tracing not implemented.
`0b1`	Conditional instruction tracing implemented.

This field reads as 0.

Access to this field is RO.

TRCBB, bit [5]

Indicates if the trace unit implements branch broadcasting.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TRCBB	Meaning
`0b0`	Branch broadcasting not implemented.
`0b1`	Branch broadcasting implemented.

This field reads as 1.

Access to this field is RO.

TRCDATA, bits [4:3]

Indicates if the trace unit implements data tracing. Data tracing is not implemented in ETE and this field is reserved for other trace architectures.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TRCDATA	Meaning
`0b00`	Data tracing not implemented.
`0b11`	Data tracing implemented.

All other values are reserved.

This field reads as `0b00`.

Access to this field is RO.

INSTP0, bits [2:1]

Indicates if load and store instructions are P0 instructions. Load and store instructions as P0 instructions is not implemented in ETE and this field is reserved for other trace architectures.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The value of this field is an IMPLEMENTATION DEFINED choice of:

INSTP0	Meaning
0b00	Load and store instructions are not P0 instructions.
0b11	Load and store instructions are P0 instructions.

All other values are reserved.

When FEAT_ETE is implemented, the only permitted value is 0b00.

Access to this field is RO.

Bit [0]

Reserved, RES1.

Accessing TRCIDR0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCIDR0
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1000	0b111

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR0;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR0;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.32 TRCIDR1, Trace ID Register 1

The TRCIDR1 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

AArch64 System register TRCIDR1 bits [31:0] are architecturally mapped to External register TRCIDR1[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCIDR1 are UNDEFINED.

Attributes

TRCIDR1 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

DESIGNER, bits [31:24]

Indicates which company designed the trace unit. The permitted values of this field are the same as MIDR_EL1.Implementer.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Bits [23:16]

Reserved, RES0.

Bits [15:12]

Reserved, RES1.

TRCARCHMAJ, bits [11:8]

Major architecture version.

TRCARCHMAJ	Meaning
`0b1111`	If both TRCIDR1.TRCARCHMAJ and TRCIDR1.TRCARCHMIN == `0xF` then refer to TRCDEVARCH.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

All other values are reserved.

This field reads as `0b1111`.

Access to this field is RO.

TRCARCHMIN, bits [7:4]

Minor architecture version.

TRCARCHMIN	Meaning
`0b1111`	If both TRCIDR1.TRCARCHMAJ and TRCIDR1.TRCARCHMIN == `0xF` then refer to TRCDEVARCH.

All other values are reserved.

This field reads as `0b1111`.

Access to this field is RO.

REVISION, bits [3:0]

Implementation revision.

Returns an IMPLEMENTATION DEFINED value that identifies the revision of the trace unit.

Arm deprecates any use of this field and recommends that implementations set this field to zero.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Accessing TRCIDR1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCIDR1
```

op0	op1	CRn	CRm	op2
`0b10`	`0b001`	`0b0000`	`0b1001`	`0b111`

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR1;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.33 TRCIDR10, Trace ID Register 10

The TRCIDR10 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

AArch64 System register TRCIDR10 bits [31:0] are architecturally mapped to External register TRCIDR10[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCIDR10 are UNDEFINED.

Attributes

TRCIDR10 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

NUMP1KEY, bits [31:0]

When TRCIDR0.TRCDATA != '00':

Indicates the number of P1 right-hand keys. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Otherwise:

Reserved, RES0.

Accessing TRCIDR10

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCIDR10
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0010	0b110

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR10;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR10;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR10;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.34 TRCIDR11, Trace ID Register 11

The TRCIDR11 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

AArch64 System register TRCIDR11 bits [31:0] are architecturally mapped to External register TRCIDR11[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCIDR11 are UNDEFINED.

Attributes

TRCIDR11 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

NUMP1SPC, bits [31:0]

When TRCIDR0.TRCDATA != '00':

Indicates the number of special P1 right-hand keys. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Otherwise:

Reserved, RES0.

Accessing TRCIDR11

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCIDR11
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0011	0b110

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR11;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR11;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR11;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.35 TRCIDR12, Trace ID Register 12

The TRCIDR12 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

AArch64 System register TRCIDR12 bits [31:0] are architecturally mapped to External register TRCIDR12[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCIDR12 are UNDEFINED.

Attributes

TRCIDR12 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

NUMCONDKEY, bits [31:0]

When TRCIDR0.TRCCOND == '1':

Indicates the number of conditional instruction right-hand keys. Conditional instruction tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Otherwise:

Reserved, RES0.

Accessing TRCIDR12

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCIDR12
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0100	0b110

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR12;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR12;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR12;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.36 TRCIDR13, Trace ID Register 13

The TRCIDR13 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

AArch64 System register TRCIDR13 bits [31:0] are architecturally mapped to External register TRCIDR13[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCIDR13 are UNDEFINED.

Attributes

TRCIDR13 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

NUMCONDSPC, bits [31:0]

When TRCIDR0.TRCCOND == '1':

Indicates the number of special conditional instruction right-hand keys. Conditional instruction tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Otherwise:

Reserved, RES0.

Accessing TRCIDR13

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCIDR13
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0101	0b110

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR13;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR13;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR13;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.37 TRCIDR2, Trace ID Register 2

The TRCIDR2 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

AArch64 System register TRCIDR2 bits [31:0] are architecturally mapped to External register TRCIDR2[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCIDR2 are UNDEFINED.

Attributes

TRCIDR2 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

WFXMODE, bit [31]

Indicates whether `WFI`, `WFIT`, `WFE`, and `WFET` instructions are classified as P0 instructions:

The value of this field is an IMPLEMENTATION DEFINED choice of:

WFXMODE	Meaning
0b0	`WFI`, `WFIT`, `WFE`, and `WFET` instructions are not classified as P0 instructions.
0b1	`WFI`, `WFIT`, `WFE`, and `WFET` instructions are classified as P0 instructions.

Access to this field is RO.

VMIDOPT, bits [30:29]

Indicates the options for Virtual context identifier selection.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

VMIDOPT	Meaning
0b00	Virtual context identifier selection not supported. TRCCONFIGR.VMIDOPT is RES0.
0b01	Virtual context identifier selection supported. TRCCONFIGR.VMIDOPT is implemented.
0b10	Virtual context identifier selection not supported. TRCCONFIGR.VMIDOPT is RES1.

All other values are reserved.

If TRCIDR2.VMIDSIZE == 0b00000 then this field is 0b00.

If TRCIDR2.VMIDSIZE != 0b00000 then this field is 0b10.

Access to this field is RO.

CCSIZE, bits [28:25]

When TRCIDR0.TRCCCI == '1':

Indicates the size of the cycle counter.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CCSIZE	Meaning
0b0000	The cycle counter is 12 bits in length.
0b0001	The cycle counter is 13 bits in length.
0b0010	The cycle counter is 14 bits in length.
0b0011	The cycle counter is 15 bits in length.
0b0100	The cycle counter is 16 bits in length.
0b0101	The cycle counter is 17 bits in length.
0b0110	The cycle counter is 18 bits in length.
0b0111	The cycle counter is 19 bits in length.
0b1000	The cycle counter is 20 bits in length.

All other values are reserved.

Access to this field is RO.

Otherwise:

Reserved, RES0.

DVSIZE, bits [24:20]

When TRCIDR0.TRCDATA != '00':

Indicates the data value size in bytes. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

DVSIZE	Meaning
0b00000	Data value tracing not implemented.
0b00100	Data value tracing has a maximum of 32-bit data values.
0b01000	Data value tracing has a maximum of 64-bit data values.

All other values are reserved.

Access to this field is RO.

Otherwise:

Reserved, RES0.

DASIZE, bits [19:15]

When TRCIDR0.TRCDATA != '00':

Indicates the data address size in bytes. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

The value of this field is an IMPLEMENTATION DEFINED choice of:

DASIZE	Meaning
0b00000	Data address tracing not implemented.
0b00100	Data address tracing has a maximum of 32-bit data addresses.
0b01000	Data address tracing has a maximum of 64-bit data addresses.

All other values are reserved.

Access to this field is RO.

Otherwise:

Reserved, RES0.

VMIDSIZE, bits [14:10]

Indicates the trace unit Virtual context identifier size.

The value of this field is an IMPLEMENTATION DEFINED choice of:

VMIDSIZE	Meaning
0b00000	Virtual context identifier tracing is not supported.
0b00001	8-bit Virtual context identifier size.
0b00010	16-bit Virtual context identifier size.
0b00100	32-bit Virtual context identifier size.

All other values are reserved.

If the PE does not implement EL2 then this field is 0b00000.

If the PE implements EL2 then this field is 0b00100.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Access to this field is RO.

CIDSIZE, bits [9:5]

Indicates the Context identifier size.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CIDSIZE	Meaning
0b00000	Context identifier tracing is not supported.
0b00100	32-bit Context identifier size.

All other values are reserved.

This field reads as 0b00100.

Access to this field is RO.

IASIZE, bits [4:0]

Virtual instruction address size.

The value of this field is an IMPLEMENTATION DEFINED choice of:

IASIZE	Meaning
0b00100	Maximum of 32-bit instruction address size.
0b01000	Maximum of 64-bit instruction address size.

All other values are reserved.

This field reads as 0b01000.

Access to this field is RO.

Accessing TRCIDR2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCIDR2
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1010	0b111

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR2;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR2;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR2;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.38 TRCIDR3, Trace ID Register 3

The TRCIDR3 characteristics are:

Purpose

Returns the base architecture of the trace unit.

Configuration

AArch64 System register TRCIDR3 bits [31:0] are architecturally mapped to External register TRCIDR3[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCIDR3 are UNDEFINED.

Attributes

TRCIDR3 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

NOOVERFLOW, bit [31]

Indicates if overflow prevention is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NOOVERFLOW	Meaning
0b0	Overflow prevention is not implemented.
0b1	Overflow prevention is implemented.

If TRCIDR3.STALLCTL == 0 then this field is 0.

Access to this field is RO.

NUMPROC, bits [13:12, 30:28]

Indicates the number of PEs available for tracing.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

NUMPROC	Meaning
0b00000	The trace unit can trace one PE.

This field reads as 0b00000.

Access to this field is RO.

SYSSTALL, bit [27]

Indicates if stalling of the PE is permitted.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SYSSTALL	Meaning
0b0	Stalling of the PE is not permitted.
0b1	Stalling of the PE is permitted.

The value of this field might be dynamic and change based on system conditions.

If TRCIDR3.STALLCTL == 0 then this field is 0.

Access to this field is RO.

STALLCTL, bit [26]

Indicates if trace unit implements stalling of the PE.

The value of this field is an IMPLEMENTATION DEFINED choice of:

STALLCTL	Meaning
0b0	Stalling of the PE is not implemented.
0b1	Stalling of the PE is implemented.

Access to this field is RO.

SYNCPR, bit [25]

Indicates if an implementation has a fixed synchronization period.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SYNCPR	Meaning
0b0	TRCSYNCPR is read/write so software can change the synchronization period.
0b1	TRCSYNCPR is read-only so the synchronization period is fixed.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

This field reads as 0.

Access to this field is RO.

TRCERR, bit [24]

Indicates forced tracing of System Error exceptions is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TRCERR	Meaning
0b0	Forced tracing of System Error exceptions is not implemented.
0b1	Forced tracing of System Error exceptions is implemented.

This field reads as 1.

Access to this field is RO.

Bit [23]

Reserved, RES0.

EXLEVEL_NS_EL2, bit [22]

Indicates if Non-secure EL2 is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EXLEVEL_NS_EL2	Meaning
0b0	Non-secure EL2 is not implemented.
0b1	Non-secure EL2 is implemented.

Access to this field is RO.

EXLEVEL_NS_EL1, bit [21]

Indicates if Non-secure EL1 is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EXLEVEL_NS_EL1	Meaning
0b0	Non-secure EL1 is not implemented.
0b1	Non-secure EL1 is implemented.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EXLEVEL_NS_EL0, bit [20]

Indicates if Non-secure EL0 is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EXLEVEL_NS_EL0	Meaning
0b0	Non-secure EL0 is not implemented.
0b1	Non-secure EL0 is implemented.

Access to this field is RO.

EXLEVEL_S_EL3, bit [19]

Indicates if EL3 is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EXLEVEL_S_EL3	Meaning
0b0	EL3 is not implemented.
0b1	EL3 is implemented.

Access to this field is RO.

EXLEVEL_S_EL2, bit [18]

Indicates if Secure EL2 is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EXLEVEL_S_EL2	Meaning
0b0	Secure EL2 is not implemented.
0b1	Secure EL2 is implemented.

Access to this field is RO.

EXLEVEL_S_EL1, bit [17]

Indicates if Secure EL1 is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EXLEVEL_S_EL1	Meaning
0b0	Secure EL1 is not implemented.
0b1	Secure EL1 is implemented.

Access to this field is RO.

EXLEVEL_S_EL0, bit [16]

Indicates if Secure EL0 is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EXLEVEL_S_EL0	Meaning
0b0	Secure EL0 is not implemented.
0b1	Secure EL0 is implemented.

Access to this field is RO.

Bits [15:14]

Reserved, RES0.

CCITMIN, bits [11:0]

When TRCIDR0.TRCCCI == '0':

Indicates the minimum value that can be programmed in TRCCCCTLR.THRESHOLD.

Reads as 0x000

Access to this field is RO.

When TRCIDR0.TRCCCI == '1':

Indicates the minimum value that can be programmed in TRCCCCTLR.THRESHOLD.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CCITMIN	Meaning
0x001..0xFFF	The minimum value that can be programmed in TRCCCCTLR.THRESHOLD.

The minimum value of this field is 0x001.

Access to this field is RO.

Otherwise:

Reserved, RES0.

Accessing TRCIDR3

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, TRCIDR3
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1011	0b111

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR3;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR3;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR3;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.39 TRCIDR4, Trace ID Register 4

The TRCIDR4 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

AArch64 System register TRCIDR4 bits [31:0] are architecturally mapped to External register TRCIDR4[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCIDR4 are UNDEFINED.

Attributes

TRCIDR4 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

NUMVMIDC, bits [31:28]

Indicates the number of Virtual Context Identifier Comparators that are available for tracing.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMVMIDC	Meaning
0b0000..0b1000	The number of Virtual Context Identifier Comparators in this implementation.

All other values are reserved.

If the PE does not implement EL2 then this field is 0b0000.

Access to this field is RO.

NUMCIDC, bits [27:24]

Indicates the number of Context Identifier Comparators that are available for tracing.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

NUMCIDC	Meaning
0b0000..0b1000	The number of Context Identifier Comparators in this implementation.

All other values are reserved.

Access to this field is RO.

NUMSSCC, bits [23:20]

Indicates the number of Single-shot Comparator Controls that are available for tracing.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMSSCC	Meaning
0b0000..0b1000	The number of Single-shot Comparator Controls in this implementation.

All other values are reserved.

Access to this field is RO.

NUMRSPAIR, bits [19:16]

Indicates the number of resource selector pairs that are available for tracing.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMRSPAIR	Meaning
0b0000	This implementation has zero resource selector pairs.
0b0001..0b1111	The number of resource selector pairs in this implementation, minus one.

All other values are reserved.

Access to this field is RO.

NUMPC, bits [15:12]

Indicates the number of PE Comparator Inputs that are available for tracing.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMPC	Meaning
0b0000..0b1000	The number of PE Comparator Inputs in this implementation.

All other values are reserved.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bits [11:9]

Reserved, RES0.

SUPPDAC, bit [8]

When TRCIDR4.NUMACPAIRS != '0000':

Indicates whether data address comparisons are implemented. Data address comparisons are not implemented in ETE and are reserved for other trace architectures. Allocated in other trace architectures.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SUPPDAC	Meaning
`0b0`	Data address comparisons not implemented.
`0b1`	Data address comparisons implemented.

This field reads as `0b0`.

Access to this field is RO.

Otherwise:

Reserved, RES0.

NUMDVC, bits [7:4]

Indicates the number of data value comparators. Data value comparators are not implemented in ETE and are reserved for other trace architectures. Allocated in other trace architectures.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMDVC	Meaning
`0b0000..0b1000`	The number of data value comparators in this implementation.

All other values are reserved.

This field reads as `0b0000`.

Access to this field is RO.

NUMACPAIRS, bits [3:0]

Indicates the number of Address Comparator pairs that are available for tracing.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMACPAIRS	Meaning
`0b0000..0b1000`	The number of Address Comparator pairs in this implementation.

All other values are reserved.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing TRCIDR4

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCIDR4
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1100	0b111

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR4;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR4;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR4;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.40 TRCIDR5, Trace ID Register 5

The TRCIDR5 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

AArch64 System register TRCIDR5 bits [31:0] are architecturally mapped to External register TRCIDR5[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCIDR5 are UNDEFINED.

Attributes

TRCIDR5 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

OE, bit [31]

Indicates support for the ETE Trace Output Enable.

The value of this field is an IMPLEMENTATION DEFINED choice of:

OE	Meaning
0b0	ETE Trace Output Enable is not implemented.
0b1	ETE Trace Output Enable is implemented.

When FEAT_ETEv1p3 is implemented and when any IMPLEMENTATION DEFINED trace output interface is implemented, this field is 1.

Access to this field is RO.

NUMCNTR, bits [30:28]

Indicates the number of Counters that are available for tracing.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

NUMCNTR	Meaning
0b000..0b100	The number of Counters implemented.

All other values are reserved.

If TRCIDR4.NUMRSPAIR == 0b0000 then this field is 0b000.

Access to this field is RO.

NUMSEQSTATE, bits [27:25]

Indicates if the Sequencer is implemented and the number of Sequencer states that are implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMSEQSTATE	Meaning
0b000	The Sequencer is not implemented.
0b100	Four Sequencer states are implemented.

All other values are reserved.

If TRCIDR4.NUMRSPAIR == 0b0000 then this field is 0b000.

Access to this field is RO.

Bit [24]

Reserved, RES0.

LPOVERRIDE, bit [23]

Indicates support for Low-power Override Mode.

The value of this field is an IMPLEMENTATION DEFINED choice of:

LPOVERRIDE	Meaning
0b0	The trace unit does not support Low-power Override Mode.
0b1	The trace unit supports Low-power Override Mode.

Access to this field is RO.

ATBTRIG, bit [22]

Indicates if the implementation can support ATB triggers.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ATBTRIG	Meaning
0b0	The implementation does not support ATB triggers.
0b1	The implementation supports ATB triggers.

If TRCIDR4.NUMRSPAIR == 0b0000 then this field is 0.

Access to this field is RO.

TRACEIDSIZE, bits [21:16]

Indicates the trace ID width.

The value of this field is an IMPLEMENTATION DEFINED choice of:

TRACEIDSIZE	Meaning
0b000000	The external trace interface is not implemented.
0b000111	The implementation supports a 7-bit trace ID.

All other values are reserved.

Note

AMBA ATB requires a 7-bit trace ID width.

Access to this field is RO.

Bits [15:12]

Reserved, RES0.

NUMEXTINSEL, bits [11:9]

Indicates how many External Input Selector resources are implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMEXTINSEL	Meaning
0b000..0b100	The number of External Input Selector resources implemented.

All other values are reserved.

Access to this field is RO.

NUMEXTIN, bits [8:0]

Indicates how many External Inputs are implemented.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

NUMEXTIN	Meaning
0b111111111	Unified PMU event selection.

All other values are reserved.

Access to this field is RO.

Accessing TRCIDR5

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCIDR5
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1101	0b111

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR5;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR5;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
     ↪== '1' then
    Halt(DebugHalt_SoftwareAccess);
else
    X[t, 64] = TRCIDR5;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.41 TRCIDR6, Trace ID Register 6

The TRCIDR6 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

AArch64 System register TRCIDR6 bits [31:0] are architecturally mapped to External register TRCIDR6[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCIDR6 are UNDEFINED.

Attributes

TRCIDR6 is a 64-bit register.

Field descriptions

Bits [63:3]

Reserved, RES0.

EXLEVEL_RL_EL2, bit [2]

Indicates if Realm EL2 is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EXLEVEL_RL_EL2	Meaning
0b0	Realm EL2 is not implemented.
0b1	Realm EL2 is implemented.

Access to this field is RO.

EXLEVEL_RL_EL1, bit [1]

Indicates if Realm EL1 is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EXLEVEL_RL_EL1	Meaning
0b0	Realm EL1 is not implemented.
0b1	Realm EL1 is implemented.

Access to this field is RO.

EXLEVEL_RL_EL0, bit [0]

Indicates if Realm EL0 is implemented.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EXLEVEL_RL_EL0	Meaning
0b0	Realm EL0 is not implemented.
0b1	Realm EL0 is implemented.

Access to this field is RO.

Accessing TRCIDR6

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCIDR6
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1110	0b111

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR6;
elsif PSTATE.EL == EL2 then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR6;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR6;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.42 TRCIDR7, Trace ID Register 7

The TRCIDR7 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

AArch64 System register TRCIDR7 bits [31:0] are architecturally mapped to External register TRCIDR7[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCIDR7 are UNDEFINED.

Attributes

TRCIDR7 is a 64-bit register.

Field descriptions

Bits [63:0]

Reserved, RES0.

Accessing TRCIDR7

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCIDR7
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1111	0b111

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
     ↪HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR7;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR7;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR7;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.43 TRCIDR8, Trace ID Register 8

The TRCIDR8 characteristics are:

Purpose

Returns the maximum speculation depth of the instruction trace element stream.

Configuration

AArch64 System register TRCIDR8 bits [31:0] are architecturally mapped to External register TRCIDR8[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCIDR8 are UNDEFINED.

Attributes

TRCIDR8 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

MAXSPEC, bits [31:0]

Indicates the maximum speculation depth of the instruction trace element stream. This is the maximum number of P0 elements in the trace element stream that can be speculative at any time.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Accessing TRCIDR8

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCIDR8
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0000	0b110

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR8;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR8;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR8;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.44 TRCIDR9, Trace ID Register 9

The TRCIDR9 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

AArch64 System register TRCIDR9 bits [31:0] are architecturally mapped to External register TRCIDR9[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCIDR9 are UNDEFINED.

Attributes

TRCIDR9 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

NUMP0KEY, bits [31:0]

When TRCIDR0.TRCDATA != '00':

Indicates the number of P0 right-hand keys. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Otherwise:

Reserved, RES0.

Accessing TRCIDR9

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCIDR9
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0001	0b110

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR9;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR9;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIDR9;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.45 TRCIMSPEC0, Trace IMP DEF Register 0

The TRCIMSPEC0 characteristics are:

Purpose

TRCIMSPEC0 shows the presence of any IMPLEMENTATION DEFINED features, and provides an interface to enable the features that are provided.

Configuration

AArch64 System register TRCIMSPEC0 bits [31:0] are architecturally mapped to External register TRCIMSPEC0[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCIMSPEC0 are UNDEFINED.

Attributes

TRCIMSPEC0 is a 64-bit register.

Field descriptions

Bits [63:8]

Reserved, RES0.

EN, bits [7:4]

When TRCIMSPEC0.SUPPORT != '0000':

Enable. Controls whether the IMPLEMENTATION DEFINED features are enabled.

EN	Meaning
0b0000	The IMPLEMENTATION DEFINED features are not enabled. The trace unit must behave as if the IMPLEMENTATION DEFINED features are not supported.
0b0001	The trace unit behavior is IMPLEMENTATION DEFINED.
0b0010	The trace unit behavior is IMPLEMENTATION DEFINED.
0b0011	The trace unit behavior is IMPLEMENTATION DEFINED.
0b0100	The trace unit behavior is IMPLEMENTATION DEFINED.
0b0101	The trace unit behavior is IMPLEMENTATION DEFINED.
0b0110	The trace unit behavior is IMPLEMENTATION DEFINED.
0b0111	The trace unit behavior is IMPLEMENTATION DEFINED.
0b1000	The trace unit behavior is IMPLEMENTATION DEFINED.
0b1001	The trace unit behavior is IMPLEMENTATION DEFINED.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EN	Meaning
0b1010	The trace unit behavior is IMPLEMENTATION DEFINED.
0b1011	The trace unit behavior is IMPLEMENTATION DEFINED.
0b1100	The trace unit behavior is IMPLEMENTATION DEFINED.
0b1101	The trace unit behavior is IMPLEMENTATION DEFINED.
0b1110	The trace unit behavior is IMPLEMENTATION DEFINED.
0b1111	The trace unit behavior is IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to `'0000'`.

Otherwise:

Reserved, RES0.

SUPPORT, bits [3:0]

Indicates whether the implementation supports IMPLEMENTATION DEFINED features.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SUPPORT	Meaning
0b0000	No IMPLEMENTATION DEFINED features are supported.
0b0001	IMPLEMENTATION DEFINED features are supported.
0b0010	IMPLEMENTATION DEFINED features are supported.
0b0011	IMPLEMENTATION DEFINED features are supported.
0b0100	IMPLEMENTATION DEFINED features are supported.
0b0101	IMPLEMENTATION DEFINED features are supported.
0b0110	IMPLEMENTATION DEFINED features are supported.
0b0111	IMPLEMENTATION DEFINED features are supported.
0b1000	IMPLEMENTATION DEFINED features are supported.
0b1001	IMPLEMENTATION DEFINED features are supported.
0b1010	IMPLEMENTATION DEFINED features are supported.
0b1011	IMPLEMENTATION DEFINED features are supported.
0b1100	IMPLEMENTATION DEFINED features are supported.
0b1101	IMPLEMENTATION DEFINED features are supported.
0b1110	IMPLEMENTATION DEFINED features are supported.
0b1111	IMPLEMENTATION DEFINED features are supported.

Use of nonzero values requires written permission from Arm.

Access to this field is RO.

Accessing TRCIMSPEC0

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, TRCIMSPEC0
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0000	0b111

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRCIMSPECn == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIMSPEC0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIMSPEC0;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIMSPEC0;
```

```
MSR TRCIMSPEC0, <Xt>
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0000	0b111

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRCIMSPECn == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCIMSPEC0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCIMSPEC0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCIMSPEC0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.46 TRCIMSPEC<n>, Trace IMP DEF Register <n>, n = 1 - 7

The TRCIMSPEC<n> characteristics are:

Purpose

These registers might return information that is specific to an implementation, or enable features specific to an implementation to be programmed. The product Technical Reference Manual describes these registers.

Configuration

AArch64 System register TRCIMSPEC<n> bits [31:0] are architecturally mapped to External register TRCIMSPEC<n>[31:0].

This register is present only when an implementation implements TRCIMSPEC<n>, FEAT_ETE is implemented, and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCIMSPEC<n> are UNDEFINED.

Attributes

TRCIMSPEC<n> is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

Accessing TRCIMSPEC<n>

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCIMSPEC<m> ; Where m = 1-7
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0:m[2:0]	0b111

```
integer m = UInt(CRm<2:0>);

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRCIMSPECn == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIMSPEC[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIMSPEC[m];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCIMSPEC[m];
```

```
MSR TRCIMSPEC<m>, <Xt> ; Where m = 1-7
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0:m[2:0]	0b111

```
integer m = UInt(CRm<2:0>);

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRCIMSPECn == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCIMSPEC[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCIMSPEC[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCIMSPEC[m] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.47 TRCIT, Trace Instrumentation

The TRCIT characteristics are:

Purpose

Generates an instrumentation packet in the trace.

Configuration

This system instruction is present only when FEAT_ITE is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TRCIT are UNDEFINED.

Attributes

TRCIT is a 64-bit System instruction.

Field descriptions

VALUE, bits [63:0]

Value to be included in the Instrumentation packet.

Executing TRCIT

This system instruction is an alias of the SYS instruction.

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TRCIT <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b011	0b0111	0b0010	0b111

```
if !(IsFeatureImplemented(FEAT_ITE) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    AArch64.TRCIT(X[t, 64]);
elsif PSTATE.EL == EL1 then
    AArch64.TRCIT(X[t, 64]);
elsif PSTATE.EL == EL2 then
    AArch64.TRCIT(X[t, 64]);
elsif PSTATE.EL == EL3 then
    AArch64.TRCIT(X[t, 64]);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.48 TRCITECR_EL1, Instrumentation Trace Control Register (EL1)

The TRCITECR_EL1 characteristics are:

Purpose

Provides EL1 controls for Trace Instrumentation.

Configuration

This register is present only when FEAT_ITE is implemented, System register access to the trace unit registers is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to TRCITECR_EL1 are UNDEFINED.

Attributes

TRCITECR_EL1 is a 64-bit register.

Field descriptions

Bits [63:2]

Reserved, RES0.

E1E, bit [1]

EL1 Instrumentation Trace Enable.

E1E	Meaning
0b0	Instrumentation trace prohibited at EL1.
0b1	Instrumentation trace not prohibited at EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

E0E, bit [0]

EL0 Instrumentation Trace Enable.

E0E	Meaning
0b0	Instrumentation trace prohibited at EL0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

0b1	Instrumentation trace not prohibited at EL0.

This field is ignored by the PE when EL2 is implemented and enabled in the current Security state and HCR_EL2.TGE == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing TRCITECR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name TRCITECR_EL1 or TRCITECR_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCITECR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_ITE) && IsFeatureImplemented(FEAT_TRC_SR) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnITE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGRTR2_EL2.nTRCITECR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnITE == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x888];
    else
        X[t, 64] = TRCITECR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnITE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnITE == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = TRCITECR_EL2;
    else
        X[t, 64] = TRCITECR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TRCITECR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MSR TRCITECR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_ITE) && IsFeatureImplemented(FEAT_TRC_SR) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnITE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGWTR2_EL2.nTRCITECR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnITE == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x888] = X[t, 64];
    else
        TRCITECR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnITE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnITE == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        TRCITECR_EL2 = X[t, 64];
    else
        TRCITECR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TRCITECR_EL1 = X[t, 64];
```

```
When FEAT_VHE is implemented MRS <Xt>, TRCITECR_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_ITE) && IsFeatureImplemented(FEAT_TRC_SR) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x888];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if ELIsInHost(EL2) then
            if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnITE == '0' then
                UNDEFINED;
            elsif HaveEL(EL3) && MDCR_EL3.EnITE == '0' then
                if EL3SDDUndef() then
                    UNDEFINED;
                else
                    AArch64.SystemAccessTrap(EL3, 0x18);
            else
                X[t, 64] = TRCITECR_EL1;
        else
            UNDEFINED;
    elsif PSTATE.EL == EL3 then
        if ELIsInHost(EL2) then
            X[t, 64] = TRCITECR_EL1;
        else
            UNDEFINED;
```

When FEAT_VHE is implemented MSR TRCITECR_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_ITE) && IsFeatureImplemented(FEAT_TRC_SR) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x888] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnITE == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.EnITE == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCITECR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        TRCITECR_EL1 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.49 TRCITECR_EL2, Instrumentation Trace Control Register (EL2)

The TRCITECR_EL2 characteristics are:

Purpose

Provides EL2 controls for Trace Instrumentation.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_ITE is implemented, System register access to the trace unit registers is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to TRCITECR_EL2 are UNDEFINED.

Attributes

TRCITECR_EL2 is a 64-bit register.

Field descriptions

Bits [63:2]

Reserved, RES0.

E2E, bit [1]

EL2 Instrumentation Trace Enable.

E2E	Meaning
0b0	Instrumentation trace prohibited at EL2.
0b1	Instrumentation trace not prohibited at EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

E0HE, bit [0]

EL0 Instrumentation Trace Enable.

E0HE	Meaning
0b0	Instrumentation trace prohibited at EL0 when HCR_EL2.TGE == 1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E0HE	Meaning
0b1	Instrumentation trace not prohibited at EL0 when HCR_EL2.TGE == 1.

This field is ignored by the PE when any of the following are true:

- HCR_EL2.TGE == 0.
- EL2 is disabled in the current Security state.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing TRCITECR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name TRCITECR_EL2 or TRCITECR_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCITECR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_ITE) && IsFeatureImplemented(FEAT_TRC_SR) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnITE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnITE == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = TRCITECR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TRCITECR_EL2;
```

```
MSR TRCITECR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0010	0b011

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_ITE) && IsFeatureImplemented(FEAT_TRC_SR) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnITE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnITE == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCITECR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TRCITECR_EL2 = X[t, 64];
```

MRS <Xt>, TRCITECR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_ITE) && IsFeatureImplemented(FEAT_TRC_SR) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnITE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGRTR2_EL2.nTRCITECR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnITE == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x888];
    else
        X[t, 64] = TRCITECR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnITE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnITE == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = TRCITECR_EL2;
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        X[t, 64] = TRCITECR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TRCITECR_EL1;
```

MSR TRCITECR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b011

```
if !(IsFeatureImplemented(FEAT_ITE) && IsFeatureImplemented(FEAT_TRC_SR) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnITE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGWTR2_EL2.nTRCITECR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnITE == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x888] = X[t, 64];
    else
        TRCITECR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnITE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnITE == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        TRCITECR_EL2 = X[t, 64];
    else
        TRCITECR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TRCITECR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.50 TRCITEEDCR, Instrumentation Trace Extension External Debug Control Register

The TRCITEEDCR characteristics are:

Purpose

Controls instrumentation trace filtering.

Configuration

AArch64 System register TRCITEEDCR bits [31:0] are architecturally mapped to External register TRCITEEDCR[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and FEAT_ITE is implemented. Otherwise, direct accesses to TRCITEEDCR are UNDEFINED.

Attributes

TRCITEEDCR is a 64-bit register.

Field descriptions

Bits [63:7]

Reserved, RES0.

RL, bit [6]

When FEAT_RME is implemented:

Instrumentation Trace in Realm state.

RL	Meaning
0b0	Instrumentation trace prohibited in Realm state.
0b1	Instrumentation trace permitted in Realm state.

This field is ignored when `SelfHostedTraceEnabled()` returns TRUE.

This field is used in conjunction with TRCCONFIGR.ITO and TRCITEEDCR.E<m> to control whether Instrumentation trace is permitted or prohibited in Realm state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

S, bit [5]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When Secure state is implemented:

Instrumentation Trace in Secure state.

S	Meaning
`0b0`	Instrumentation trace prohibited in Secure state.
`0b1`	Instrumentation trace permitted in Secure state.

This field is ignored when `SelfHostedTraceEnabled()` returns TRUE.

When FEAT_RME is not implemented, this field is used in conjunction with TRCCONFIGR.ITO, TRCITEEDCR.E3, and TRCITEEDCR.E<m> to control whether Instrumentation trace is permitted or prohibited in Secure state.

When FEAT_RME is implemented, this field is used in conjunction with TRCCONFIGR.ITO and TRCITEEDCR.E<m> to control whether Instrumentation trace is permitted or prohibited in Secure state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NS, bit [4]

When Any of Non-secure EL2, EL1, or EL0 are implemented:

Instrumentation Trace in Non-secure state.

NS	Meaning
`0b0`	Instrumentation trace prohibited in Non-secure state.
`0b1`	Instrumentation trace permitted in Non-secure state.

This field is ignored when `SelfHostedTraceEnabled()` returns TRUE.

This field is used in conjunction with TRCCONFIGR.ITO and TRCITEEDCR.E<m> to control whether Instrumentation trace is permitted or prohibited in Non-secure state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E3, bit [3]

When EL3 is implemented:

Instrumentation Trace Enable at EL3.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E3	Meaning
0b0	Instrumentation trace prohibited at EL3.
0b1	Instrumentation trace permitted at EL3.

This field is ignored when `SelfHostedTraceEnabled()` returns TRUE.

When FEAT_RME is not implemented, TRCITEEDCR.E3 is used in conjunction with TRCCONFIGR.ITO and TRCITEEDCR.S to control whether Instrumentation trace is permitted or prohibited at EL3.

When FEAT_RME is implemented, TRCITEEDCR.E3 is used in conjunction with TRCCONFIGR.ITO to control whether Instrumentation trace is permitted or prohibited at EL3.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E<m> , bits [m], for m = 2 to 0

Instrumentation Trace Enable at EL<m>.

E<m>	Meaning
0b0	Instrumentation trace prohibited at EL<m>.
0b1	Instrumentation trace permitted at EL<m>.

This field is ignored when `SelfHostedTraceEnabled()` returns TRUE.

This bit is used in conjunction with TRCCONFIGR.ITO, TRCITEEDCR.NS, TRCITEEDCR.S, and TRCITEEDCR.RL to control whether Instrumentation trace is permitted or prohibited at EL<m> in the specified Security states.

TRCITEEDCR.E<2> is RES0 if EL2 is not implemented in any Security states.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCITEEDCR

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCITEEDCR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0010	0b001

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) &&
     ↪IsFeatureImplemented(FEAT_ITE)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCITEEDCR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCITEEDCR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCITEEDCR;
```

```
MSR TRCITEEDCR, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) &&
     ↪IsFeatureImplemented(FEAT_ITE)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCITEEDCR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCITEEDCR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCITEEDCR = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.51 TRCOSLSR, Trace OS Lock Status Register

The TRCOSLSR characteristics are:

Purpose

Returns the status of the Trace OS Lock.

Configuration

AArch64 System register TRCOSLSR bits [31:0] are architecturally mapped to External register TRCOSLSR[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCOSLSR are UNDEFINED.

Attributes

TRCOSLSR is a 64-bit register.

Field descriptions

Bits [63:5]

Reserved, RES0.

OSLM, bits [4:3, 0]

OS Lock model.

The value of this field is an IMPLEMENTATION DEFINED choice of:

OSLM	Meaning
0b000	Trace OS Lock is not implemented.
0b010	Trace OS Lock is implemented.
0b100	Trace OS Lock is not implemented, and the trace unit is controlled by the PE OS Lock.

All other values are reserved.

When FEAT_ETE is implemented, the values 0b000 and 0b010 are not permitted.

Access to this field is RO.

Bit [2]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

OSLK, bit [1]

OS Lock status.

OSLK	Meaning
0b0	The OS Lock is unlocked.
0b1	The OS Lock is locked.

When FEAT_ETE is implemented, this field indicates the state of the PE OS Lock.

When FEAT_ETMv4 is implemented, this field indicates the state of the Trace OS Lock.

Accessing TRCOSLSR

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCOSLSR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0001	0b0001	0b100

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRCOSLSR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCOSLSR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCOSLSR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCOSLSR;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.52 TRCPRGCTLR, Trace Programming Control Register

The TRCPRGCTLR characteristics are:

Purpose

Enables the trace unit.

Configuration

AArch64 System register TRCPRGCTLR bits [31:0] are architecturally mapped to External register TRCPRGCTLR[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCPRGCTLR are UNDEFINED.

Attributes

TRCPRGCTLR is a 64-bit register.

Field descriptions

Bits [63:1]

Reserved, RES0.

EN, bit [0]

Trace unit enable.

EN	Meaning
0b0	The trace unit is disabled.
0b1	The trace unit is enabled.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to '0'.

Accessing TRCPRGCTLR

Must be programmed.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCPRGCTLR
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0001	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRCPRGCTLR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCPRGCTLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCPRGCTLR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCPRGCTLR;
```

```
MSR TRCPRGCTLR, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0001	0b000

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRCPRGCTLR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCPRGCTLR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCPRGCTLR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCPRGCTLR = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.53 TRCQCTLR, Trace Q Element Control Register

The TRCQCTLR characteristics are:

Purpose

Controls when Q elements are enabled.

Configuration

AArch64 System register TRCQCTLR bits [31:0] are architecturally mapped to External register TRCQCTLR[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and TRCIDR0.QFILT == '1'. Otherwise, direct accesses to TRCQCTLR are UNDEFINED.

Attributes

TRCQCTLR is a 64-bit register.

Field descriptions

Bits [63:9]

Reserved, RES0.

MODE, bit [8]

Selects whether the Address Range Comparators selected by TRCQCTLR.RANGE indicate address ranges where the trace unit is permitted to generate Q elements or address ranges where the trace unit is not permitted to generate Q elements:

MODE	Meaning
0b0	Exclude mode. The Address Range Comparators selected by TRCQCTLR.RANGE indicate address ranges where the trace unit must not generate Q elements. If no ranges are selected, Q elements are permitted across the entire memory map.
0b1	Include Mode. The Address Range Comparators selected by TRCQCTLR.RANGE indicate address ranges where the trace unit can generate Q elements. If all the implemented bits in RANGE are set to 0 then Q elements are disabled.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

RANGE[<m>] , bits [m], for m = 7 to 0

Specifies whether Address Range Comparator <m> controls Q elements.

RANGE[<m>]	Meaning
0b0	The address range that Address Range Comparator <m> defines is not selected.
0b1	The address range that Address Range Comparator <m> defines is selected.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR4.NUMACPAIRS), access to this field is RES0.
- Otherwise, access to this field is RW.

Accessing TRCQCTLR

Must be programmed if TRCCONFIGR.QE != 0b00.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCQCTLR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && TRCIDR0.QFILT == '1') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    else
        X[t, 64] = TRCQCTLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCQCTLR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCQCTLR;
```

```
MSR TRCQCTLR, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0001	0b001

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && TRCIDR0.QFILT == '1') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCQCTLR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCQCTLR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCQCTLR = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.54 TRCRSCTLR<n>, Trace Resource Selection Control Register <n>, n = 2 - 31

The TRCRSCTLR<n> characteristics are:

Purpose

Controls the selection of the resources in the trace unit.

Configuration

Resource selector 0 always returns FALSE.

Resource selector 1 always returns TRUE.

Resource selectors are implemented in pairs. Each odd numbered resource selector is part of a pair with the even numbered resource selector that is numbered as one less than it. For example, resource selectors 2 and 3 form a pair.

AArch64 System register TRCRSCTLR<n> bits [31:0] are architecturally mapped to External register TRCRSCTLR<n>[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and ((UInt(TRCIDR4.NUMRSPAIR) + 1) * 2) > n. Otherwise, direct accesses to TRCRSCTLR<n> are UNDEFINED.

Attributes

TRCRSCTLR<n> is a 64-bit register.

Field descriptions

Bits [63:22]

Reserved, RES0.

PAIRINV, bit [21]

When (n MOD 2) == 0:

Controls whether the combined result from a resource selector pair is inverted.

PAIRINV	Meaning
0b0	Do not invert the combined output of the 2 resource selectors.
0b1	Invert the combined output of the 2 resource selectors.

If:

- A is the register TRCRSCTLR<n>.
- B is the register TRCRSCTLR<n+1>.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Then the combined output of the 2 resource selectors A and B depends on the value of (A.PAIRINV, A.INV, B.INV) as follows:

- `0b000` -> A and B.
- `0b001` -> Reserved.
- `0b010` -> not(A) and B.
- `0b011` -> not(A) and not(B).
- `0b100` -> not(A) or not(B).
- `0b101` -> not(A) or B.
- `0b110` -> Reserved.
- `0b111` -> A or B.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

INV, bit [20]

Controls whether the resource, that TRCRSCTLR<n>.GROUP and TRCRSCTLR<n>.SELECT selects, is inverted.

INV	Meaning
`0b0`	Do not invert the output of this selector.
`0b1`	Invert the output of this selector.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

GROUP, bits [19:16]

Selects a group of resources.

GROUP	Meaning	SELECT
`0b0000`	External Input Selectors.	SELECT encoding for External Input Selectors
`0b0001`	PE Comparator Inputs.	SELECT encoding for PE Comparator Inputs
`0b0010`	Counters and Sequencer.	SELECT encoding for Counters and Sequencer
`0b0011`	Single-shot Comparator Controls.	SELECT encoding for Single-shot Comparator Controls
`0b0100`	Single Address Comparators.	SELECT encoding for Single Address Comparators
`0b0101`	Address Range Comparators.	SELECT encoding for Address Range Comparators
`0b0110`	Context Identifier Comparators.	SELECT encoding for Context Identifier Comparators
`0b0111`	Virtual Context Identifier Comparators.	SELECT encoding for Virtual Context Identifier Comparators

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

All other values are reserved.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT, bits [15:0]

Resource Specific Controls. Contains the controls specific to the resource group selected by GROUP, described in the following sections.

SELECT encoding for External Input Selectors

Bits [15:4]

Reserved, RES0.

EXTIN[<m>] , bits [m], for m = 3 to 0

Selects one or more External Inputs.

EXTIN[<m>]	Meaning
0b0	Ignore EXTIN <m>.
0b1	Select EXTIN <m>.

This bit is RES0 if m >= TRCIDR5.NUMEXTINSEL.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for PE Comparator Inputs

Bits [15:8]

Reserved, RES0.

PECOMP[<m>] , bits [m], for m = 7 to 0

Selects one or more PE Comparator Inputs.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

PECOMP[<m>]	Meaning
0b0	Ignore PE Comparator Input <m>.
0b1	Select PE Comparator Input <m>.

This bit is RES0 if m >= TRCIDR4.NUMPC.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for Counters and Sequencer

Bits [15:8]

Reserved, RES0.

SEQUENCER[<m>] , bits [m+4], for m = 3 to 0

Sequencer states.

SEQUENCER[<m>]	Meaning
0b0	Ignore Sequencer state <m>.
0b1	Select Sequencer state <m>.

This bit is RES0 if m >= TRCIDR5.NUMSEQSTATE.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

COUNTERS[<m>] , bits [m], for m = 3 to 0

Counters resources at zero.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

COUNTERS[<m>]	Meaning
0b0	Ignore Counter <m>.
0b1	Select Counter <m> is zero.

This bit is RES0 if m >= TRCIDR5.NUMCNTR.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for Single-shot Comparator Controls

Bits [15:8]

Reserved, RES0.

SINGLE_SHOT[<m>] , bits [m], for m = 7 to 0

Selects one or more Single-shot Comparator Controls.

SINGLE_SHOT[<m>]	Meaning
0b0	Ignore Single-shot Comparator Control <m>.
0b1	Select Single-shot Comparator Control <m>.

This bit is RES0 if m >= TRCIDR4.NUMSSCC.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for Single Address Comparators

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SAC[<m>] , bits [m], for m = 15 to 0

Selects one or more Single Address Comparators.

SAC[<m>]	Meaning
0b0	Ignore Single Address Comparator <m>.
0b1	Select Single Address Comparator <m>.

This bit is RES0 if m >= 2 × TRCIDR4.NUMACPAIRS.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for Address Range Comparators

Bits [15:8]

Reserved, RES0.

ARC[<m>] , bits [m], for m = 7 to 0

Selects one or more Address Range Comparators.

ARC[<m>]	Meaning
0b0	Ignore Address Range Comparator <m>.
0b1	Select Address Range Comparator <m>.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

This bit is RES0 if m >= TRCIDR4.NUMACPAIRS.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for Context Identifier Comparators

Bits [15:8]

Reserved, RES0.

CID[<m>] , bits [m], for m = 7 to 0

Selects one or more Context Identifier Comparators.

CID[<m>]	Meaning
0b0	Ignore Context Identifier Comparator <m>.
0b1	Select Context Identifier Comparator <m>.

This bit is RES0 if m >= TRCIDR4.NUMCIDC.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for Virtual Context Identifier Comparators

Bits [15:8]

Reserved, RES0.

VMID[<m>] , bits [m], for m = 7 to 0

Selects one or more Virtual Context Identifier Comparators.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

VMID[<m>]	Meaning
0b0	Ignore Virtual Context Identifier Comparator <m>.
0b1	Select Virtual Context Identifier Comparator <m>.

This bit is RES0 if m >= TRCIDR4.NUMVMIDC.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCRSCTLR<n>

Must be programmed if any of the following are true:

- TRCCNTCTLR<a>.RLDEVENT.TYPE == 0 and TRCCNTCTLR<a>.RLDEVENT.SEL == n.
- TRCCNTCTLR<a>.RLDEVENT.TYPE == 1 and TRCCNTCTLR<a>.RLDEVENT.SEL == n/2.
- TRCCNTCTLR<a>.CNTEVENT.TYPE == 0 and TRCCNTCTLR<a>.CNTEVENT.SEL == n.
- TRCCNTCTLR<a>.CNTEVENT.TYPE == 1 and TRCCNTCTLR<a>.CNTEVENT.SEL == n/2.
- TRCEVENTCTL0R.EVENT0.TYPE == 0 and TRCEVENTCTL0R.EVENT0.SEL == n.
- TRCEVENTCTL0R.EVENT0.TYPE == 1 and TRCEVENTCTL0R.EVENT0.SEL == n/2.
- TRCEVENTCTL0R.EVENT1.TYPE == 0 and TRCEVENTCTL0R.EVENT1.SEL == n.
- TRCEVENTCTL0R.EVENT1.TYPE == 1 and TRCEVENTCTL0R.EVENT1.SEL == n/2.
- TRCEVENTCTL0R.EVENT2.TYPE == 0 and TRCEVENTCTL0R.EVENT2.SEL == n.
- TRCEVENTCTL0R.EVENT2.TYPE == 1 and TRCEVENTCTL0R.EVENT2.SEL == n/2.
- TRCEVENTCTL0R.EVENT3.TYPE == 0 and TRCEVENTCTL0R.EVENT3.SEL == n.
- TRCEVENTCTL0R.EVENT3.TYPE == 1 and TRCEVENTCTL0R.EVENT3.SEL == n/2.
- TRCSEQEVR<a>.B.TYPE == 0 and TRCSEQEVR<a>.B.SEL = n.
- TRCSEQEVR<a>.B.TYPE == 1 and TRCSEQEVR<a>.B.SEL = n/2.
- TRCSEQEVR<a>.F.TYPE == 0 and TRCSEQEVR<a>.F.SEL = n.
- TRCSEQEVR<a>.F.TYPE == 1 and TRCSEQEVR<a>.F.SEL = n/2.
- TRCSEQRSTEVR.RST.TYPE == 0 and TRCSEQRSTEVR.RST.SEL == n.
- TRCSEQRSTEVR.RST.TYPE == 1 and TRCSEQRSTEVR.RST.SEL == n/2.
- TRCTSCTLR.EVENT.TYPE == 0 and TRCTSCTLR.EVENT.SEL == n.
- TRCTSCTLR.EVENT.TYPE == 1 and TRCTSCTLR.EVENT.SEL == n/2.
- TRCVICTLR.EVENT.TYPE == 0 and TRCVICTLR.EVENT.SEL == n.
- TRCVICTLR.EVENT.TYPE == 1 and TRCVICTLR.EVENT.SEL == n/2.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCRSCTLR<m> ; Where m = 2-31
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0001	m[3:0]	0b00:m[4]

```
integer m = UInt(op2<0>:CRm<3:0>);

if m >= NUM_TRACE_RESOURCE_SELECTOR_PAIRS * 2 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
     ↪HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
     ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCRSCTLR[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
     ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCRSCTLR[m];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
     ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCRSCTLR[m];
```

```
MSR TRCRSCTLR<m>, <Xt> ; Where m = 2-31
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0001	m[3:0]	0b00:m[4]

```
integer m = UInt(op2<0>:CRm<3:0>);

if m >= NUM_TRACE_RESOURCE_SELECTOR_PAIRS * 2 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCRSCTLR[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCRSCTLR[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCRSCTLR[m] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.55 TRCRSR, Trace Resources Status Register

The TRCRSR characteristics are:

Purpose

Use this to set, or read, the status of the resources.

Configuration

AArch64 System register TRCRSR bits [31:0] are architecturally mapped to External register TRCRSR[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCRSR are UNDEFINED.

Attributes

TRCRSR is a 64-bit register.

Field descriptions

Bits [63:13]

Reserved, RES0.

TA, bit [12]

Tracing active.

TA	Meaning
0b0	Tracing is not active.
0b1	Tracing is active.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

EVENT[<m>] , bits [m+8], for m = 3 to 0

Untraced status of ETEEvents.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EVENT[<m>]	Meaning
0b0	An ETEEvent <m> has not occurred.
0b1	An ETEEvent <m> has occurred while the resources were in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When TRCIDR4.NUMRSPAIR == ‘0000’, access to this field is RES0.
- Access to this field is RES0 if all of the following are true:
  - TRCIDR4.NUMRSPAIR != ‘0000’
  - m > UInt(TRCIDR0.NUMEVENT)
- Otherwise, access to this field is RW.

Bits [7:4]

Reserved, RES0.

EXTIN[<m>] , bits [m], for m = 3 to 0

The sticky status of the External Input Selectors.

EXTIN[<m>]	Meaning
0b0	An event selected by External Input Selector <m> has not occurred.
0b1	At least one event selected by External Input Selector <m> has occurred while the resources were in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR5.NUMEXTINSEL), access to this field is RES0.
- Otherwise, access to this field is RW.

Accessing TRCRSR

Must always be programmed.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCRSR
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1010	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCRSR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCRSR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCRSR;
```

```
MSR TRCRSR, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1010	0b000

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCRSR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCRSR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCRSR = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.56 TRCSEQEVR<n>, Trace Sequencer State Transition Control Register <n>, n = 0 - 2

The TRCSEQEVR<n> characteristics are:

Purpose

Moves the Sequencer state:

- Backwards, from state n+1 to state n when a programmed resource event occurs.
- Forwards, from state n to state n+1 when a programmed resource event occurs.

Configuration

AArch64 System register TRCSEQEVR<n> bits [31:0] are architecturally mapped to External register TRCSEQEVR<n>[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and TRCIDR5.NUMSEQSTATE != ‘000’. Otherwise, direct accesses to TRCSEQEVR<n> are UNDEFINED.

Attributes

TRCSEQEVR<n> is a 64-bit register.

Field descriptions

Bits [63:16]

Reserved, RES0.

B_TYPE, bit [15]

Backward field. Selects an event that causes the Sequencer to move from state n+1 to state n. For example, if TRCSEQEVR2.B.SEL == `0x14`, then when event `0x14` occurs, the Sequencer moves from state 3 to state 2.

Chooses the type of Resource Selector.

B_TYPE	Meaning
`0b0`	A single Resource Selector. TRCSEQEVR.B.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
`0b1`	A Boolean-combined pair of Resource Selectors. TRCSEQEVR.B.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCSEQEVR.B.SEL[4] is RES0.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [14:13]

Reserved, RES0.

B_SEL, bits [12:8]

Backward field. Selects an event that causes the Sequencer to move from state n+1 to state n. For example, if TRCSEQEVR2.B.SEL == `0x14`, then when event `0x14` occurs, the Sequencer moves from state 3 to state 2.

Defines the selected Resource Selector or pair of Resource Selectors. TRCSEQEVR.B.TYPE controls whether TRCSEQEVR.B.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

F_TYPE, bit [7]

Forward field. Selects an event that causes the Sequencer to move from state n to state n+1. For example, if TRCSEQEVR1.F.SEL == `0x12`, then when event `0x12` occurs, the Sequencer moves from state 1 to state 2.

Chooses the type of Resource Selector.

F_TYPE	Meaning
`0b0`	A single Resource Selector. TRCSEQEVR.F.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
`0b1`	A Boolean-combined pair of Resource Selectors. TRCSEQEVR.F.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCSEQEVR.F.SEL[4] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

F_SEL, bits [4:0]

Forward field. Selects an event that causes the Sequencer to move from state n to state n+1. For example, if TRCSEQEVR1.F.SEL == `0x12`, then when event `0x12` occurs, the Sequencer moves from state 1 to state 2.

Defines the selected Resource Selector or pair of Resource Selectors. TRCSEQEVR.F.TYPE controls whether TRCSEQEVR.F.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCSEQEVR<n>

Must be programmed if TRCRSCTLR<a>.GROUP == `0b0010` and TRCRSCTLR<a>.SEQUENCER != `0b0000`.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCSEQEVR<m> ; Where m = 0-2
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b00:m[1:0]	0b100

```
integer m = UInt(CRm<1:0>);

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSEQEVR[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSEQEVR[m];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSEQEVR[m];
```

MSR TRCSEQEVR<m>, <Xt> ; Where m = 0-2

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b00:m[1:0]	0b100

```
integer m = UInt(CRm<1:0>);

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSEQEVR[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSEQEVR[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        TRCSEQEVR[m] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.57 TRCSEQRSTEVR, Trace Sequencer Reset Control Register

The TRCSEQRSTEVR characteristics are:

Purpose

Moves the Sequencer to state 0 when a programmed resource event occurs.

Configuration

AArch64 System register TRCSEQRSTEVR bits [31:0] are architecturally mapped to External register TRCSEQRSTEVR[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and TRCIDR5.NUMSEQSTATE != ‘000’. Otherwise, direct accesses to TRCSEQRSTEVR are UNDEFINED.

Attributes

TRCSEQRSTEVR is a 64-bit register.

Field descriptions

Bits [63:8]

Reserved, RES0.

RST_TYPE, bit [7]

Reset field. Selects an event that causes the Sequencer to move to state 0.

Chooses the type of Resource Selector.

RST_TYPE	Meaning
0b0	A single Resource Selector. TRCSEQRSTEVR.RST.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
0b1	A Boolean-combined pair of Resource Selectors. TRCSEQRSTEVR.RST.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCSEQRSTEVR.RST.SEL[4] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

RST_SEL, bits [4:0]

Reset field. Selects an event that causes the Sequencer to move to state 0.

Defines the selected Resource Selector or pair of Resource Selectors. TRCSEQRSTEVR.RST.TYPE controls whether TRCSEQRSTEVR.RST.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCSEQRSTEVR

Must be programmed if TRCRSCTLR<a>.GROUP == `0b0010` and TRCRSCTLR<a>.SEQUENCER != `0b0000`.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCSEQRSTEVR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0110	0b100

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && TRCIDR5.NUMSEQSTATE !=
     ↪'000') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSEQRSTEVR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSEQRSTEVR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSEQRSTEVR;
```

```
MSR TRCSEQRSTEVR, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0110	0b100

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && TRCIDR5.NUMSEQSTATE !=
     ↪'000') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSEQRSTEVR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSEQRSTEVR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSEQRSTEVR = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.58 TRCSEQSTR, Trace Sequencer State Register

The TRCSEQSTR characteristics are:

Purpose

Use this to set, or read, the Sequencer state.

Configuration

AArch64 System register TRCSEQSTR bits [31:0] are architecturally mapped to External register TRCSEQSTR[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and TRCIDR5.NUMSEQSTATE != ‘000’. Otherwise, direct accesses to TRCSEQSTR are UNDEFINED.

Attributes

TRCSEQSTR is a 64-bit register.

Field descriptions

Bits [63:2]

Reserved, RES0.

STATE, bits [1:0]

Set or returns the state of the Sequencer.

STATE	Meaning
0b00	State 0.
0b01	State 1.
0b10	State 2.
0b11	State 3.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCSEQSTR

Must be programmed if TRCRSCTLR<a>.GROUP == 0b0010 and TRCRSCTLR<a>.SEQUENCER != 0b0000.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, TRCSEQSTR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0111	0b100

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && TRCIDR5.NUMSEQSTATE !=
     ↪'000') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRCSEQSTR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSEQSTR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSEQSTR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSEQSTR;
```

```
MSR TRCSEQSTR, <Xt>
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0111	0b100

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && TRCIDR5.NUMSEQSTATE !=
     ↪'000') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRCSEQSTR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSEQSTR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSEQSTR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSEQSTR = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.59 TRCSSCCR<n>, Trace Single-shot Comparator Control Register <n>, n = 0 - 7

The TRCSSCCR<n> characteristics are:

Purpose

Controls the corresponding Single-shot Comparator Control resource.

Configuration

AArch64 System register TRCSSCCR<n> bits [31:0] are architecturally mapped to External register TRCSSCCR<n>[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and UInt(TRCIDR4.NUMSSCC) > n. Otherwise, direct accesses to TRCSSCCR<n> are UNDEFINED.

Attributes

TRCSSCCR<n> is a 64-bit register.

Field descriptions

Bits [63:25]

Reserved, RES0.

RST, bit [24]

Selects the Single-shot Comparator Control mode.

RST	Meaning
0b0	The Single-shot Comparator Control is in single-shot mode.
0b1	The Single-shot Comparator Control is in multi-shot mode.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ARC[<m>] , bits [m+16], for m = 7 to 0

Selects one or more Address Range Comparators for Single-shot control.

ARC[<m>]	Meaning
0b0	The Address Range Comparator <m>, is not selected for Single-shot control.
0b1	The Address Range Comparator <m>, is selected for Single-shot control.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR4.NUMACPAIRS), access to this field is RES0.
- Otherwise, access to this field is RW.

SAC[<m>] , bits [m], for m = 15 to 0

Selects one or more Single Address Comparators for Single-shot control.

SAC[<m>]	Meaning
0b0	The Single Address Comparator <m>, is not selected for Single-shot control.
0b1	The Single Address Comparator <m>, is selected for Single-shot control.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= (UInt(TRCIDR4.NUMACPAIRS) * 2), access to this field is RES0.
- Otherwise, access to this field is RW.

Accessing TRCSSCCR<n>

Must be programmed if any TRCRSCTLR<a>.GROUP == 0b0011 and TRCRSCTLR<a>.SINGLE_SHOT[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCSSCCR<m> ; Where m = 0-7
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0001	0b0:m[2:0]	0b010

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
integer m = UInt(CRm<2:0>);

if m >= NUM_TRACE_SINGLE_SHOT_COMPARATOR_CONTROLS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSSCCR[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSSCCR[m];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSSCCR[m];
```

```
MSR TRCSSCCR<m>, <Xt> ; Where m = 0-7
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0001	0b0:m[2:0]	0b010

```
integer m = UInt(CRm<2:0>);

if m >= NUM_TRACE_SINGLE_SHOT_COMPARATOR_CONTROLS then
    UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSSCCR[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSSCCR[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSSCCR[m] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.60 TRCSSCSR<n>, Trace Single-shot Comparator Control Status Register <n>, n = 0 - 7

The TRCSSCSR<n> characteristics are:

Purpose

Returns the status of the corresponding Single-shot Comparator Control.

Configuration

AArch64 System register TRCSSCSR<n> bits [31:0] are architecturally mapped to External register TRCSSCSR<n>[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and UInt(TRCIDR4.NUMSSCC) > n. Otherwise, direct accesses to TRCSSCSR<n> are UNDEFINED.

Attributes

TRCSSCSR<n> is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

STATUS, bit [31]

Single-shot Comparator Control status. Indicates if any of the comparators selected by this Single-shot Comparator control have matched. The selected comparators are defined by TRCSSCCR<n>.ARC, TRCSSCCR<n>.SAC, and TRCSSPCICR<n>.PC.

STATUS	Meaning
0b0	No match has occurred. When the first match occurs, this field takes a value of 1. It remains at 1 until explicitly modified by a write to this register.
0b1	One or more matches has occurred. If TRCSSCCR<n>.RST == 0 then: • There is only one match and no more matches are possible. • Software must reset this field to 0 to re-enable the Single-shot Comparator Control.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

PENDING, bit [30]

Single-shot pending status. The Single-shot Comparator Control fired while the resources were in the Paused state.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

PENDING	Meaning
0b0	No match has occurred.
0b1	One or more matches has occurred.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [29:4]

Reserved, RES0.

PC, bit [3]

PE Comparator Input support. Indicates if the Single-shot Comparator Control supports PE Comparator Inputs.

PC	Meaning
0b0	This Single-shot Comparator Control does not support PE Comparator Inputs. Selecting any PE Comparator Inputs using the associated TRCSSPCICR<n> results in CONSTRAINED UNPREDICTABLE behavior of the Single-shot Comparator Control resource. The Single-shot Comparator Control might match unexpectedly or might not match.
0b1	This Single-shot Comparator Control supports PE Comparator Inputs.

Access to this field is RO.

DV, bit [2]

Data value comparator support. Data value comparisons are not implemented in ETE and are reserved for other trace architectures. Allocated in other trace architectures.

DV	Meaning
0b0	This Single-shot Comparator Control does not support data value comparisons.
0b1	This Single-shot Comparator Control supports data value comparisons.

This field reads as 0.

Access to this field is RO.

DA, bit [1]

Data Address Comparator support. Data address comparisons are not implemented in ETE and are reserved for other trace architectures. Allocated in other trace architectures.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

DA	Meaning
0b0	This Single-shot Comparator Control does not support data address comparisons.
0b1	This Single-shot Comparator Control supports data address comparisons.

This field reads as 0.

Access to this field is RO.

INST, bit [0]

Instruction Address Comparator support. Indicates if the Single-shot Comparator Control supports instruction address comparisons.

INST	Meaning
0b0	This Single-shot Comparator Control does not support instruction address comparisons.
0b1	This Single-shot Comparator Control supports instruction address comparisons.

This field reads as 1.

Access to this field is RO.

Accessing TRCSSCSR<n>

Must be programmed if TRCRSCTLR<a>.GROUP == 0b0011 and TRCRSCTLR<a>.SINGLE_SHOT[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCSSCSR<m> ; Where m = 0-7
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0001	0b1:m[2:0]	0b010

```
integer m = UInt(CRm<2:0>);

if m >= NUM_TRACE_SINGLE_SHOT_COMPARATOR_CONTROLS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRCSSCSRn == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSSCSR[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSSCSR[m];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSSCSR[m];
```

```
MSR TRCSSCSR<m>, <Xt> ; Where m = 0-7
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0001	0b1:m[2:0]	0b010

```
integer m = UInt(CRm<2:0>);

if m >= NUM_TRACE_SINGLE_SHOT_COMPARATOR_CONTROLS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRCSSCSRn == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSSCSR[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSSCSR[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSSCSR[m] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.61 TRCSSPCICR<n>, Trace Single-shot Processing Element Comparator Input Control Register <n>, n = 0 - 7

The TRCSSPCICR<n> characteristics are:

Purpose

Returns the status of the corresponding Single-shot Comparator Control.

Configuration

AArch64 System register TRCSSPCICR<n> bits [31:0] are architecturally mapped to External register TRCSSPCICR<n>[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, UInt(TRCIDR4.NUMSSCC) > n, UInt(TRCIDR4.NUMPC) > 0, and TRCSSCSR<n>.PC == '1'. Otherwise, direct accesses to TRCSSPCICR<n> are UNDEFINED.

Attributes

TRCSSPCICR<n> is a 64-bit register.

Field descriptions

Bits [63:8]

Reserved, RES0.

PC[<m>] , bits [m], for m = 7 to 0

Selects one or more PE Comparator Inputs for Single-shot control.

PC[<m>]	Meaning
0b0	The single PE Comparator Input <m>, is not selected as for Single-shot control.
0b1	The single PE Comparator Input <m>, is selected as for Single-shot control.

This bit is RES0 if m >= TRCIDR4.NUMPC.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing TRCSSPCICR<n>

Must be programmed if implemented and any TRCRSCTLR<a>.GROUP == `0b0011` and TRCRSCTLR<a>.SINGLE_SHOT[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCSSPCICR<m> ; Where m = 0-7
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0001	0b0:m[2:0]	0b011

```
integer m = UInt(CRm<2:0>);

if m >= NUM_TRACE_SINGLE_SHOT_COMPARATOR_CONTROLS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSSPCICR[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSSPCICR[m];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSSPCICR[m];
```

MSR TRCSSPCICR<m>, <Xt> ; Where m = 0-7

op0	op1	CRn	CRm	op2
0b10	0b001	0b0001	0b0:m[2:0]	0b011

```
integer m = UInt(CRm<2:0>);

if m >= NUM_TRACE_SINGLE_SHOT_COMPARATOR_CONTROLS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSSPCICR[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSSPCICR[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSSPCICR[m] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.62 TRCSTALLCTLR, Trace Stall Control Register

The TRCSTALLCTLR characteristics are:

Purpose

Enables trace unit functionality that prevents trace unit buffer overflows.

Configuration

AArch64 System register TRCSTALLCTLR bits [31:0] are architecturally mapped to External register TRCSTALLCTLR[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and TRCIDR3.STALLCTL == '1'. Otherwise, direct accesses to TRCSTALLCTLR are UNDEFINED.

Attributes

TRCSTALLCTLR is a 64-bit register.

Field descriptions

Bits [63:14]

Reserved, RES0.

NOOVERFLOW, bit [13]

When TRCIDR3.NOOVERFLOW == '1':

Trace overflow prevention.

NOOVERFLOW	Meaning
0b0	Trace unit buffer overflow prevention is disabled.
0b1	Trace unit buffer overflow prevention is enabled.

Note

Enabling this feature might cause a significant performance impact.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

Bits [12:9]

Reserved, RES0.

ISTALL, bit [8]

Instruction stall control. Controls if a trace unit can stall the PE when the trace buffer space is less than LEVEL.

ISTALL	Meaning
`0b0`	The trace unit must not stall the PE.
`0b1`	The trace unit can stall the PE.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [7:4]

Reserved, RES0.

LEVEL, bits [3:0]

Threshold level field. The field can support 16 monotonic levels from `0b0000` to `0b1111`.

The value `0b0000` defines the Minimal invasion level. This setting has a greater risk of a trace unit buffer overflow.

The value `0b1111` defines the Maximum invasion level. This setting has a reduced risk of a trace unit buffer overflow.

Note

For some implementations, invasion might occur at the minimal invasion level.

One or more of the least significant bits of LEVEL are permitted to be RES0. Arm recommends that LEVEL[3:2] are fully implemented. Arm strongly recommends that LEVEL[3] is always implemented. If one or more bits are RES0 and are written with a nonzero value, the effective value of LEVEL is rounded down to the nearest power of 2 value which has the RES0 bits as zero. For example, if LEVEL[1:0] are RES0 and a value of `0b1110` is written to LEVEL, the effective value of LEVEL is `0b1100`.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCSTALLCTLR

Must be programmed if implemented.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCSTALLCTLR
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1011	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && TRCIDR3.STALLCTL == '1') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSTALLCTLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSTALLCTLR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSTALLCTLR;
```

```
MSR TRCSTALLCTLR, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1011	0b000

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && TRCIDR3.STALLCTL == '1') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSTALLCTLR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSTALLCTLR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSTALLCTLR = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.63 TRCSTATR, Trace Status Register

The TRCSTATR characteristics are:

Purpose

Returns the trace unit status.

Configuration

AArch64 System register TRCSTATR bits [31:0] are architecturally mapped to External register TRCSTATR[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCSTATR are UNDEFINED.

Attributes

TRCSTATR is a 64-bit register.

Field descriptions

Bits [63:2]

Reserved, RES0.

PMSTABLE, bit [1]

Programmers' model stable.

PMSTABLE	Meaning
0b0	The programmers' model is not stable.
0b1	The programmers' model is stable.

Accessing this field has the following behavior:

- When the trace unit is enabled, access to this field is UNKNOWN/WI.
- Otherwise, access to this field is RO.

IDLE, bit [0]

Idle status.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

IDLE	Meaning
0b0	The trace unit is not idle.
0b1	The trace unit is idle.

Accessing TRCSTATR

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCSTATR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0011	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRCSTATR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSTATR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSTATR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
else
    X[t, 64] = TRCSTATR;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.64 TRCSYNCPR, Trace Synchronization Period Register

The TRCSYNCPR characteristics are:

Purpose

Controls how often trace protocol synchronization requests occur.

Configuration

AArch64 System register TRCSYNCPR bits [31:0] are architecturally mapped to External register TRCSYNCPR[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCSYNCPR are UNDEFINED.

Attributes

TRCSYNCPR is a 64-bit register.

Field descriptions

Bits [63:5]

Reserved, RES0.

PERIOD, bits [4:0]

Defines the number of bytes of trace between each periodic trace protocol synchronization request.

PERIOD	Meaning
0b00000	Trace protocol synchronization is disabled.
0b01000	Trace protocol synchronization request occurs after 2^8 bytes of trace.
0b01001	Trace protocol synchronization request occurs after 2^9 bytes of trace.
0b01010	Trace protocol synchronization request occurs after 2^{10} bytes of trace.
0b01011	Trace protocol synchronization request occurs after 2^{11} bytes of trace.
0b01100	Trace protocol synchronization request occurs after 2^{12} bytes of trace.
0b01101	Trace protocol synchronization request occurs after 2^{13} bytes of trace.
0b01110	Trace protocol synchronization request occurs after 2^{14} bytes of trace.
0b01111	Trace protocol synchronization request occurs after 2^{15} bytes of trace.
0b10000	Trace protocol synchronization request occurs after 2^{16} bytes of trace.
0b10001	Trace protocol synchronization request occurs after 2^{17} bytes of trace.
0b10010	Trace protocol synchronization request occurs after 2^{18} bytes of trace.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

PERIOD	Meaning
0b10011	Trace protocol synchronization request occurs after 2^{19} bytes of trace.
0b10100	Trace protocol synchronization request occurs after 2^{20} bytes of trace.

Other values are reserved. If a reserved value is programmed into PERIOD, then the behavior of the synchronization period counter is CONSTRAINED UNPREDICTABLE and one of the following behaviors occurs:

- No trace protocol synchronization requests are generated by this counter.
- Trace protocol synchronization requests occur at the specified period.
- Trace protocol synchronization requests occur at some other UNKNOWN period which can vary.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCSYNCPR

Must be programmed if TRCIDR3.SYNCPR == 0.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCSYNCPR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1101	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSYNCPR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSYNCPR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCSYNCPR;
```

```
MSR TRCSYNCPR, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1101	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSYNCPR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    else
        TRCSYNCPR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCSYNCPR = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.65 TRCTRACEIDR, Trace ID Register

The TRCTRACEIDR characteristics are:

Purpose

Sets the trace ID for instruction trace.

Configuration

AArch64 System register TRCTRACEIDR bits [31:0] are architecturally mapped to External register TRCTRACEIDR[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCTRACEIDR are UNDEFINED.

Attributes

TRCTRACEIDR is a 64-bit register.

Field descriptions

Bits [63:7]

Reserved, RES0.

TRACEID, bits [6:0]

Trace ID field. Sets the trace ID value for instruction trace. The width of the field is indicated by the value of TRCIDR5.TRACEIDSIZE. Unimplemented bits are RES0.

If an implementation supports AMBA ATB, then:

- The width of the field is 7 bits.
- Writing a reserved trace ID value does not affect behavior of the trace unit but it might cause UNPREDICTABLE behavior of the trace capture infrastructure.

See the AMBA ATB Protocol Specification for information about which ATID values are reserved.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCTRACEIDR

Must be programmed if implemented.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCTRACEIDR
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0000	0b001

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCTRACEIDR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCTRACEIDR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCTRACEIDR;
```

```
MSR TRCTRACEIDR, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0000	0b001

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCTRACEIDR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCTRACEIDR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCTRACEIDR = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.66 TRCTSCTLR, Trace Timestamp Control Register

The TRCTSCTLR characteristics are:

Purpose

Controls the insertion of global timestamps in the trace stream.

Configuration

AArch64 System register TRCTSCTLR bits [31:0] are architecturally mapped to External register TRCTSCTLR[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and TRCIDR0.TSSIZE != '00000'. Otherwise, direct accesses to TRCTSCTLR are UNDEFINED.

Attributes

TRCTSCTLR is a 64-bit register.

Field descriptions

Bits [63:8]

Reserved, RES0.

EVENT_TYPE, bit [7]

When TRCIDR4.NUMRSPAIR != '0000':

Chooses the type of Resource Selector.

EVENT_TYPE	Meaning
0b0	A single Resource Selector. TRCTSCTLR.EVENT.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
0b1	A Boolean-combined pair of Resource Selectors. TRCTSCTLR.EVENT.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCTSCTLR.EVENT.SEL[4] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bits [6:5]

Reserved, RES0.

EVENT_SEL, bits [4:0]

When TRCIDR4.NUMRSPAIR != '0000':

Defines the selected Resource Selector or pair of Resource Selectors. TRCTSCTLR.EVENT.TYPE controls whether TRCTSCTLR.EVENT.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TRCTSCTLR

Must be programmed if TRCCONFIGR.TS == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCTSCTLR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1100	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && TRCIDR0.TSSIZE != '00000')
    then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCTSCTLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCTSCTLR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCTSCTLR;
```

```
MSR TRCTSCTLR, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b1100	0b000

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && TRCIDR0.TSSIZE != '00000')
     ↪then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCTSCTLR = X[t, 64];
elsif PSTATE.EL == EL2 then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCTSCTLR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCTSCTLR = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.67 TRCVICTLR, Trace ViewInst Main Control Register

The TRCVICTLR characteristics are:

Purpose

Controls instruction trace filtering.

Configuration

AArch64 System register TRCVICTLR bits [31:0] are architecturally mapped to External register TRCVICTLR[31:0].

This register is present only when FEAT_ETE is implemented and System register access to the trace unit registers is implemented. Otherwise, direct accesses to TRCVICTLR are UNDEFINED.

Attributes

TRCVICTLR is a 64-bit register.

Field descriptions

Bits [63:27]

Reserved, RES0.

EXLEVEL_RL_EL2, bit [26]

When FEAT_RME is implemented:

Filter instruction trace for EL2 in Realm state.

EXLEVEL_RL_EL2	Meaning
0b0	When TRCVICTLR.EXLEVEL_NS_EL2 is 0 the trace unit generates instruction trace for EL2 in Realm state. When TRCVICTLR.EXLEVEL_NS_EL2 is 1 the trace unit does not generate instruction trace for EL2 in Realm state.
0b1	When TRCVICTLR.EXLEVEL_NS_EL2 is 0 the trace unit does not generate instruction trace for EL2 in Realm state. When TRCVICTLR.EXLEVEL_NS_EL2 is 1 the trace unit generates instruction trace for EL2 in Realm state.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_RL_EL1, bit [25]

When FEAT_RME is implemented:

Filter instruction trace for EL1 in Realm state.

EXLEVEL_RL_EL1	Meaning
0b0	When TRCVICTLR.EXLEVEL_NS_EL1 is 0 the trace unit generates instruction trace for EL1 in Realm state. When TRCVICTLR.EXLEVEL_NS_EL1 is 1 the trace unit does not generate instruction trace for EL1 in Realm state.
0b1	When TRCVICTLR.EXLEVEL_NS_EL1 is 0 the trace unit does not generate instruction trace for EL1 in Realm state. When TRCVICTLR.EXLEVEL_NS_EL1 is 1 the trace unit generates instruction trace for EL1 in Realm state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_RL_EL0, bit [24]

When FEAT_RME is implemented:

Filter instruction trace for EL0 in Realm state.

EXLEVEL_RL_EL0	Meaning
0b0	When TRCVICTLR.EXLEVEL_NS_EL0 is 0 the trace unit generates instruction trace for EL0 in Realm state. When TRCVICTLR.EXLEVEL_NS_EL0 is 1 the trace unit does not generate instruction trace for EL0 in Realm state.
0b1	When TRCVICTLR.EXLEVEL_NS_EL0 is 0 the trace unit does not generate instruction trace for EL0 in Realm state. When TRCVICTLR.EXLEVEL_NS_EL0 is 1 the trace unit generates instruction trace for EL0 in Realm state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bit [23]

Reserved, RES0.

EXLEVEL_NS_EL2, bit [22]

When Non-secure EL2 is implemented:

Filter instruction trace for EL2 in Non-secure state.

EXLEVEL_NS_EL2	Meaning
0b0	The trace unit generates instruction trace for EL2 in Non-secure state.
0b1	The trace unit does not generate instruction trace for EL2 in Non-secure state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_NS_EL1, bit [21]

When Non-secure EL1 is implemented:

Filter instruction trace for EL1 in Non-secure state.

EXLEVEL_NS_EL1	Meaning
0b0	The trace unit generates instruction trace for EL1 in Non-secure state.
0b1	The trace unit does not generate instruction trace for EL1 in Non-secure state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_NS_EL0, bit [20]

When Non-secure EL0 is implemented:

Filter instruction trace for EL0 in Non-secure state.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EXLEVEL_NS_EL0	Meaning
0b0	The trace unit generates instruction trace for EL0 in Non-secure state.
0b1	The trace unit does not generate instruction trace for EL0 in Non-secure state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL3, bit [19]

When EL3 is implemented:

Filter instruction trace for EL3.

EXLEVEL_S_EL3	Meaning
0b0	The trace unit generates instruction trace for EL3.
0b1	The trace unit does not generate instruction trace for EL3.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL2, bit [18]

When Secure EL2 is implemented:

Filter instruction trace for EL2 in Secure state.

EXLEVEL_S_EL2	Meaning
0b0	The trace unit generates instruction trace for EL2 in Secure state.
0b1	The trace unit does not generate instruction trace for EL2 in Secure state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EXLEVEL_S_EL1, bit [17]

When Secure EL1 is implemented:

Filter instruction trace for EL1 in Secure state.

EXLEVEL_S_EL1	Meaning
0b0	The trace unit generates instruction trace for EL1 in Secure state.
0b1	The trace unit does not generate instruction trace for EL1 in Secure state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL0, bit [16]

When Secure EL0 is implemented:

Filter instruction trace for EL0 in Secure state.

EXLEVEL_S_EL0	Meaning
0b0	The trace unit generates instruction trace for EL0 in Secure state.
0b1	The trace unit does not generate instruction trace for EL0 in Secure state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [15:12]

Reserved, RES0.

TRCERR, bit [11]

When TRCIDR3.TRCERR == '1':

Controls the forced tracing of System Error exceptions.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TRCERR	Meaning
0b0	Forced tracing of System Error exceptions is disabled.
0b1	Forced tracing of System Error exceptions is enabled.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRCRESET, bit [10]

Controls the forced tracing of PE Resets.

TRCRESET	Meaning
0b0	Forced tracing of PE Resets is disabled.
0b1	Forced tracing of PE Resets is enabled.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SSSTATUS, bit [9]

ViewInst start/stop function status.

SSSTATUS	Meaning
0b0	Stopped State. The ViewInst start/stop function is in the stopped state.
0b1	Started State. The ViewInst start/stop function is in the started state.

Before software enables the trace unit, it must write to this field to set the initial state of the ViewInst start/stop function. If the ViewInst start/stop function is not used then set this field to 1. Arm recommends that the value of this field is set before each trace session begins.

If the trace unit becomes disabled while a start point or stop point is still speculative, then the value of TRCVICTLR.SSSTATUS is UNKNOWN and might represent the result of a speculative start point or stop point.

If software which is running on the PE being traced disables the trace unit, either by clearing TRCPRGCTLR.EN or locking the OS Lock, Arm recommends that a DSB and an ISB instruction are executed before disabling the trace unit to prevent any start points or stop points being speculative at the point of disabling the trace unit. This procedure assumes that all start points or stop points occur before the barrier instructions are executed. The procedure does not guarantee that there are no speculative start points or stop points when disabling, although it helps minimize the probability.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- Access to this field is RES1 if all of the following are true:
  - TRCIDR4.NUMACPAIRS == ‘0000’
  - TRCIDR4.NUMPC == ‘0000’
- Otherwise, access to this field is RW.

Bit [8]

Reserved, RES0.

EVENT_TYPE, bit [7]

When TRCIDR4.NUMRSPAIR != '0000':

Chooses the type of Resource Selector.

EVENT_TYPE	Meaning
0b0	A single Resource Selector. TRCVICTLR.EVENT.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
0b1	A Boolean-combined pair of Resource Selectors. TRCVICTLR.EVENT.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCVICTLR.EVENT.SEL[4] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [6:5]

Reserved, RES0.

Bits [4:0]

When TRCIDR4.NUMRSPAIR != '0000':

EVENT_SEL, bits [4:0]

Defines the selected Resource Selector or pair of Resource Selectors. TRCVICTLR.EVENT.TYPE controls whether TRCVICTLR.EVENT.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

If an unimplemented Resource Selector is selected using this field, the value returned on a direct read of this field is UNKNOWN.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

When TRCIDR4.NUMRSPAIR == '0000':

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Reserved, bits [4:0]

This field is reserved:

- Bits [4:1] are RES0.
- Bit [0] is RES1.

Otherwise:

Reserved, RES0.

Accessing TRCVICTLR

Must be programmed.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCVICTLR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0000	0b010

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRCVICTLR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVICTLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        X[t, 64] = TRCVICTLR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVICTLR;
```

```
MSR TRCVICTLR, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0000	0b010

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRCVICTLR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVICTLR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVICTLR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
else
    TRCVICTLR = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.68 TRCVIIECTLR, Trace ViewInst Include/Exclude Control Register

The TRCVIIECTLR characteristics are:

Purpose

Use this to select, or read, the Address Range Comparators for the ViewInst include/exclude function.

Configuration

AArch64 System register TRCVIIECTLR bits [31:0] are architecturally mapped to External register TRCVIIECTLR[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and UInt(TRCIDR4.NUMACPAIRS) > 0. Otherwise, direct accesses to TRCVIIECTLR are UNDEFINED.

Attributes

TRCVIIECTLR is a 64-bit register.

Field descriptions

Bits [63:24]

Reserved, RES0.

EXCLUDE[<m>] , bits [m+16], for m = 7 to 0

Exclude Address Range Comparator <m>. Selects whether Address Range Comparator <m> is in use with the ViewInst exclude function.

EXCLUDE[<m>]	Meaning
0b0	The address range that Address Range Comparator <m> defines, is not selected for the ViewInst exclude function.
0b1	The address range that Address Range Comparator <m> defines, is selected for the ViewInst exclude function.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR4.NUMACPAIRS), access to this field is RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- Otherwise, access to this field is RW.

Bits [15:8]

Reserved, RES0.

INCLUDE[<m>] , bits [m], for m = 7 to 0

Include Address Range Comparator <m>.

Selects whether Address Range Comparator <m> is in use with the ViewInst include function.

Selecting no comparators for the ViewInst include function indicates that all instructions are included by default.

The ViewInst exclude function then indicates which ranges are excluded.

INCLUDE[<m>]	Meaning
0b0	The address range that Address Range Comparator <m> defines, is not selected for the ViewInst include function.
0b1	The address range that Address Range Comparator <m> defines, is selected for the ViewInst include function.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR4.NUMACPAIRS), access to this field is RES0.
- Otherwise, access to this field is RW.

Accessing TRCVIIECTLR

Must be programmed if TRCIDR4.NUMACPAIRS > 0b0000.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCVIIECTLR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0001	0b010

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && UInt(TRCIDR4.NUMACPAIRS) >
     ↪0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVIIECTLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVIIECTLR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVIIECTLR;
```

```
MSR TRCVIIECTLR, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0001	0b010

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && UInt(TRCIDR4.NUMACPAIRS) >
     ↪0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVIIECTLR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVIIECTLR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVIIECTLR = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.69 TRCVIPCSSCTLR, Trace ViewInst Start/Stop PE Comparator Control Register

The TRCVIPCSSCTLR characteristics are:

Purpose

Use this to select, or read, which PE Comparator Inputs can control the ViewInst start/stop function.

Configuration

AArch64 System register TRCVIPCSSCTLR bits [31:0] are architecturally mapped to External register TRCVIPCSSCTLR[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and UInt(TRCIDR4.NUMPC) > 0. Otherwise, direct accesses to TRCVIPCSSCTLR are UNDEFINED.

Attributes

TRCVIPCSSCTLR is a 64-bit register.

Field descriptions

Bits [63:24]

Reserved, RES0.

STOP[<m>] , bits [m+16], for m = 7 to 0

Selects whether PE Comparator Input <m> is in use with the ViewInst start/stop function for the purpose of stopping trace.

STOP[<m>]	Meaning
0b0	The PE Comparator Input <m> is not selected as a stop resource.
0b1	The PE Comparator Input <m> is selected as a stop resource.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR4.NUMPC), access to this field is RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- Otherwise, access to this field is RW.

Bits [15:8]

Reserved, RES0.

START[<m>] , bits [m], for m = 7 to 0

Selects whether PE Comparator Input <m> is in use with the ViewInst start/stop function for the purpose of starting trace.

START[<m>]	Meaning
0b0	The PE Comparator Input <m> is not selected as a start resource.
0b1	The PE Comparator Input <m> is selected as a start resource.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR4.NUMPC), access to this field is RES0.
- Otherwise, access to this field is RW.

Accessing TRCVIPCSSCTLR

Must be programmed if TRCIDR4.NUMPC != 0b0000.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCVIPCSSCTLR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0011	0b010

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && UInt(TRCIDR4.NUMPC) > 0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVIPCSSCTLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVIPCSSCTLR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVIPCSSCTLR;
```

MSR TRCVIPCSSCTLR, <Xt>

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0011	0b010

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && UInt(TRCIDR4.NUMPC) > 0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVIPCSSCTLR = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVIPCSSCTLR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVIPCSSCTLR = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.70 TRCVISSCTLR, Trace ViewInst Start/Stop Control Register

The TRCVISSCTLR characteristics are:

Purpose

Use this to select, or read, the Single Address Comparators for the ViewInst start/stop function.

Configuration

AArch64 System register TRCVISSCTLR bits [31:0] are architecturally mapped to External register TRCVISSCTLR[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and UInt(TRCIDR4.NUMACPAIRS) > 0. Otherwise, direct accesses to TRCVISSCTLR are UNDEFINED.

Attributes

TRCVISSCTLR is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

STOP[<m>] , bits [m+16], for m = 15 to 0

Selects whether Single Address Comparator <m> is used with the ViewInst start/stop function for the purpose of stopping trace.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

STOP[<m>]	Meaning
0b0	The Single Address Comparator <m> is not selected as a stop resource.
0b1	The Single Address Comparator <m> is selected as a stop resource.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= (UInt(TRCIDR4.NUMACPAIRS) * 2), access to this field is RES0.
- Otherwise, access to this field is RW.

START[<m>] , bits [m], for m = 15 to 0

Selects whether Single Address Comparator <m> is used with the ViewInst start/stop function for the purpose of starting trace.

START[<m>]	Meaning
0b0	The Single Address Comparator <m> is not selected as a start resource.
0b1	The Single Address Comparator <m> is selected as a start resource.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= (UInt(TRCIDR4.NUMACPAIRS) * 2), access to this field is RES0.
- Otherwise, access to this field is RW.

Accessing TRCVISSCTLR

Must be programmed if TRCIDR4.NUMACPAIRS > 0b0000.

For any 2 comparators selected for the ViewInst start/stop function, the comparator containing the lower address must be a lower numbered comparator.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCVISSCTLR
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0010	0b010

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && UInt(TRCIDR4.NUMACPAIRS) >
     ↪0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVISSCTLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVISSCTLR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVISSCTLR;
```

```
MSR TRCVISSCTLR, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0000	0b0010	0b010

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && UInt(TRCIDR4.NUMACPAIRS) >
     ↪0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVISSCTLR = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVISSCTLR = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVISSCTLR = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.71 TRCVMIDCCTLR0, Trace Virtual Context Identifier Comparator Control Register 0

The TRCVMIDCCTLR0 characteristics are:

Purpose

Virtual Context Identifier Comparator mask values for the TRCVMIDCVR<n> registers, where n=0-3.

Configuration

AArch64 System register TRCVMIDCCTLR0 bits [31:0] are architecturally mapped to External register TRCVMIDCCTLR0[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, UInt(TRCIDR4.NUMVMIDC) > `0x0`, and UInt(TRCIDR2.VMIDSIZE) > 0. Otherwise, direct accesses to TRCVMIDCCTLR0 are UNDEFINED.

Attributes

TRCVMIDCCTLR0 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

COMP3[<m>] , bits [m+24], for m = 7 to 0

When UInt(TRCIDR4.NUMVMIDC) > 3:

TRCVMIDCVR3 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR3. Each bit in this field corresponds to a byte in TRCVMIDCVR3.

COMP3[<m>]	Meaning
`0b0`	The trace unit includes TRCVMIDCVR3[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.
`0b1`	The trace unit ignores TRCVMIDCVR3[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.VMIDSIZE), access to this field is RES0.
- Otherwise, access to this field is RW.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

COMP2[<m>] , bits [m+16], for m = 7 to 0

When UInt(TRCIDR4.NUMVMIDC) > 2:

TRCVMIDCVR2 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR2. Each bit in this field corresponds to a byte in TRCVMIDCVR2.

COMP2[<m>]	Meaning
0b0	The trace unit includes TRCVMIDCVR2[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.
0b1	The trace unit ignores TRCVMIDCVR2[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.VMIDSIZE), access to this field is RES0.
- Otherwise, access to this field is RW.

Otherwise:

Reserved, RES0.

COMP1[<m>] , bits [m+8], for m = 7 to 0

When UInt(TRCIDR4.NUMVMIDC) > 1:

TRCVMIDCVR1 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR1. Each bit in this field corresponds to a byte in TRCVMIDCVR1.

COMP1[<m>]	Meaning
0b0	The trace unit includes TRCVMIDCVR1[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.
0b1	The trace unit ignores TRCVMIDCVR1[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.VMIDSIZE), access to this field is RES0.
- Otherwise, access to this field is RW.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

COMP0[<m>] , bits [m], for m = 7 to 0

When UInt(TRCIDR4.NUMVMIDC) > 0:

TRCVMIDCVR0 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR0. Each bit in this field corresponds to a byte in TRCVMIDCVR0.

COMP0[<m>]	Meaning
0b0	The trace unit includes TRCVMIDCVR0[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.
0b1	The trace unit ignores TRCVMIDCVR0[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.VMIDSIZE), access to this field is RES0.
- Otherwise, access to this field is RW.

Otherwise:

Reserved, RES0.

Accessing TRCVMIDCCTLR0

If software uses the TRCVMIDCVR<n> registers, where n=0-3, then it must program this register.

If software sets a mask bit to 1 then it must program the relevant byte in TRCVMIDCVR<n> to 0x00.

If any bit is 1 and the relevant byte in TRCVMIDCVR<n> is not 0x00, the behavior of the Virtual Context Identifier Comparator is CONSTRAINED UNPREDICTABLE. In this scenario the comparator might match unexpectedly or might not match.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCVMIDCCTLR0
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0011	0b0010	0b010

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && UInt(TRCIDR4.NUMVMIDC) > 0x0
     ↪&& UInt(TRCIDR2.VMIDSIZE) > 0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVMIDCCTLR0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVMIDCCTLR0;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVMIDCCTLR0;
```

```
MSR TRCVMIDCCTLR0, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0011	0b0010	0b010

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && UInt(TRCIDR4.NUMVMIDC) > 0x0
     ↪&& UInt(TRCIDR2.VMIDSIZE) > 0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVMIDCCTLR0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVMIDCCTLR0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVMIDCCTLR0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.72 TRCVMIDCCTLR1, Trace Virtual Context Identifier Comparator Control Register 1

The TRCVMIDCCTLR1 characteristics are:

Purpose

Virtual Context Identifier Comparator mask values for the TRCVMIDCVR<n> registers, where n=4-7.

Configuration

AArch64 System register TRCVMIDCCTLR1 bits [31:0] are architecturally mapped to External register TRCVMIDCCTLR1[31:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, UInt(TRCIDR4.NUMVMIDC) > `0x4`, and UInt(TRCIDR2.VMIDSIZE) > 0. Otherwise, direct accesses to TRCVMIDCCTLR1 are UNDEFINED.

Attributes

TRCVMIDCCTLR1 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

COMP7[<m>] , bits [m+24], for m = 7 to 0

When UInt(TRCIDR4.NUMVMIDC) > 7:

TRCVMIDCVR7 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR7. Each bit in this field corresponds to a byte in TRCVMIDCVR7.

COMP7[<m>]	Meaning
`0b0`	The trace unit includes TRCVMIDCVR7[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.
`0b1`	The trace unit ignores TRCVMIDCVR7[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.VMIDSIZE), access to this field is RES0.
- Otherwise, access to this field is RW.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

COMP6[<m>] , bits [m+16], for m = 7 to 0

When UInt(TRCIDR4.NUMVMIDC) > 6:

TRCVMIDCVR6 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR6. Each bit in this field corresponds to a byte in TRCVMIDCVR6.

COMP6[<m>]	Meaning
0b0	The trace unit includes TRCVMIDCVR6[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.
0b1	The trace unit ignores TRCVMIDCVR6[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.VMIDSIZE), access to this field is RES0.
- Otherwise, access to this field is RW.

Otherwise:

Reserved, RES0.

COMP5[<m>] , bits [m+8], for m = 7 to 0

When UInt(TRCIDR4.NUMVMIDC) > 5:

TRCVMIDCVR5 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR5. Each bit in this field corresponds to a byte in TRCVMIDCVR5.

COMP5[<m>]	Meaning
0b0	The trace unit includes TRCVMIDCVR5[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.
0b1	The trace unit ignores TRCVMIDCVR5[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.VMIDSIZE), access to this field is RES0.
- Otherwise, access to this field is RW.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

COMP4[<m>] , bits [m], for m = 7 to 0

When UInt(TRCIDR4.NUMVMIDC) > 4:

TRCVMIDCVR4 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR4. Each bit in this field corresponds to a byte in TRCVMIDCVR4.

COMP4[<m>]	Meaning
0b0	The trace unit includes TRCVMIDCVR4[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.
0b1	The trace unit ignores TRCVMIDCVR4[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= UInt(TRCIDR2.VMIDSIZE), access to this field is RES0.
- Otherwise, access to this field is RW.

Otherwise:

Reserved, RES0.

Accessing TRCVMIDCCTLR1

If software uses the TRCVMIDCVR<n> registers, where n=4-7, then it must program this register.

If software sets a mask bit to 1 then it must program the relevant byte in TRCVMIDCVR<n> to 0x00.

If any bit is 1 and the relevant byte in TRCVMIDCVR<n> is not 0x00, the behavior of the Virtual Context Identifier Comparator is CONSTRAINED UNPREDICTABLE. In this scenario the comparator might match unexpectedly or might not match.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCVMIDCCTLR1
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0011	0b0011	0b010

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && UInt(TRCIDR4.NUMVMIDC) > 0x4
     ↪&& UInt(TRCIDR2.VMIDSIZE) > 0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVMIDCCTLR1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVMIDCCTLR1;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVMIDCCTLR1;
```

```
MSR TRCVMIDCCTLR1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0011	0b0011	0b010

```
if !(IsFeatureImplemented(FEAT_ETE) && IsFeatureImplemented(FEAT_TRC_SR) && UInt(TRCIDR4.NUMVMIDC) > 0x4
     ↪&& UInt(TRCIDR2.VMIDSIZE) > 0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVMIDCCTLR1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVMIDCCTLR1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVMIDCCTLR1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.73 TRCVMIDCVR<n>, Trace Virtual Context Identifier Comparator Value Register <n>, n = 0 - 7

The TRCVMIDCVR<n> characteristics are:

Purpose

Contains the Virtual Context Identifier Comparator value.

Configuration

AArch64 System register TRCVMIDCVR<n> bits [63:0] are architecturally mapped to External register TRCVMIDCVR<n>[63:0].

This register is present only when FEAT_ETE is implemented, System register access to the trace unit registers is implemented, and UInt(TRCIDR4.NUMVMIDC) > n. Otherwise, direct accesses to TRCVMIDCVR<n> are UNDEFINED.

Attributes

TRCVMIDCVR<n> is a 64-bit register.

Field descriptions

VALUE, bits [63:0]

Virtual context identifier value. The width of this field is indicated by TRCIDR2.VMIDSIZE. Unimplemented bits are RES0. After a PE Reset, the trace unit assumes that the Virtual context identifier is zero until the PE updates the Virtual context identifier .

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCVMIDCVR<n>

Must be programmed if any of the following are true:

- TRCRSCTLR<a>.GROUP == `0b0111` and TRCRSCTLR<a>.VMID[n] == 1.
- TRCACATR<a>.CONTEXTTYPE == `0b10` or `0b11` and TRCACATR<a>.CONTEXT == n.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRCVMIDCVR<m> ; Where m = 0-7
```

op0	op1	CRn	CRm	op2
`0b10`	`0b001`	`0b0011`	m[2:0]:`0b0`	`0b001`

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
integer m = UInt(CRm<3:1>);

if m >= NUM_TRACE_VIRTUAL_CONTEXT_IDENTIFIER_COMPARATORS then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVMIDCVR[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVMIDCVR[m];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        X[t, 64] = TRCVMIDCVR[m];
```

```
MSR TRCVMIDCVR<m>, <Xt> ; Where m = 0-7
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b0011	m[2:0]:0b0	0b001

```
integer m = UInt(CRm<3:1>);

if m >= NUM_TRACE_VIRTUAL_CONTEXT_IDENTIFIER_COMPARATORS then
    UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVMIDCVR[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVMIDCVR[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_TRBE_EXT) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR2.TTA
         ↪== '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        TRCVMIDCVR[m] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.74 TRFCR_EL1, Trace Filter Control Register (EL1)

The TRFCR_EL1 characteristics are:

Purpose

Provides EL1 controls for Trace.

Configuration

AArch64 System register TRFCR_EL1 bits [31:0] are architecturally mapped to AArch32 System register TRFCR[31:0].

This register is present only when FEAT_TRF is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TRFCR_EL1 are UNDEFINED.

Attributes

TRFCR_EL1 is a 64-bit register.

Field descriptions

Bits [63:12]

Reserved, RES0.

DnVM, bit [11]

When FEAT_TRBEv1p1 is implemented and FEAT_NV is implemented:

Reserved for software use in nested virtualization. See also TRFCR_EL2.DnVM.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

KE, bit [10]

When FEAT_TRBE_EXC is implemented:

Kernel exception enable for TRBE Profiling exceptions taken to EL1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

KE	Meaning
0b0	TRBE Profiling exceptions taken to EL1 are always masked at EL1.
0b1	Enabled TRBE Profiling exceptions taken to EL1 are masked at EL1 when PSTATE.PM is 1 and unmasked when PSTATE.PM is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EE, bits [9:8]

When FEAT_TRBE_EXC is implemented:

Exception Enable.

EE	Meaning	Applies when
0b00	Disabled. TRBE Profiling exceptions for EL1 are disabled. All of the following apply: • Unless enabled by a higher Exception level, TRBE Profiling exceptions are not generated. • TRBSR_EL1.IRQ drives the interrupt request signal **TRBIRQ**. • Accesses to TRBSR_EL1 at EL1 ignore the value of HCR_EL2.NV1.	
0b01	Reserved for software use in nested virtualization. Behaves as 0b00 for the purpose of controlling the TRBE Profiling exception and interrupt request signal **TRBIRQ**, and as 0b11 for the purpose of accesses to TRBSR_EL1.	FEAT_NV is implemented
0b10	Reserved for software use in nested virtualization. Behaves as 0b11 for the purposes of controlling the TRBE Profiling exception and interrupt request signal **TRBIRQ**, and accesses to TRBSR_EL1.	FEAT_NV is implemented
0b11	Enabled. TRBE Profiling exceptions for EL1 are enabled, as follows: • All trace buffer management events are recorded in TRBSR_EL1, unless they are configured to be recorded in TRBSR_EL3 by MDCR_EL3.TRBEE or TRBSR_EL2 by TRFCR_EL2.EE. • TRBE Profiling exceptions are generated and taken to EL1 when unmasked and TRBSR_EL1.IRQ is 1, unless the Effective value of HCR_EL2.TGE is 1, in which case the exception is taken to EL2. • The interrupt request signal **TRBIRQ** is not asserted.	

For more information on the values reserved for software use in nested virtualization, see TRFCR_EL2.EE.

If the Effective value of TRFCR_EL2.EE is 0b00, then the Effective value of TRFCR_EL1.EE is 0b00.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '00'.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [7]

Reserved, RES0.

TS, bits [6:5]

Timestamp Control. Controls which timebase is used for trace timestamps.

TS	Meaning	Applies when
`0b00`	Reserved for software use in nested virtualization. Behaves as `0b01`. See also TRFCR_EL2.TS,	FEAT_NV2p1 is implemented
`0b01`	Virtual timestamp. The traced timestamp is the physical counter value minus the value of CNTVOFF_EL2.	
`0b10`	Guest physical timestamp. The traced timestamp is the physical counter value minus a physical offset. If any of the following are true, then the physical offset is zero, otherwise the physical offset is the value of CNTPOFF_EL2: • SCR_EL3.ECVEn == 0. • CNTHCTL_EL2.ECV == 0. • FEAT_ECV_POFF is not implemented.	FEAT_ECV is implemented
`0b11`	Physical timestamp. The traced timestamp is the physical counter value.	

All other values are reserved.

If any of the following are true, then this field is ignored by the PE:

- EL2 is implemented and TRFCR_EL2.TS is not `0b00`.
- `SelfHostedTraceEnabled()` == FALSE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [4]

Reserved, RES0.

CX, bit [3]

When FEAT_NV2p1 is implemented:

Reserved for software use in nested virtualization. See also TRFCR_EL2.CX.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [2]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E1TRE, bit [1]

EL1 Trace Enable.

E1TRE	Meaning
0b0	Trace is prohibited at EL1.
0b1	Trace is allowed at EL1.

If any of the following are true, then this field is ignored by the PE:

- `SelfHostedTraceEnabled()` == FALSE.
- The Effective value of HCR_EL2.TGE is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

E0TRE, bit [0]

EL0 Trace Enable.

E0TRE	Meaning
0b0	Trace is prohibited at EL0.
0b1	Trace is allowed at EL0.

If any of the following are true, then this field is ignored by the PE:

- `SelfHostedTraceEnabled()` == FALSE.
- The Effective value of HCR_EL2.TGE is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Accessing TRFCR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRFCR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_TRF) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif EL2Enabled() && MDCR_EL2.TTRF == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x880];
    else
        X[t, 64] = TRFCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = TRFCR_EL2;
    else
        X[t, 64] = TRFCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TRFCR_EL1;
```

MSR TRFCR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_TRF) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.TRFCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TTRF == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x880] = X[t, 64];
    else
        TRFCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif ELIsInHost(EL2) then
        TRFCR_EL2 = X[t, 64];
    else
        TRFCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TRFCR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, TRFCR_EL12

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_TRF) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x880];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TTRF == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = TRFCR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = TRFCR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented MSR TRFCR_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b0001	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_TRF) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x880] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TTRF == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRFCR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        TRFCR_EL1 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.4.75 TRFCR_EL2, Trace Filter Control Register (EL2)

The TRFCR_EL2 characteristics are:

Purpose

Provides EL2 controls for Trace.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

AArch64 System register TRFCR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HTRFCR[31:0].

This register is present only when FEAT_TRF is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to TRFCR_EL2 are UNDEFINED.

Attributes

TRFCR_EL2 is a 64-bit register.

Field descriptions

Bits [63:12]

Reserved, RES0.

DnVM, bit [11]

When FEAT_TRBEv1p1 is implemented:

Disable use of physical address trace buffer pointers.

DnVM	Meaning
0b0	Use of physical address trace buffer pointers is permitted.
0b1	Use of physical address trace buffer pointers is disabled. The PE behaves as if TRBLIMITR_EL1.nVM is 0.

If EL2 is disabled in the owning Security state, or the trace buffer owning Exception level is EL2, then the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

KE, bit [10]

When FEAT_TRBE_EXC is implemented:

Kernel exception enable for TRBE Profiling exceptions taken to EL2.

KE	Meaning
0b0	TRBE Profiling exceptions taken to EL2 are always masked at EL2.
0b1	Enabled TRBE Profiling exceptions taken to EL2 are masked at EL2 when PSTATE.PM is 1 and unmasked when PSTATE.PM is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EE, bits [9:8]

When FEAT_TRBE_EXC is implemented:

Exception Enable.

EE	Meaning
0b00	Disabled. TRBE Profiling exceptions for EL2 and EL1 are disabled. All of the following apply: • No trace buffer management events are recorded in TRBSR_EL2. • Unless enabled by a higher Exception level, TRBE Profiling exceptions are not generated. • TRBSR_EL1.IRQ drives the interrupt request signal **TRBIRQ**. • Accesses to TRBSR_EL1 at EL1 ignore the value of HCR_EL2.NV1 and accesses to TRBSR_EL1 at EL2 ignore the value of HCR_EL2.E2H.
0b01	Delegated. TRBE Profiling exceptions for EL2 are disabled, but might be enabled for EL1 by TRFCR_EL1.EE. All of the following apply: • No trace buffer management events are recorded in TRBSR_EL2. • TRBSR_EL2.IRQ is ignored and TRBE Profiling exceptions are not taken to EL2, other than for the case when the Effective value of HCR_EL2.TGE is 1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EE	Meaning

0b10	Enabled. TRBE Profiling exceptions for EL2 are enabled for trace buffer management events targeting EL2, as follows:
	• Trace buffer management events due to a fault on a write to the trace buffer that would generate a Data Abort exception taken to EL2 if generated by a store instruction executed at the owning Exception level are recorded in TRBSR_EL2, unless they are configured to be recorded in TRBSR_EL3 by MDCR_EL3.TRBEE. If the trace buffer owning Exception level is EL2, then this means any fault on a write to the trace buffer. If the trace buffer owning Exception level is EL1, then this means any of the following faults on a write to the trace buffer:
	  • Stage 2 faults.
	  • If HCR_EL2.TEA is 1, synchronous External aborts.
	  • If HCR_EL2.GPF is 1, Granule Protection Faults (GPFs).
	• Trace buffer management events due to Granule Protection Check faults other than GPFs on a write to the trace buffer are recorded in TRBSR_EL2, unless they are configured to be recorded in TRBSR_EL3 by MDCR_EL3.TRBEE.
	• TRBE Profiling exceptions are generated and taken to EL2 when unmasked and TRBSR_EL2.IRQ is 1.

0b11	Trap all. TRBE Profiling exceptions for EL2 are enabled for all trace buffer management events, as follows: • All trace buffer management events are recorded in TRBSR_EL2, unless they are configured to be recorded in TRBSR_EL3 by MDCR_EL3.TRBEE. • TRBE Profiling exceptions are generated and taken to EL2 when unmasked and TRBSR_EL2.IRQ is 1.

If the Effective value of MDCR_EL3.TRBEE is 0b00, then the Effective value of TRFCR_EL2.EE is 0b00. Otherwise, if EL2 is not implemented or the Effective value of SCR_EL3.{NS, EEL2} is {0, 0}, then the Effective value of TRFCR_EL2.EE is 0b01.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '00'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [7]

Reserved, RES0.

TS, bits [6:5]

Timestamp Control. Controls which timebase is used for trace timestamps.

TS	Meaning	Applies when
0b00	Timestamp controlled by TRFCR_EL1.TS or TRFCR.TS.	
0b01	Virtual timestamp. The traced timestamp is the physical counter value minus the value of CNTVOFF_EL2.	

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TS	Meaning	Applies when
0b10	Guest physical timestamp. The traced timestamp is the physical counter value minus a physical offset. If any of the following are true, the physical offset is zero, otherwise the physical offset is the value of CNTPOFF_EL2: • SCR_EL3.ECVEn == 0. • CNTHCTL_EL2.ECV == 0. • FEAT_ECV_POFF is not implemented.	FEAT_ECV is implemented
0b11	Physical timestamp. The traced timestamp is the physical counter value.	

If `SelfHostedTraceEnabled()` == FALSE, then this field is ignored by the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'00'`.

Bit [4]

Reserved, RES0.

CX, bit [3]

CONTEXTIDR_EL2 and VMID trace enable.

CX	Meaning
0b0	CONTEXTIDR_EL2 and VMID trace prohibited.
0b1	CONTEXTIDR_EL2 and VMID trace allowed.

If `SelfHostedTraceEnabled()` == FALSE, then this field is ignored by the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Bit [2]

Reserved, RES0.

E2TRE, bit [1]

EL2 Trace Enable.

E2TRE	Meaning
0b0	Trace is prohibited at EL2.
0b1	Trace is allowed at EL2.

If `SelfHostedTraceEnabled()` == FALSE, then this field is ignored by the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E0HTRE, bit [0]

EL0 Trace Enable.

E0HTRE	Meaning
0b0	Trace is prohibited at EL0.
0b1	Trace is allowed at EL0.

If any of the following are true, then this field is ignored by the PE:

- `SelfHostedTraceEnabled()` == FALSE.
- The Effective value of HCR_EL2.TGE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Accessing TRFCR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, TRFCR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_TRF) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = TRFCR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TRFCR_EL2;
```

```
MSR TRFCR_EL2, <Xt>
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b100	0b0001	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_TRF) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRFCR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TRFCR_EL2 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, TRFCR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_TRF) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif EL2Enabled() && MDCR_EL2.TTRF == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x880];
    else
        X[t, 64] = TRFCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = TRFCR_EL2;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    else
        X[t, 64] = TRFCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = TRFCR_EL1;
```

When FEAT_VHE is implemented MSR TRFCR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0001	0b0010	0b001

```
if !(IsFeatureImplemented(FEAT_TRF) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1')
         ↪&& HDFGWTR_EL2.TRFCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TTRF == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x880] = X[t, 64];
    else
        TRFCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        TRFCR_EL2 = X[t, 64];
    else
        TRFCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    TRFCR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5 Performance Monitors registers

This section lists the Performance Monitoring registers in AArch64.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.1 PMCCFILTR_EL0, Performance Monitors Cycle Count Filter Register

The PMCCFILTR_EL0 characteristics are:

Purpose

Determines the modes in which the Cycle Counter, PMCCNTR_EL0, increments.

Configuration

AArch64 System register PMCCFILTR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMCCFILTR[31:0].

When FEAT_PMUv3_TH is implemented, or FEAT_PMUv3p8 is implemented, or FEAT_PMUv3_EXT64 is implemented, or FEAT_PMUv3_SME is implemented, AArch64 System register PMCCFILTR_EL0 bits [63:32] are architecturally mapped to External register PMCCFILTR_EL0[63:32].

AArch64 System register PMCCFILTR_EL0 bits [31:0] are architecturally mapped to External register PMCCFILTR_EL0[31:0].

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMCCFILTR_EL0 are UNDEFINED.

Attributes

PMCCFILTR_EL0 is a 64-bit register.

Field descriptions

Bits [63:58]

Reserved, RES0.

VS, bits [57:56]

When FEAT_PMUv3_SME is implemented:

SVE mode filtering. Controls counting cycles in Streaming and Non-streaming SVE modes.

VS	Meaning
0b00	This mechanism has no effect on the filtering of cycles.
0b01	The PE does not count cycles in Streaming SVE mode.
0b10	The PE does not count cycles in Non-streaming SVE mode.

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [55:32]

Reserved, RES0.

P, bit [31]

EL1 filtering. Controls counting cycles in EL1.

P	Meaning
0b0	This mechanism has no effect on filtering of cycles.
0b1	The PE does not count cycles in EL1.

If Secure and Non-secure states are implemented, then counting cycles in Non-secure EL1 is further controlled by PMCCFILTR_EL0.NSK.

If FEAT_RME is implemented, then counting cycles in Realm EL1 is further controlled by PMCCFILTR_EL0.RLK.

If EL3 is implemented, then counting cycles in EL3 is further controlled by PMCCFILTR_EL0.M.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

U, bit [30]

EL0 filtering. Controls counting cycles in EL0.

U	Meaning
0b0	This mechanism has no effect on filtering of cycles.
0b1	The PE does not count cycles in EL0.

If Secure and Non-secure states are implemented, then counting cycles in Non-secure EL0 is further controlled by PMCCFILTR_EL0.NSU.

If FEAT_RME is implemented, then counting cycles in Realm EL0 is further controlled by PMCCFILTR_EL0.RLU.

The reset behavior of this field is:

- On a Cold reset:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

NSK, bit [29]

When EL3 is implemented:

Non-secure EL1 filtering. Controls counting cycles in Non-secure EL1. If PMCCFILTR_EL0.NSK is not equal to PMCCFILTR_EL0.P, then the PE does not count cycles in Non-secure EL1. Otherwise, this mechanism has no effect on filtering of cycles in Non-secure EL1.

NSK	Meaning
0b0	When PMCCFILTR_EL0.P == 0, this mechanism has no effect on filtering of cycles. When PMCCFILTR_EL0.P == 1, the PE does not count cycles in Non-secure EL1.
0b1	When PMCCFILTR_EL0.P == 0, the PE does not count cycles in Non-secure EL1. When PMCCFILTR_EL0.P == 1, this mechanism has no effect on filtering of cycles.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSU, bit [28]

When EL3 is implemented:

Non-secure EL0 filtering. Controls counting cycles in Non-secure EL0. If PMCCFILTR_EL0.NSU is not equal to PMCCFILTR_EL0.U, then the PE does not count cycles in Non-secure EL0. Otherwise, this mechanism has no effect on filtering of cycles in Non-secure EL0.

NSU	Meaning
0b0	When PMCCFILTR_EL0.U == 0, this mechanism has no effect on filtering of cycles. When PMCCFILTR_EL0.U == 1, the PE does not count cycles in Non-secure EL0.
0b1	When PMCCFILTR_EL0.U == 0, the PE does not count cycles in Non-secure EL0. When PMCCFILTR_EL0.U == 1, this mechanism has no effect on filtering of cycles.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

NSH, bit [27]

When EL2 is implemented:

EL2 filtering. Controls counting cycles in EL2.

NSH	Meaning
0b0	The PE does not count cycles in EL2.
0b1	This mechanism has no effect on filtering of cycles.

If EL3 is implemented and FEAT_SEL2 is implemented, then counting cycles in Secure EL2 is further controlled by PMCCFILTR_EL0.SH.

If FEAT_RME is implemented, then counting cycles in Realm EL2 is further controlled by PMCCFILTR_EL0.RLH.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

M, bit [26]

When EL3 is implemented:

EL3 filtering. Controls counting cycles in EL3. If PMCCFILTR_EL0.M is not equal to PMCCFILTR_EL0.P, then the PE does not count cycles in EL3. Otherwise, this mechanism has no effect on filtering of cycles in EL3.

M	Meaning
0b0	When PMCCFILTR_EL0.P == 0, this mechanism has no effect on filtering of cycles. When PMCCFILTR_EL0.P == 1, the PE does not count cycles in EL3.
0b1	When PMCCFILTR_EL0.P == 0, the PE does not count cycles in EL3. When PMCCFILTR_EL0.P == 1, this mechanism has no effect on filtering of cycles.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

Bit [25]

Reserved, RES0.

SH, bit [24]

When EL3 is implemented and FEAT_SEL2 is implemented:

Secure EL2 filtering. Controls counting cycles in Secure EL2. If PMCCFILTR_EL0.SH is equal to PMCCFILTR_EL0.NSH, then the PE does not count cycles in Secure EL2. Otherwise, this mechanism has no effect on filtering of cycles in Secure EL2.

SH	Meaning
0b0	When PMCCFILTR_EL0.NSH == 0, the PE does not count cycles in Secure EL2. When PMCCFILTR_EL0.NSH == 1, this mechanism has no effect on filtering of cycles.
0b1	When PMCCFILTR_EL0.NSH == 0, this mechanism has no effect on filtering of cycles. When PMCCFILTR_EL0.NSH == 1, the PE does not count cycles in Secure EL2.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

When Secure EL2 is not implemented, access to this field is RES0.

Otherwise:

Reserved, RES0.

Bit [23]

Reserved, RES0.

RLK, bit [22]

When FEAT_RME is implemented:

Realm EL1 filtering. Controls counting cycles in Realm EL1. If PMCCFILTR_EL0.RLK is not equal to PMCCFILTR_EL0.P, then the PE does not count cycles in Realm EL1. Otherwise, this mechanism has no effect on filtering of cycles in Realm EL1.

RLK	Meaning
0b0	When PMCCFILTR_EL0.P == 0, this mechanism has no effect on filtering of cycles. When PMCCFILTR_EL0.P == 1, the PE does not count cycles in Realm EL1.
0b1	When PMCCFILTR_EL0.P == 0, the PE does not count cycles in Realm EL1. When PMCCFILTR_EL0.P == 1, this mechanism has no effect on filtering of cycles.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLU, bit [21]

When FEAT_RME is implemented:

Realm EL0 filtering. Controls counting cycles in Realm EL0. If PMCCFILTR_EL0.RLU is not equal to PMCCFILTR_EL0.U, then the PE does not count cycles in Realm EL0. Otherwise, this mechanism has no effect on filtering of cycles in Realm EL0.

RLU	Meaning
0b0	When PMCCFILTR_EL0.U == 0, this mechanism has no effect on filtering of cycles. When PMCCFILTR_EL0.U == 1, the PE does not count cycles in Realm EL0.
0b1	When PMCCFILTR_EL0.U == 0, the PE does not count cycles in Realm EL0. When PMCCFILTR_EL0.U == 1, this mechanism has no effect on filtering of cycles.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLH, bit [20]

When FEAT_RME is implemented:

Realm EL2 filtering. Controls counting cycles in Realm EL2. If PMCCFILTR_EL0.RLH is equal to PMCCFILTR_EL0.NSH, then the PE does not count cycles in Realm EL2. Otherwise, this mechanism has no effect on filtering of cycles in Realm EL2.

RLH	Meaning
0b0	When PMCCFILTR_EL0.NSH == 0, the PE does not count cycles in Realm EL2. When PMCCFILTR_EL0.NSH == 1, this mechanism has no effect on filtering of cycles.
0b1	When PMCCFILTR_EL0.NSH == 0, this mechanism has no effect on filtering of cycles. When PMCCFILTR_EL0.NSH == 1, the PE does not count cycles in Realm EL2.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [19:0]

Reserved, RES0.

Accessing PMCCFILTR_EL0

PMCCFILTR_EL0 can also be accessed by using PMXEVTYPER_EL0 with PMSELR_EL0.SEL set to `0b11111`.

Permitted reads and writes of PMCCFILTR_EL0 are RAZ/WI if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- PMUSERENR_EL0.UEN == 1.
- PMUACR_EL1.C == 0.

Permitted writes of PMCCFILTR_EL0 are ignored if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- PMUSERENR_EL0.{UEN,CR} == {1,1}.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMCCFILTR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b1111	0b111

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')
         ↪then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMCCFILTR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1' && PMUACR_EL1.C == '0' then
            X[t, 64] = Zeros(64);
        else
            X[t, 64] = PMCCFILTR_EL0;
    elsif PSTATE.EL == EL1 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
             ↪HDFGRTR_EL2.PMCCFILTR_EL0 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = PMCCFILTR_EL0;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = PMCCFILTR_EL0;
    elsif PSTATE.EL == EL3 then
        X[t, 64] = PMCCFILTR_EL0;
```

MSR PMCCFILTR_EL0, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b1111	0b111

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')
         ↪then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMCCFILTR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1' && (PMUACR_EL1.C == '0' ||
        ↪PMUSERENR_EL0.CR == '1') then
        return;
    else
        PMCCFILTR_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        ↪HDFGWTR_EL2.PMCCFILTR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCCFILTR_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCCFILTR_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMCCFILTR_EL0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.2 PMCCNTR_EL0, Performance Monitors Cycle Count Register

The PMCCNTR_EL0 characteristics are:

Purpose

Holds the value of the processor Cycle Counter, CCNT, that counts processor clock cycles. See 'Time as measured by the Performance Monitors cycle counter' for more information.

PMCCFILTR_EL0 determines the modes and states in which the PMCCNTR_EL0 can increment.

Configuration

All counters are subject to any changes in clock frequency, including clock stopping caused by the WFI and WFE instructions. This means that it is CONSTRAINED UNPREDICTABLE whether or not PMCCNTR_EL0 continues to increment when clocks are stopped by WFI and WFE instructions.

AArch64 System register PMCCNTR_EL0 bits [63:0] are architecturally mapped to AArch32 System register PMCCNTR[63:0].

AArch64 System register PMCCNTR_EL0 bits [63:0] are architecturally mapped to External register PMCCNTR_EL0[63:0].

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMCCNTR_EL0 are UNDEFINED.

Attributes

PMCCNTR_EL0 is a 64-bit register.

Field descriptions

CCNT, bits [63:0]

Cycle count. Depending on the values of PMCR_EL0.{LC,D}, this field increments in one of the following ways:

- Every processor clock cycle.
- Every 64th processor clock cycle.

Writing 1 to PMCR_EL0.C sets this field to 0.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing PMCCNTR_EL0

Permitted reads and writes of PMCCNTR_EL0 are RAZ/WI if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- PMUSERENR_EL0.UEN == 1.
- PMUACR_EL1.C == 0.

Permitted writes of PMCCNTR_EL0 are ignored if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- PMUSERENR_EL0.{UEN,CR} == {1,1}.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMCCNTR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1101	0b000

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif (IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<UEN,CR,EN> == '000') ||
         ↪(!IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<CR,EN> == '00') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMCCNTR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1' && PMUACR_EL1.C == '0' then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = PMCCNTR_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.PMCCNTR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCCNTR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCCNTR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMCCNTR_EL0;
```

```
MSR PMCCNTR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1101	0b000

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')
         ↪then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMCCNTR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1' && (PMUACR_EL1.C == '0' ||
         ↪PMUSERENR_EL0.CR == '1') then
        return;
    else
        PMCCNTR_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.PMCCNTR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCCNTR_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCCNTR_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMCCNTR_EL0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.3 PMCCNTSVR_EL1, Performance Monitors Cycle Count Saved Value Register

The PMCCNTSVR_EL1 characteristics are:

Purpose

Captures the PMU Cycle counter, PMCCNTR_EL0.

Configuration

AArch64 System register PMCCNTSVR_EL1 bits [63:0] are architecturally mapped to External register PMCCNTSVR_EL1[63:0].

This register is present only when FEAT_PMUv3_SS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMCCNTSVR_EL1 are UNDEFINED.

Attributes

PMCCNTSVR_EL1 is a 64-bit register.

Field descriptions

CCNT, bits [63:0]

Sampled Cycle Count. The value of PMCCNTR_EL0 at the last successful Capture event.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing PMCCNTSVR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMCCNTSVR_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b1110	0b1011	0b111

```
if !(IsFeatureImplemented(FEAT_PMUv3_SS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMSS == '0' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGRTR2_EL2.nPMSSDATA == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPMSS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCCNTSVR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMSS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPMSS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCCNTSVR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMCCNTSVR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.4 PMCEID0_EL0, Performance Monitors Common Event Identification Register 0

The PMCEID0_EL0 characteristics are:

Purpose

Defines which Common architectural events and Common microarchitectural events are implemented, or counted, using PMU events in the ranges `0x0000` to `0x001F` and `0x4000` to `0x401F`.

For more information about the Common events and the use of the PMCEID<n>_EL0 registers see 'The PMU event number space and common events'.

Configuration

AArch64 System register PMCEID0_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMCEID0[31:0].

AArch64 System register PMCEID0_EL0 bits [63:32] are architecturally mapped to AArch32 System register PMCEID2[31:0].

AArch64 System register PMCEID0_EL0 bits [31:0] are architecturally mapped to External register PMCEID0[31:0].

AArch64 System register PMCEID0_EL0 bits [63:32] are architecturally mapped to External register PMCEID2[31:0].

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMCEID0_EL0 are UNDEFINED.

Attributes

PMCEID0_EL0 is a 64-bit register.

Field descriptions

IDhi<n> , bits [n+32], for n = 31 to 0

When FEAT_PMUv3p1 is implemented:

IDhi[n] corresponds to Common event (`0x4000` + n).

For each bit:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

IDhi<n>	Meaning
0b0	The Common event is not implemented, or not counted.
0b1	The Common event is implemented.

When the value of a bit in the field is 1, the corresponding Common event is implemented and counted.

Note

Arm recommends that if a Common event is never counted, the value of the corresponding bit is 0.

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional Common event.

Note

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features and has an event number that can be represented in the PMCEID<n>_EL0 registers of that earlier version of the PMU architecture.

Otherwise:

Reserved, RES0.

ID<n> , bits [n], for n = 31 to 0

ID[n] corresponds to Common event n.

For each bit:

ID<n>	Meaning
0b0	The Common event is not implemented, or not counted.
0b1	The Common event is implemented.

When the value of a bit in the field is 1, the corresponding Common event is implemented and counted.

Note

Arm recommends that if a Common event is never counted, the value of the corresponding bit is 0.

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional Common event.

Note

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features and has an event number that can be represented in the PMCEID<n>_EL0 registers of that earlier version of the PMU architecture.

Accessing PMCEID0_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMCEID0_EL0
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1100	0b110

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')
         ↪then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.TID == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMCEIDn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCEID0_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.PMCEIDn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCEID0_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCEID0_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMCEID0_EL0;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.5 PMCEID1_EL0, Performance Monitors Common Event Identification Register 1

The PMCEID1_EL0 characteristics are:

Purpose

Defines which Common architectural events and Common microarchitectural events are implemented, or counted, using PMU events in the ranges `0x0020` to `0x003F` and `0x4020` to `0x403F`.

For more information about the Common events and the use of the PMCEID<n>_EL0 registers see 'The PMU event number space and common events'.

Configuration

AArch64 System register PMCEID1_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMCEID1[31:0].

AArch64 System register PMCEID1_EL0 bits [63:32] are architecturally mapped to AArch32 System register PMCEID3[31:0].

AArch64 System register PMCEID1_EL0 bits [31:0] are architecturally mapped to External register PMCEID1[31:0].

AArch64 System register PMCEID1_EL0 bits [63:32] are architecturally mapped to External register PMCEID3[31:0].

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMCEID1_EL0 are UNDEFINED.

Attributes

PMCEID1_EL0 is a 64-bit register.

Field descriptions

IDhi<n> , bits [n+32], for n = 31 to 0

When FEAT_PMUv3p1 is implemented:

IDhi[n] corresponds to Common event (`0x4020` + n).

For each bit:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

IDhi<n>	Meaning
0b0	The Common event is not implemented, or not counted.
0b1	The Common event is implemented.

When the value of a bit in the field is 1, the corresponding Common event is implemented and counted.

Note

Arm recommends that if a Common event is never counted, the value of the corresponding bit is 0.

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional Common event.

Note

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features and has an event number that can be represented in the PMCEID<n>_EL0 registers of that earlier version of the PMU architecture.

Otherwise:

Reserved, RES0.

ID<n> , bits [n], for n = 31 to 0

ID[n] corresponds to Common event (0x0020 + n).

For each bit:

ID<n>	Meaning
0b0	The Common event is not implemented, or not counted.
0b1	The Common event is implemented.

When the value of a bit in the field is 1, the corresponding Common event is implemented and counted.

Note

Arm recommends that if a Common event is never counted, the value of the corresponding bit is 0.

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional Common event.

Note

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features and has an event number that can be represented in the PMCEID<n>_EL0 registers of that earlier version of the PMU architecture.

Accessing PMCEID1_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMCEID1_EL0
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1100	0b111

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')
         ↪then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.TID == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMCEIDn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCEID1_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.PMCEIDn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCEID1_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCEID1_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMCEID1_EL0;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.6 PMCNTENCLR_EL0, Performance Monitors Count Enable Clear Register

The PMCNTENCLR_EL0 characteristics are:

Purpose

Allows software to disable the following counters:

- The cycle counter PMCCNTR_EL0.
- The event counters PMEVCNTR<n>_EL0.
- When FEAT_PMUv3_ICNTR is implemented, the instruction counter PMICNTR_EL0.

Reading from this register shows which counters are enabled.

Configuration

AArch64 System register PMCNTENCLR_EL0 bits [63:0] are architecturally mapped to AArch64 System register PMCNTENSET_EL0[63:0].

AArch64 System register PMCNTENCLR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMCNTENCLR[31:0].

AArch64 System register PMCNTENCLR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMCNTENSET[31:0].

AArch64 System register PMCNTENCLR_EL0 bits [31:0] are architecturally mapped to External register PMCNTENCLR_EL0[31:0].

AArch64 System register PMCNTENCLR_EL0 bits [31:0] are architecturally mapped to External register PMCNTENSET_EL0[31:0].

When FEAT_PMUv3p9 is implemented or FEAT_PMUv3_EXT64 is implemented, AArch64 System register PMCNTENCLR_EL0 bits [63:32] are architecturally mapped to External register PMCNTENCLR_EL0[63:32].

When FEAT_PMUv3p9 is implemented or FEAT_PMUv3_EXT64 is implemented, AArch64 System register PMCNTENCLR_EL0 bits [63:32] are architecturally mapped to External register PMCNTENSET_EL0[63:32].

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMCNTENCLR_EL0 are UNDEFINED.

Attributes

PMCNTENCLR_EL0 is a 64-bit register.

Field descriptions

63 … 33	32
RES0	F0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

Bits [63:33]

Reserved, RES0.

F0, bit [32]

When FEAT_PMUv3_ICNTR is implemented:

PMICNTR_EL0 disable. On writes, allows software to disable PMICNTR_EL0. On reads, returns the PMICNTR_EL0 enable status.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F0	Meaning
0b0	PMICNTR_EL0 disabled.
0b1	PMICNTR_EL0 enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing this field has the following behavior:

- Access to this field is RAZ/WI if all of the following are true:
  - EL3 is implemented
  - PSTATE.EL != EL3
  - MDCR_EL3.EnPM2 == ‘0’
- Access to this field is RAZ/WI if all of the following are true:
  - PSTATE.EL == EL0
  - Any of the following are true:
    - PMUSERENR_EL0.UEN == ‘0’
    - PMUACR_EL1.F0 == ‘0’
- Access to this field is RAZ/WI if all of the following are true:
  - FEAT_FGT2 is implemented
  - EL2Enabled()
  - PSTATE.EL IN {EL1, EL0}
  - [HCR_EL2.E2H, HCR_EL2.TGE] != ‘11’
  - Any of the following are true:
    - All of the following are true:
      - EL3 is implemented
      - SCR_EL3.FGTEn2 == ‘0’
    - All of the following are true:
      - HDFGRTR2_EL2.nPMICFILTR_EL0 == ‘0’
      - HDFGWTR2_EL2.nPMICFILTR_EL0 == ‘0’
- Access to this field is RAZ/WI if all of the following are true:
  - FEAT_FGT2 is implemented
  - EL2Enabled()
  - PSTATE.EL == EL0
  - [HCR_EL2.E2H, HCR_EL2.TGE] != ‘11’
  - Any of the following are true:
    - All of the following are true:
      - EL3 is implemented
      - SCR_EL3.FGTEn2 == ‘0’
    - HDFGRTR2_EL2.nPMICFILTR_EL0 == ‘0’
  - PMUSERENR_EL0.IR == ‘1’
- Access to this field is WO/RAZ if all of the following are true:
  - FEAT_FGT2 is implemented
  - EL2Enabled()
  - PSTATE.EL IN {EL1, EL0}
  - [HCR_EL2.E2H, HCR_EL2.TGE] != ‘11’
  - Any of the following are true:
    - All of the following are true:
      - EL3 is implemented
      - SCR_EL3.FGTEn2 == ‘0’
    - HDFGRTR2_EL2.nPMICFILTR_EL0 == ‘0’
- Access to this field is RO if all of the following are true:
  - FEAT_FGT2 is implemented
  - EL2Enabled()

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- PSTATE.EL IN {EL1, EL0}
- [HCR_EL2.E2H, HCR_EL2.TGE] != ‘11’
- Any of the following are true:
  - All of the following are true:
    - EL3 is implemented
    - SCR_EL3.FGTEn2 == ‘0’
  - HDFGWTR2_EL2.nPMICFILTR_EL0 == ‘0’
- Access to this field is RO if all of the following are true:
  - PSTATE.EL == EL0
  - PMUSERENR_EL0.IR == ‘1’
- Otherwise, access to this field is W1C.

Otherwise:

Reserved, RES0.

C, bit [31]

PMCCNTR_EL0 disable. On writes, allows software to disable PMCCNTR_EL0. On reads, returns the PMCCNTR_EL0 enable status.

C	Meaning
0b0	PMCCNTR_EL0 disabled.
0b1	PMCCNTR_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is RAZ/WI if all of the following are true:
  - FEAT_PMUv3p9 is implemented
  - PSTATE.EL == EL0
  - PMUSERENR_EL0.UEN == ‘1’
  - PMUACR_EL1.C == ‘0’
- Access to this field is RO if all of the following are true:
  - FEAT_PMUv3p9 is implemented
  - PSTATE.EL == EL0
  - [PMUSERENR_EL0.UEN, PMUSERENR_EL0.CR] == ‘11’
- Otherwise, access to this field is W1C.

P<m> , bits [m], for m = 30 to 0

PMEVCNTR<m>_EL0 disable. On writes, allows software to disable PMEVCNTR<m>_EL0. On reads, returns the PMEVCNTR<m>_EL0 enable status.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

P<m>	Meaning
0b0	PMEVCNTR<m>_EL0 disabled.
0b1	PMEVCNTR<m>_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= GetNumEventCountersAccessible(), access to this field is RAZ/WI.
- Access to this field is RAZ/WI if all of the following are true:
  - FEAT_PMUv3p9 is implemented
  - PSTATE.EL == EL0
  - PMUSERENR_EL0.UEN == '1'
  - PMUACR_EL1.P<m> == '0'
- Access to this field is RO if all of the following are true:
  - FEAT_PMUv3p9 is implemented
  - PSTATE.EL == EL0
  - [PMUSERENR_EL0.UEN, PMUSERENR_EL0.ER] == '11'
- Otherwise, access to this field is W1C.

Accessing PMCNTENCLR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMCNTENCLR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1100	0b010

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')
         ↪then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMCNTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    else
        X[t, 64] = PMCNTENCLR_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.PMCNTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCNTENCLR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCNTENCLR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMCNTENCLR_EL0;
```

```
MSR PMCNTENCLR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1100	0b010

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')
         ↪then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMCNTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCNTENCLR_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.PMCNTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCNTENCLR_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCNTENCLR_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMCNTENCLR_EL0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.7 PMCNTENSET_EL0, Performance Monitors Count Enable Set Register

The PMCNTENSET_EL0 characteristics are:

Purpose

Allows software to enable the following counters:

- The cycle counter PMCCNTR_EL0.
- The event counters PMEVCNTR<n>_EL0.
- When FEAT_PMUv3_ICNTR is implemented, the instruction counter PMICNTR_EL0.

Reading from this register shows which counters are enabled.

Configuration

AArch64 System register PMCNTENSET_EL0 bits [63:0] are architecturally mapped to AArch64 System register PMCNTENCLR_EL0[63:0].

AArch64 System register PMCNTENSET_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMCNTENSET[31:0].

AArch64 System register PMCNTENSET_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMCNTENCLR[31:0].

AArch64 System register PMCNTENSET_EL0 bits [31:0] are architecturally mapped to External register PMCNTENSET_EL0[31:0].

AArch64 System register PMCNTENSET_EL0 bits [31:0] are architecturally mapped to External register PMCNTENCLR_EL0[31:0].

When FEAT_PMUv3p9 is implemented or FEAT_PMUv3_EXT64 is implemented, AArch64 System register PMCNTENSET_EL0 bits [63:32] are architecturally mapped to External register PMCNTENSET_EL0[63:32].

When FEAT_PMUv3p9 is implemented or FEAT_PMUv3_EXT64 is implemented, AArch64 System register PMCNTENSET_EL0 bits [63:32] are architecturally mapped to External register PMCNTENCLR_EL0[63:32].

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMCNTENSET_EL0 are UNDEFINED.

Attributes

PMCNTENSET_EL0 is a 64-bit register.

Field descriptions

Bits [63:33]

Reserved, RES0.

F0, bit [32]

When FEAT_PMUv3_ICNTR is implemented:

PMICNTR_EL0 enable. On writes, allows software to enable PMICNTR_EL0. On reads, returns the PMICNTR_EL0 enable status.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F0	Meaning
0b0	PMICNTR_EL0 disabled.
0b1	PMICNTR_EL0 enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing this field has the following behavior:

- Access to this field is RAZ/WI if all of the following are true:
  - EL3 is implemented
  - PSTATE.EL != EL3
  - MDCR_EL3.EnPM2 == ‘0’
- Access to this field is RAZ/WI if all of the following are true:
  - PSTATE.EL == EL0
  - Any of the following are true:
    - PMUSERENR_EL0.UEN == ‘0’
    - PMUACR_EL1.F0 == ‘0’
- Access to this field is RAZ/WI if all of the following are true:
  - FEAT_FGT2 is implemented
  - EL2Enabled()
  - PSTATE.EL IN {EL1, EL0}
  - [HCR_EL2.E2H, HCR_EL2.TGE] != ‘11’
  - Any of the following are true:
    - All of the following are true:
      - EL3 is implemented
      - SCR_EL3.FGTEn2 == ‘0’
    - All of the following are true:
      - HDFGRTR2_EL2.nPMICFILTR_EL0 == ‘0’
      - HDFGWTR2_EL2.nPMICFILTR_EL0 == ‘0’
- Access to this field is RAZ/WI if all of the following are true:
  - FEAT_FGT2 is implemented
  - EL2Enabled()
  - PSTATE.EL == EL0
  - [HCR_EL2.E2H, HCR_EL2.TGE] != ‘11’
  - Any of the following are true:
    - All of the following are true:
      - EL3 is implemented
      - SCR_EL3.FGTEn2 == ‘0’
    - HDFGRTR2_EL2.nPMICFILTR_EL0 == ‘0’
  - PMUSERENR_EL0.IR == ‘1’
- Access to this field is WO/RAZ if all of the following are true:
  - FEAT_FGT2 is implemented
  - EL2Enabled()
  - PSTATE.EL IN {EL1, EL0}
  - [HCR_EL2.E2H, HCR_EL2.TGE] != ‘11’
  - Any of the following are true:
    - All of the following are true:
      - EL3 is implemented
      - SCR_EL3.FGTEn2 == ‘0’
    - HDFGRTR2_EL2.nPMICFILTR_EL0 == ‘0’
- Access to this field is RO if all of the following are true:
  - FEAT_FGT2 is implemented
  - EL2Enabled()

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- PSTATE.EL IN {EL1, EL0}
- [HCR_EL2.E2H, HCR_EL2.TGE] != ‘11’
- Any of the following are true:
  - All of the following are true:
    - EL3 is implemented
    - SCR_EL3.FGTEn2 == ‘0’
  - HDFGWTR2_EL2.nPMICFILTR_EL0 == ‘0’
- Access to this field is RO if all of the following are true:
  - PSTATE.EL == EL0
  - PMUSERENR_EL0.IR == ‘1’
- Otherwise, access to this field is W1S.

Otherwise:

Reserved, RES0.

C, bit [31]

PMCCNTR_EL0 enable. On writes, allows software to enable PMCCNTR_EL0. On reads, returns the PMCCNTR_EL0 enable status.

C	Meaning
0b0	PMCCNTR_EL0 disabled.
0b1	PMCCNTR_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is RAZ/WI if all of the following are true:
  - FEAT_PMUv3p9 is implemented
  - PSTATE.EL == EL0
  - PMUSERENR_EL0.UEN == ‘1’
  - PMUACR_EL1.C == ‘0’
- Access to this field is RO if all of the following are true:
  - FEAT_PMUv3p9 is implemented
  - PSTATE.EL == EL0
  - [PMUSERENR_EL0.UEN, PMUSERENR_EL0.CR] == ‘11’
- Otherwise, access to this field is W1S.

P<m> , bits [m], for m = 30 to 0

PMEVCNTR<m>_EL0 enable. On writes, allows software to enable PMEVCNTR<m>_EL0. On reads, returns the PMEVCNTR<m>_EL0 enable status.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

P<m>	Meaning
0b0	PMEVCNTR<m>_EL0 disabled.
0b1	PMEVCNTR<m>_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= GetNumEventCountersAccessible(), access to this field is RAZ/WI.
- Access to this field is RAZ/WI if all of the following are true:
  - FEAT_PMUv3p9 is implemented
  - PSTATE.EL == EL0
  - PMUSERENR_EL0.UEN == '1'
  - PMUACR_EL1.P<m> == '0'
- Access to this field is RO if all of the following are true:
  - FEAT_PMUv3p9 is implemented
  - PSTATE.EL == EL0
  - [PMUSERENR_EL0.UEN, PMUSERENR_EL0.ER] == '11'
- Otherwise, access to this field is W1S.

Accessing PMCNTENSET_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMCNTENSET_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1100	0b001

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')
         ↪then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMCNTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    else
        X[t, 64] = PMCNTENSET_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.PMCNTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCNTENSET_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMCNTENSET_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMCNTENSET_EL0;
```

MSR PMCNTENSET_EL0, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1100	0b001

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')
         ↪then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMCNTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCNTENSET_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.PMCNTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCNTENSET_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCNTENSET_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMCNTENSET_EL0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.8 PMCR_EL0, Performance Monitors Control Register

The PMCR_EL0 characteristics are:

Purpose

Provides details of the Performance Monitors implementation, including the number of counters implemented, and configures and controls the counters.

Configuration

AArch64 System register PMCR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMCR[31:0].

AArch64 System register PMCR_EL0 bits [31:0] are architecturally mapped to External register PMCR_EL0[31:0].

When FEAT_PMUv3_EXT64 is implemented, AArch64 System register PMCR_EL0 bits [63:32] are architecturally mapped to External register PMCR_EL0[63:32].

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMCR_EL0 are UNDEFINED.

Attributes

PMCR_EL0 is a 64-bit register.

Field descriptions

Bits [63:33]

Reserved, RES0.

FZS, bit [32]

When FEAT_SPEv1p2 is implemented:

Freeze-on-SPE event. Stop counters when PMBLIMITR_EL1.{PMFZ,E} is {1,1} and profiling is stopped.

FZS	Meaning
0b0	Do not freeze on a Statistical Profiling Buffer Management event.
0b1	Affected counters do not count following a Statistical Profiling Buffer Management event.

The pseudocode function `SPEProfilingStopped` describes when profiling is stopped.

The counters affected by this field are:

- The event counters in the first range.
- If FEAT_PMUv3_ICNTR is implemented, the instruction counter PMICNTR_EL0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- If FEAT_SPE_DPFZS is implemented and PMCR_EL0.DP is 1, the cycle counter PMCCNTR_EL0.

Other event counters are not affected by this field.

When FEAT_SPE_DPFZS is not implemented or PMCR_EL0.DP is 0, PMCCNTR_EL0 is not affected by this field.

For more information about event counter ranges, see MDCR_EL2.HPMN.

The reset behavior of this field is:

- On a Warm reset:
  - When FEAT_AA32 is implemented, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IMP, bits [31:24]

When FEAT_PMUv3p7 is not implemented:

Implementer code.

If this field is zero, then PMCR_EL0.IDCODE is RES0 and software must use MIDR_EL1 to identify the PE.

Otherwise, this field and PMCR_EL0.IDCODE identify the PMU implementation to software. The implementer codes are allocated by Arm. A nonzero value has the same interpretation as MIDR_EL1.Implementer.

This field has an IMPLEMENTATION DEFINED value.

Arm deprecates use of this field.

Access to this field is RO.

Otherwise:

Reserved, RAZ.

IDCODE, bits [23:16]

When PMCR_EL0.IMP != '00000000':

Identification code. Arm deprecates use of this field.

Each implementer must maintain a list of identification codes that are specific to the implementer. A specific implementation is identified by the combination of the implementer code and the identification code.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Otherwise:

Reserved, RES0.

N, bits [15:11]

Indicates the number of event counters implemented. This value is in the range of `0b00000`-`0b11111`. If the value is `0b00000`, then only PMCCNTR_EL0 is implemented. If the value is `0b11111`, then PMCCNTR_EL0 and 31 event counters are implemented.

When EL2 is implemented and enabled for the current Security state, reads of this field from EL1 and EL0 return the Effective value of MDCR_EL2.HPMN.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bit [10]

Reserved, RES0.

FZO, bit [9]

When FEAT_PMUv3p7 is implemented:

Freeze-on-overflow. Stop event counters on overflow.

FZO	Meaning
0b0	Do not freeze on overflow.
0b1	Affected counters do not count when any of the following applies: • For any event counter PMEVCNTR<m>_EL0 in the first range, PMOVSCLR_EL0[m] is 1, and either FEAT_SEBEP is not implemented or PMEVTYPER<m>_EL0.SYNC is 0. • FEAT_PMUv3_ICNTR is implemented, PMOVSCLR_EL0.F0 is 1, and either FEAT_SEBEP is not implemented or PMICFILTR_EL0.SYNC is 0.

The counters affected by this field are:

- The event counters in the first range.
- If FEAT_PMUv3_ICNTR is implemented, the instruction counter PMICNTR_EL0.
- If PMCR_EL0.DP is 1, the cycle counter PMCCNTR_EL0.

Other event counters are not affected by this field.

When PMCR_EL0.DP is 0, PMCCNTR_EL0 is not affected by this field.

For more information about event counter ranges, see MDCR_EL2.HPMN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [8]

Reserved, RES0.

LP, bit [7]

When FEAT_PMUv3p5 is implemented:

Long event counter enable. Determines when unsigned overflow is recorded by PMOVSCLR_EL0.P[n].

LP	Meaning
0b0	Event counter overflow on increment that causes unsigned overflow of PMEVCNTR<n>_EL0[31:0].
0b1	Event counter overflow on increment that causes unsigned overflow of PMEVCNTR<n>_EL0[63:0].

When FEAT_EBEP is implemented and the PMU Profiling exception is enabled, the Effective value of this field is 1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The counters affected by this field are the event counters in the first range. For more information about event counter ranges, see MDCR_EL2.HPMN.

Other event counters and PMCCNTR_EL0 are not affected by this field.

When FEAT_PMUv3_ICNTR is implemented, PMICNTR_EL0 is not affected by this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

LC, bit [6]

When FEAT_AA32 is implemented:

Long cycle counter enable. Determines when unsigned overflow is recorded by PMOVSCLR_EL0.C.

LC	Meaning
0b0	Cycle counter overflow on increment that causes unsigned overflow of PMCCNTR_EL0[31:0].
0b1	Cycle counter overflow on increment that causes unsigned overflow of PMCCNTR_EL0[63:0].

When FEAT_EBEP is implemented and the PMU Profiling exception is enabled, the Effective value of this field is 1.

Arm deprecates use of PMCR_EL0.LC = 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

DP, bit [5]

When ((HaveEL(EL3) || (IsFeatureImplemented(FEAT_PMUv3p1) && HaveEL(EL2))) || IsFeatureImplemented(FEAT_PMUv3p7)) || IsFeatureImplemented(FEAT_SPE_DPFZS):

Disable cycle counter when event counting is prohibited.

DP	Meaning
0b0	Cycle counting by PMCCNTR_EL0 is not affected by this mechanism.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

DP	Meaning
0b1	Cycle counting by PMCCNTR_EL0 is disabled in prohibited regions and when event counting is frozen: • If FEAT_PMUv3p1 is implemented, EL2 is implemented, and MDCR_EL2.HPMD is 1, then cycle counting by PMCCNTR_EL0 is disabled at EL2. • If FEAT_SPE_DPFZS is implemented and event counting is frozen by PMCR_EL0.FZS, then cycle counting by PMCCNTR_EL0 is disabled. • If FEAT_PMUv3p7 is implemented and event counting is frozen by PMCR_EL0.FZO, then cycle counting by PMCCNTR_EL0 is disabled. • If FEAT_PMUv3p7 is implemented, EL3 is implemented, and MDCR_EL3.MPMX is 1, then cycle counting by PMCCNTR_EL0 is disabled at EL3. • If EL3 is implemented, MDCR_EL3.SPME is 0, and either FEAT_PMUv3p7 is not implemented or MDCR_EL3.MPMX is 0, then cycle counting by PMCCNTR_EL0 is disabled at EL3 and in Secure state.

The conditions when this field disables the cycle counter are the same as when event counting by an event counter in the first range is prohibited or frozen. For more information about event counter ranges, see MDCR_EL2.HPMN.

If FEAT_PMUv3p7 and FEAT_SPEv1p2 are implemented, meaning PMCR_EL0.FZS is implemented, and FEAT_SPE_DPFZS is not implemented, then cycle counting by PMCCNTR_EL0 is not affected by PMCR_EL0.FZS.

For more information, see ‘Prohibiting event and cycle counting’.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

X, bit [4]

When the implementation includes a PMU event export bus:

Enable export of events in an IMPLEMENTATION DEFINED PMU event export bus.

X	Meaning
0b0	Do not export events.
0b1	Export events where not prohibited.

This field enables the exporting of events over an IMPLEMENTATION DEFINED PMU event export bus to another device.

No events are exported when counting is prohibited.

This field does not affect the generation of Performance Monitors overflow interrupt requests or signaling to a cross-trigger interface (CTI) that can be implemented as signals exported from the PE.

If FEAT_ETE is implemented, this field does not affect the use of PMU events as an External Input by the trace unit.

If FEAT_ETMv4 is implemented, this field does affect the use of PMU events as an External Input by the trace unit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RAZ/WI.

D, bit [3]

When FEAT_AA32 is implemented:

Clock divider.

D	Meaning
0b0	When enabled, PMCCNTR_EL0 counts every clock cycle.
0b1	When enabled, PMCCNTR_EL0 counts once every 64 clock cycles.

If the Effective value of PMCR_EL0.LC is 1, then this field is ignored and the cycle counter counts every clock cycle.

Arm deprecates use of PMCR_EL0.D = 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

C, bit [2]

Cycle counter reset. The effects of writing to this field are:

C	Meaning
0b0	No action.
0b1	Reset PMCCNTR_EL0 to zero.

Note

Resetting PMCCNTR_EL0 does not change the cycle counter overflow field. The value of PMCR_EL0.LC is ignored, and bits [63:0] of the cycle counter are reset.

Access to this field is WO/RAZ.

P, bit [1]

Event counter reset.

P	Meaning
0b0	No action.
0b1	Reset all affected event counters PMEVCNTR<n>_EL0 to zero.

The event counters affected by this field are:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- All event counters in the first range.
- If any of the following are true, all event counters in the second range:
  - EL2 is disabled or not implemented in the current Security state.
  - The PE is executing at EL2 or EL3.

Writes to this field do not affect other event counters, the cycle counter PMCCNTR_EL0, or the instruction counter PMICNTR_EL0.

For more information about event counter ranges, see MDCR_EL2.HPMN.

Note

Resetting the event counters does not change the event counter overflow fields. If FEAT_PMUv3p5 is implemented, the values of MDCR_EL2.HLP and PMCR_EL0.LP are ignored, and bits [63:0] of all affected event counters are reset.

Access to this field is WO/RAZ.

E, bit [0]

Enable.

E	Meaning
0b0	Affected counters are disabled and do not count.
0b1	Affected counters are enabled by PMCNTENSET_EL0.

The counters affected by this field are:

- The event counters in the first range. For more information about event counter ranges, see MDCR_EL2.HPMN.
- If FEAT_PMUv3_ICNTR is implemented, the instruction counter PMICNTR_EL0.
- The cycle counter PMCCNTR_EL0.

Other event counters are not affected by this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Accessing PMCR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMCR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1100	0b000

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' || (IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1')
        ↪then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            if EL2Enabled() && HCR_EL2.TGE == '1' then
                AArch64.SystemAccessTrap(EL2, 0x18);
            else
                AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPMCR == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = PMCR_EL0;
    elsif PSTATE.EL == EL1 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPMCR == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = PMCR_EL0;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = PMCR_EL0;
    elsif PSTATE.EL == EL3 then
        X[t, 64] = PMCR_EL0;
```

```
MSR PMCR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1100	0b000

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' || (IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1')
         ↪then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMCR_EL0 == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMCR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCR_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.PMCR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMCR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCR_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCR_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMCR_EL0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.9 PMECR_EL1, Performance Monitors Extended Control Register (EL1)

The PMECR_EL1 characteristics are:

Purpose

Provides EL1 configuration options for the Performance Monitors.

Configuration

This register is present only when (FEAT_EBEP is implemented or FEAT_PMUv3_SS is implemented) and FEAT_AA64 is implemented. Otherwise, direct accesses to PMECR_EL1 are UNDEFINED.

Attributes

PMECR_EL1 is a 64-bit register.

Field descriptions

Bits [63:5]

Reserved, RES0.

SSE, bits [4:3]

When FEAT_PMUv3_SS is implemented:

Snapshot Enable. Controls the generation of Capture events.

SSE	Meaning
0b00	Capture events are disabled.
0b10	Capture events are enabled and prohibited.
0b11	Capture events are enabled and allowed.

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset:
  - When the highest implemented Exception level is EL1, this field resets to '00'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

KPME, bit [2]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_EBEP is implemented:

Local (Kernel) PMU Exception Enable. Enables PMU Profiling exceptions taken to the current Exception level.

KPME	Meaning
`0b0`	PMU Profiling exceptions taken to the current Exception level are disabled.
`0b1`	PMU Profiling exceptions taken to the current Exception level are not affected by this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PMEE, bits [1:0]

When FEAT_EBEP is implemented:

Performance Monitors Exception Enable. Controls the generation of the **PMUIRQ** signal and the PMU Profiling exception at EL0 and EL1.

PMEE	Meaning
`0b00`	The **PMUIRQ** signal is asserted on a PMU overflow, and the PMU Profiling exception is disabled.
`0b11`	The **PMUIRQ** signal is deasserted, and the PMU Profiling exception is enabled.

All other values are reserved.

This field is ignored by the PE when any of the following are true:

- All of the following are true:
  - EL3 is implemented.
  - MDCR_EL3.PMEE != `0b01`.
- All of the following are true:
  - EL2 is implemented and enabled in the current Security State.
  - MDCR_EL2.PMEE != `0b01`.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'00'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing PMECR_EL1

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, PMECR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1110	0b101

```
if !((IsFeatureImplemented(FEAT_EBEP) || IsFeatureImplemented(FEAT_PMUv3_SS)) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGRTR2_EL2.nPMECR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMECR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMECR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMECR_EL1;
```

```
MSR PMECR_EL1, <Xt>
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1110	0b101

```
if !((IsFeatureImplemented(FEAT_EBEP) || IsFeatureImplemented(FEAT_PMUv3_SS)) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGWTR2_EL2.nPMECR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMECR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMECR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMECR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.10 PMEVCNTR<n>_EL0, Performance Monitors Event Count Registers, n = 0 - 30

The PMEVCNTR<n>_EL0 characteristics are:

Purpose

Holds event counter n, which counts events, where n is 0 to 30.

Configuration

AArch64 System register PMEVCNTR<n>_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMEVCNTR<n>[31:0].

AArch64 System register PMEVCNTR<n>_EL0 bits [31:0] are architecturally mapped to External register PMEVCNTR_EL0[31:0].

When FEAT_PMUv3p5 is implemented, AArch64 System register PMEVCNTR<n>_EL0 bits [63:32] are architecturally mapped to External register PMEVCNTR_EL0[63:32].

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMEVCNTR<n>_EL0 are UNDEFINED.

Attributes

PMEVCNTR<n>_EL0 is a 64-bit register.

Field descriptions

When FEAT_PMUv3p5 is implemented:

EVCNT, bits [63:0]

Event counter n. Value of event counter n, where n is the number of this register and is a number from 0 to 30.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bits [63:32]

Reserved, RES0.

EVCNT, bits [31:0]

Event counter n. Value of event counter n, where n is the number of this register and is a number from 0 to 30.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMEVCNTR<n>_EL0

PMEVCNTR<n>_EL0 can also be accessed by using PMXEVCNTR_EL0 with PMSELR_EL0.SEL set to the value of <n>.

If FEAT_FGT is implemented and <n> is greater than or equal to the number of accessible event counters, then the behavior of permitted reads and writes of PMEVCNTR<n>_EL0 is as follows:

- If <n> is greater than or equal to the Effective value of PMCCR.EPMN, the access is UNDEFINED.
- Otherwise, the access is trapped to EL2.

If FEAT_FGT is not implemented and <n> is greater than or equal to the number of accessible event counters, then reads and writes of PMEVCNTR<n>_EL0 are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP.
- Accesses to the register behave as if <n> is an UNKNOWN value less-than-or-equal-to the index of the highest accessible event counter.
- If EL2 is implemented and enabled in the current Security state, and <n> is less than the number of implemented event counters, accesses from EL1 or permitted accesses from EL0 are trapped to EL2.

Permitted reads and writes of PMEVCNTR<n>_EL0 are RAZ/WI if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- PMUSERENR_EL0.UEN == 1.
- PMUACR_EL1.P<n> == 0.

Permitted writes of PMEVCNTR<n>_EL0 are ignored if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- PMUSERENR_EL0.{UEN,ER} == {1,1}.

Note

In EL0, an access is permitted if it is enabled by PMUSERENR_EL0.{UEN,ER,EN}.
If EL2 is implemented and enabled in the current Security state, in EL1 and EL0, MDCR_EL2.HPMN identifies the number of accessible event counters. Otherwise, the number of accessible event counters is the number of implemented event counters. For more information, see MDCR_EL2.HPMN.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMEVCNTR<m>_EL0 ; Where m = 0-30
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b10:m[4:3]	m[2:0]

```
integer m = UInt(CRm<1:0>:op2<2:0>);

if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif m >= GetNumEventCountersSelfHosted() then
    if IsFeatureImplemented(FEAT_FGT) then
        UNDEFINED;
    else
        ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif (IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<UEN,ER,EN> == '000') ||
         ↪(!IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<ER,EN> == '00') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMEVCNTRn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && m >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1' && PMUACR_EL1[m] == '0' then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = PMEVCNTR_EL0[m];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.PMEVCNTRn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && m >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMEVCNTR_EL0[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMEVCNTR_EL0[m];
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMEVCNTR_EL0[m];
```

MSR PMEVCNTR<m>_EL0, <Xt> ; Where m = 0-30

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b10:m[4:3]	m[2:0]

```
integer m = UInt(CRm<1:0>:op2<2:0>);

if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif m >= GetNumEventCountersSelfHosted() then
    if IsFeatureImplemented(FEAT_FGT) then
        UNDEFINED;
    else
        ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')
         ↪then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMEVCNTRn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && m >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1' && (PMUACR_EL1[m] == '0' ||
         ↪PMUSERENR_EL0.ER == '1') then
        return;
    else
        PMEVCNTR_EL0[m] = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.PMEVCNTRn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif EL2Enabled() && m >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMEVCNTR_EL0[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMEVCNTR_EL0[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMEVCNTR_EL0[m] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.11 PMEVCNTSVR<n>_EL1, Performance Monitors Event Count Saved Value Registers, n = 0 - 30

The PMEVCNTSVR<n>_EL1 characteristics are:

Purpose

Captures the PMU Event counter <n>, PMEVCNTR<n>_EL0.

Configuration

AArch64 System register PMEVCNTSVR<n>_EL1 bits [63:0] are architecturally mapped to External register PMEVCNTSVR_EL1[63:0].

This register is present only when FEAT_PMUv3_SS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMEVCNTSVR<n>_EL1 are UNDEFINED.

Attributes

PMEVCNTSVR<n>_EL1 is a 64-bit register.

Field descriptions

EVCNT, bits [63:0]

Sampled Event Count. The value of PMEVCNTR<n>_EL0 at the last successful Capture event.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing PMEVCNTSVR<n>_EL1

If <n> is greater-than-or-equal-to the Effective value of PMCCR.EPMN, then direct reads of PMEVCNTSVR<n>_EL1 are UNDEFINED.

Otherwise, direct reads of PMEVCNTSVR<n>_EL1 generate a Trap exception to EL2 when all of the following are true:

- <n> is greater-than-or-equal-to the number of snapshot registers accessible at the current Exception level.
- EL2 is implemented and enabled in the current Security state.
- The access is from EL1.

Note

If EL2 is implemented and enabled in the current Security state, MDCR_EL2.HPMN identifies the number of accessible snapshot registers at EL1. Otherwise, the number of accessible snapshot registers is the number of implemented event counters. See MDCR_EL2.HPMN for more details.

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, PMEVCNTSVR<m>_EL1 ; Where m = 0-30
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b1110	0b10:m[4:3]	m[2:0]

```
integer m = UInt(CRm<1:0>:op2<2:0>);

if !(IsFeatureImplemented(FEAT_PMUv3_SS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif m >= GetNumEventCountersSelfHosted() then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMSS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGRTR2_EL2.nPMSSDATA == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && m >= GetNumEventCountersAccessible() then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPMSS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMEVCNTSVR_EL1[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMSS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPMSS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMEVCNTSVR_EL1[m];
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMEVCNTSVR_EL1[m];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.12 PMEVTYPER<n>_EL0, Performance Monitors Event Type Registers, n = 0 - 30

The PMEVTYPER<n>_EL0 characteristics are:

Purpose

Configures event counter n, where n is 0 to 30.

Configuration

AArch64 System register PMEVTYPER<n>_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMEVTYPER<n>[31:0].

AArch64 System register PMEVTYPER<n>_EL0 bits [31:0] are architecturally mapped to External register PMEVTYPER_EL0[31:0].

When FEAT_PMUv3_TH is implemented, or FEAT_PMUv3p8 is implemented, or FEAT_PMUv3_EXT64 is implemented, or FEAT_PMUv3_SME is implemented, AArch64 System register PMEVTYPER<n>_EL0 bits [63:32] are architecturally mapped to External register PMEVTYPER_EL0[63:32].

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMEVTYPER<n>_EL0 are UNDEFINED.

Attributes

PMEVTYPER<n>_EL0 is a 64-bit register.

Field descriptions

TC, bits [63:61]

When (IsFeatureImplemented(FEAT_PMUv3_TH) && ((!IsFeatureImplemented(FEAT_PMUv3_EDGE)) || (PMEVTYPER<n>_EL0.TE == '0'))) && (((!IsFeatureImplemented(FEAT_PMUv3_TH2)) || ((n MOD 2) == 0)) || (PMEVTYPER<n>_EL0.TLC IN {'0x'})):

Threshold Control. Defines the threshold function. In the description of this field:

- V_B[n] is the value the event specified by PMEVTYPER<n>_EL0 would increment event counter n by on a processor cycle if the threshold function is disabled.
- For odd values of n, V[n-1] is the value that event counter n-1 increments by on the same processor cycle. V[n-1] is the result of applying the threshold and edge functions on event counter n-1. If event counter n-1 is disabled, then V[n-1] is zero. V[n-1] is not defined for even values of n.
- TH[n] is the value of PMEVTYPER<n>_EL0.TH.

TC	Meaning
0b000	Not-equal. The counter increments by V_B[n] on each processor cycle when V_B[n] is not equal to TH[n].
0b001	Not-equal, count. The counter increments by 1 on each processor cycle when V_B[n] is not equal to TH[n].

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TC	Meaning
0b010	Equals. The counter increments by V_B[n] on each processor cycle when V_B[n] is equal to TH[n].
0b011	Equals, count. The counter increments by 1 on each processor cycle when V_B[n] is equal to TH[n].
0b100	Greater-than-or-equal. The counter increments by V_B[n] on each processor cycle when V_B[n] is greater than or equal to TH[n].
0b101	Greater-than-or-equal, count. The counter increments by 1 on each processor cycle when V_B[n] is greater than or equal to TH[n].
0b110	Less-than. The counter increments by V_B[n] on each processor cycle when V_B[n] is less than TH[n].
0b111	Less-than, count. The counter increments by 1 on each processor cycle when V_B[n] is less than TH[n].

Comparisons treat V_B[n] and TH[n] as unsigned integer values.

On each processor cycle when the condition specified by PMEVTYPER<n>_EL0.TC[2:1] is true:

- If PMEVTYPER<n>_EL0.TC[0] is 0, then the counter increments by V_B[n].
- If PMEVTYPER<n>_EL0.TC[0] is 1, then the counter increments by 1.

On each processor cycle when the condition specified by PMEVTYPER<n>_EL0.TC[2:1] is false:

- If FEAT_PMUv3_TH2 is implemented, n is odd, and PMEVTYPER<n>_EL0.TLC is `0b01`, then the counter increments by V[n-1].
- Otherwise, the counter does not increment.

If PMEVTYPER<n>_EL0.{TC, TLC, TH} are zero, then the threshold function is disabled.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_AA32EL1 is implemented, this field resets to `'000'`.
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

When FEAT_PMUv3_TH2 is implemented, PMEVTYPER<n>_EL0.TE == '0', (n MOD 2) == 1, and PMEVTYPER<n>_EL0.TLC == '10':

Threshold Control. Defines the threshold function. In the description of this field:

- V_B[n] is the value the event specified by PMEVTYPER<n>_EL0 would increment event counter n by on a processor cycle if the threshold function is disabled.
- V[n-1] is the value that event counter n-1 increments by on the same processor cycle. V[n-1] is the result of applying the threshold and edge functions on event counter n-1. If event counter n-1 is disabled, then V[n-1] is zero.
- TH[n] is the value of PMEVTYPER<n>_EL0.TH.

TC	Meaning
0b000	Not-equal. The counter increments by V[n-1] on each processor cycle when V_B[n] is not equal to TH[n].
0b010	Equals. The counter increments by V[n-1] on each processor cycle when V_B[n] is equal to TH[n].
0b100	Greater-than-or-equal. The counter increments by V[n-1] on each processor cycle when V_B[n] is greater than or equal to TH[n].
0b110	Less-than. The counter increments by V[n-1] on each processor cycle when V_B[n] is less than TH[n].

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

All other values are reserved.

Comparisons treat V_B[n] and TH[n] as unsigned integer values.

On each processor cycle when the condition specified by PMEVTYPER<n>_EL0.TC is true, the counter increments by V[n-1].

On each processor cycle when the condition specified by PMEVTYPER<n>_EL0.TC is false, the counter does not increment.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_AA32EL1 is implemented, this field resets to `'000'`.
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

When FEAT_PMUv3_EDGE is implemented and PMEVTYPER<n>_EL0.TE == '1':

Threshold Control. Defines the threshold function. In the description of this field:

- V_B[n] is the value the event specified by PMEVTYPER<n>_EL0 would increment event counter n by on a processor cycle if the threshold function is disabled.
- For odd values of n, V[n-1] is the value that event counter n-1 increments by on the same processor cycle. V[n-1] is the result of applying the threshold and edge functions on event counter n-1. If event counter n-1 is disabled, then V[n-1] is zero. V[n-1] is not defined for even values of n.
- TH[n] is the value of PMEVTYPER<n>_EL0.TH.

TC	Meaning
`0b001`	Equal to not-equal. The counter increments on each processor cycle when V_B[n] is not equal to TH[n] and V_B[n] was equal to TH[n] on the previous processor cycle.
`0b010`	Equal to/from not-equal. The counter increments on each processor cycle when either: • V_B[n] is not equal to TH[n] and V_B[n] was equal to TH[n] on the previous processor cycle. • V_B[n] is equal to TH[n] and V_B[n] was not equal to TH[n] on the previous processor cycle.
`0b011`	Not-equal to equal. The counter increments on each processor cycle when V_B[n] is equal to TH[n] and V_B[n] was not equal to TH[n] on the previous processor cycle.
`0b101`	Less-than to greater-than-or-equal. The counter increments on each processor cycle when V_B[n] is greater than or equal to TH[n] and V_B[n] was less than TH[n] on the previous processor cycle.
`0b110`	Less-than to/from greater-than-or-equal. The counter increments on each processor cycle when either: • V_B[n] is greater than or equal to TH[n] and V_B[n] was less than TH[n] on the previous processor cycle. • V_B[n] is less than TH[n] and V_B[n] was greater than or equal to TH[n] on the previous processor cycle.
`0b111`	Greater-than-or-equal to less-than. The counter increments on each processor cycle when V_B[n] is less than TH[n] and V_B[n] was greater than or equal to TH[n] on the previous processor cycle.

All other values are reserved.

Comparisons treat V_B[n] and TH[n] as unsigned integer values.

On each processor cycle when the condition specified by PMEVTYPER<n>_EL0.TC is true:

- If FEAT_PMUv3_TH2 is implemented, n is odd, and PMEVTYPER<n>_EL0.TLC is `0b10`, then the counter increments by V[n-1].
- Otherwise, the counter increments by 1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

On each processor cycle when the condition specified by PMEVTYPER<n>_EL0.TC is false, the counter does not increment.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_AA32EL1 is implemented, this field resets to `'000'`.
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TE, bit [60]

When FEAT_PMUv3_EDGE is implemented:

Threshold Edge. Enables the edge condition. When PMEVTYPER<n>_EL0.TE is 1, the event counter increments on cycles when the result of the threshold condition changes. See PMEVTYPER<n>_EL0.TC for more information.

TE	Meaning
`0b0`	Threshold edge condition disabled.
`0b1`	Threshold edge condition enabled.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_AA32EL1 is implemented, this field resets to `'0'`.
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [59]

Reserved, RES0.

SYNC, bit [58]

When FEAT_SEBEP is implemented:

Synchronous mode. Controls whether a PMU Profiling exception generated by the counter is synchronous or asynchronous.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SYNC	Meaning
0b0	Asynchronous PMU Profiling exception is enabled.
0b1	Synchronous PMU Profiling exception is enabled.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VS, bits [57:56]

When FEAT_PMUv3_SME is implemented:

SVE mode filtering. Controls counting events in Streaming and Non-streaming SVE modes.

VS	Meaning
0b00	This mechanism has no effect on the filtering of events.
0b01	The PE does not count events in Streaming SVE mode.
0b10	The PE does not count events in Non-streaming SVE mode.

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TLC, bits [55:54]

When FEAT_PMUv3_TH2 is implemented and (n MOD 2) == 1:

Threshold Linking Control. Extends PMEVTYPER<n>_EL0.TC with additional controls for event linking. See PMEVTYPER<n>_EL0.TC.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TLC	Meaning
0b00	Threshold linking disabled.
0b01	Threshold linking enabled. If the threshold condition described by PMEVTYPER<n>_EL0.TC is false, the counter increments by V[n-1]. Otherwise, the counter increments as described by PMEVTYPER<n>_EL0.TC.
0b10	Threshold linking enabled. If the threshold condition described by PMEVTYPER<n>_EL0.TC is true, the counter increments by V[n-1]. Otherwise, the counter does not increment.

All other values are reserved.

See PMEVTYPER<n>_EL0.TC for more information

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [53:44]

Reserved, RES0.

TH, bits [43:32]

When FEAT_PMUv3_TH is implemented:

Threshold value. Provides the unsigned value for the threshold function defined by PMEVTYPER<n>_EL0.TC.

If PMEVTYPER<n>_EL0.{TC, TH} are both zero and either FEAT_PMUv3_TH2 is not implemented or PMEVTYPER<n>_EL0.TLC is also zero, then the threshold function is disabled.

If PMMIR_EL1.THWIDTH is less than 12, then bits PMEVTYPER<n>_EL0.TH[11:UInt(PMMIR_EL1.THWIDTH)] are RES0. This accounts for the behavior when writing a value greater-than-or-equal-to $2^{\text{UInt(PMMIR_EL1.THWIDTH)}}$.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_AA32EL1 is implemented, this field resets to 0x000.
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

P, bit [31]

EL1 filtering. Controls counting events in EL1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

P	Meaning
0b0	This mechanism has no effect on filtering of events.
0b1	The PE does not count events in EL1.

If Secure and Non-secure states are implemented, then counting events in Non-secure EL1 is further controlled by PMEVTYPER<n>_EL0.NSK.

If FEAT_RME is implemented, then counting events in Realm EL1 is further controlled by PMEVTYPER<n>_EL0.RLK.

If EL3 is implemented, then counting events in EL3 is further controlled by PMEVTYPER<n>_EL0.M.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

U, bit [30]

EL0 filtering. Controls counting events in EL0.

U	Meaning
0b0	This mechanism has no effect on filtering of events.
0b1	The PE does not count events in EL0.

If Secure and Non-secure states are implemented, then counting events in Non-secure EL0 is further controlled by PMEVTYPER<n>_EL0.NSU.

If FEAT_RME is implemented, then counting events in Realm EL0 is further controlled by PMEVTYPER<n>_EL0.RLU.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

NSK, bit [29]

When EL3 is implemented:

Non-secure EL1 filtering. Controls counting events in Non-secure EL1. If PMEVTYPER<n>_EL0.NSK is not equal to PMEVTYPER<n>_EL0.P, then the PE does not count events in Non-secure EL1. Otherwise, this mechanism has no effect on filtering of events in Non-secure EL1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

NSK	Meaning
0b0	When PMEVTYPER<n>_EL0.P == 0, this mechanism has no effect on filtering of events. When PMEVTYPER<n>_EL0.P == 1, the PE does not count events in Non-secure EL1.
0b1	When PMEVTYPER<n>_EL0.P == 0, the PE does not count events in Non-secure EL1. When PMEVTYPER<n>_EL0.P == 1, this mechanism has no effect on filtering of events.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSU, bit [28]

When EL3 is implemented:

Non-secure EL0 filtering. Controls counting events in Non-secure EL0. If PMEVTYPER<n>_EL0.NSU is not equal to PMEVTYPER<n>_EL0.U, then the PE does not count events in Non-secure EL0. Otherwise, this mechanism has no effect on filtering of events in Non-secure EL0.

NSU	Meaning
0b0	When PMEVTYPER<n>_EL0.U == 0, this mechanism has no effect on filtering of events. When PMEVTYPER<n>_EL0.U == 1, the PE does not count events in Non-secure EL0.
0b1	When PMEVTYPER<n>_EL0.U == 0, the PE does not count events in Non-secure EL0. When PMEVTYPER<n>_EL0.U == 1, this mechanism has no effect on filtering of events.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSH, bit [27]

When EL2 is implemented:

EL2 filtering. Controls counting events in EL2.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

NSH	Meaning
0b0	The PE does not count events in EL2.
0b1	This mechanism has no effect on filtering of events.

If EL3 is implemented and FEAT_SEL2 is implemented, then counting events in Secure EL2 is further controlled by PMEVTYPER<n>_EL0.SH.

If FEAT_RME is implemented, then counting events in Realm EL2 is further controlled by PMEVTYPER<n>_EL0.RLH.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

M, bit [26]

When EL3 is implemented:

EL3 filtering. Controls counting events in EL3. If PMEVTYPER<n>_EL0.M is not equal to PMEVTYPER<n>_EL0.P, then the PE does not count events in EL3. Otherwise, this mechanism has no effect on filtering of events in EL3.

M	Meaning
0b0	When PMEVTYPER<n>_EL0.P == 0, this mechanism has no effect on filtering of events. When PMEVTYPER<n>_EL0.P == 1, the PE does not count events in EL3.
0b1	When PMEVTYPER<n>_EL0.P == 0, the PE does not count events in EL3. When PMEVTYPER<n>_EL0.P == 1, this mechanism has no effect on filtering of events.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MT, bit [25]

When FEAT_MTPMU is implemented or an IMPLEMENTATION DEFINED multi-threaded PMU extension is implemented:

Multithreading.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

MT	Meaning
0b0	Count events only on controlling PE.
0b1	Count events from any PE with the same affinity at level 1 and above as this PE.

Unless otherwise stated:

- If the event counts PE cycles when a stall condition is true and a second condition is true, then the counter counts Processor cycles when the stall condition is true for all of these PEs, and the second condition is true for any of these PEs.
- If the event counts PE cycles when any other condition is true, then the counter counts Processor cycles when the condition is true for any of these PEs.
- Otherwise, the event counts by the sum of the count across all of these PEs.

For the stall events, the stall condition means the applicable condition described by the STALL, STALL_FRONTEND, or STALL_BACKEND event.

The second condition is any condition in addition to this.

For example, for the STALL_FRONTEND_L1I event, the stall condition is STALL_FRONTEND, and the second condition is when there is a demand instruction miss in the first level of instruction cache.

For the STALL, STALL_FRONTEND, and STALL_BACKEND events themselves, the second condition is the null TRUE condition.

See 'Multithreaded implementations' and 'Cycle event counting in multithreaded implementations'.

From Armv8.6, the IMPLEMENTATION DEFINED multi-threaded PMU extension is not permitted, meaning if FEAT_MTPMU is not implemented, this field is RES0. See ID_AA64DFR0_EL1.MTPMU.

This field is ignored by the PE and treated as zero when FEAT_MTPMU is implemented and disabled.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SH, bit [24]

When EL3 is implemented and FEAT_SEL2 is implemented:

Secure EL2 filtering. Controls counting events in Secure EL2. If PMEVTYPER<n>_EL0.SH is equal to PMEVTYPER<n>_EL0.NSH, then the PE does not count events in Secure EL2. Otherwise, this mechanism has no effect on filtering of events in Secure EL2.

SH	Meaning
0b0	When PMEVTYPER<n>_EL0.NSH == 0, the PE does not count events in Secure EL2. When PMEVTYPER<n>_EL0.NSH == 1, this mechanism has no effect on filtering of events.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SH	Meaning
0b1	When PMEVTYPER<n>_EL0.NSH == 0, this mechanism has no effect on filtering of events. When PMEVTYPER<n>_EL0.NSH == 1, the PE does not count events in Secure EL2.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

When Secure EL2 is not implemented, access to this field is RES0.

Otherwise:

Reserved, RES0.

Bit [23]

Reserved, RES0.

RLK, bit [22]

When FEAT_RME is implemented:

Realm EL1 filtering. Controls counting events in Realm EL1. If PMEVTYPER<n>_EL0.RLK is not equal to PMEVTYPER<n>_EL0.P, then the PE does not count events in Realm EL1. Otherwise, this mechanism has no effect on filtering of events in Realm EL1.

RLK	Meaning
0b0	When PMEVTYPER<n>_EL0.P == 0, this mechanism has no effect on filtering of events. When PMEVTYPER<n>_EL0.P == 1, the PE does not count events in Realm EL1.
0b1	When PMEVTYPER<n>_EL0.P == 0, the PE does not count events in Realm EL1. When PMEVTYPER<n>_EL0.P == 1, this mechanism has no effect on filtering of events.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLU, bit [21]

When FEAT_RME is implemented:

Realm EL0 filtering. Controls counting events in Realm EL0. If PMEVTYPER<n>_EL0.RLU is not equal to PMEVTYPER<n>_EL0.U, then the PE does not count events in Realm EL0. Otherwise, this mechanism has no effect on filtering of events in Realm EL0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

RLU	Meaning
0b0	When PMEVTYPER<n>_EL0.U == 0, this mechanism has no effect on filtering of events. When PMEVTYPER<n>_EL0.U == 1, the PE does not count events in Realm EL0.
0b1	When PMEVTYPER<n>_EL0.U == 0, the PE does not count events in Realm EL0. When PMEVTYPER<n>_EL0.U == 1, this mechanism has no effect on filtering of events.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLH, bit [20]

When FEAT_RME is implemented:

Realm EL2 filtering. Controls counting events in Realm EL2. If PMEVTYPER<n>_EL0.RLH is equal to PMEVTYPER<n>_EL0.NSH, then the PE does not count events in Realm EL2. Otherwise, this mechanism has no effect on filtering of events in Realm EL2.

RLH	Meaning
0b0	When PMEVTYPER<n>_EL0.NSH == 0, the PE does not count events in Realm EL2. When PMEVTYPER<n>_EL0.NSH == 1, this mechanism has no effect on filtering of events.
0b1	When PMEVTYPER<n>_EL0.NSH == 0, this mechanism has no effect on filtering of events. When PMEVTYPER<n>_EL0.NSH == 1, the PE does not count events in Realm EL2.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [19:16]

Reserved, RES0.

evtCount[15:10], bits [15:10]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_PMUv3p1 is implemented:

Extension to evtCount[9:0]. For more information, see evtCount[9:0].

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

evtCount[9:0], bits [9:0]

Event to count.

The event number of the event that is counted by event counter PMEVCNTR<n>_EL0.

The ranges of event numbers allocated to each type of event are shown in 'Allocation of the PMU event number space'.

If FEAT_PMUv3p8 is implemented and PMEVTYPER<n>_EL0.evtCount is programmed to an event that is reserved or not supported by the PE, no events are counted and the value returned by a direct or external read of the PMEVTYPER<n>_EL0.evtCount field is the value written to the field.

Note

Arm recommends this behavior for all implementations of FEAT_PMUv3.

Otherwise, if PMEVTYPER<n>_EL0.evtCount is programmed to an event that is reserved or not supported by the PE, the behavior depends on the value written:

- For the range `0x0000` to `0x003F`, no events are counted and the value returned by a direct or external read of the PMEVTYPER<n>_EL0.evtCount field is the value written to the field.
- If FEAT_PMUv3p1 is implemented, for the range `0x4000` to `0x403F`, no events are counted and the value returned by a direct or external read of the PMEVTYPER<n>_EL0.evtCount field is the value written to the field.
- For other values, it is UNPREDICTABLE what event, if any, is counted and the value returned by a direct or external read of the PMEVTYPER<n>_EL0.evtCount field is UNKNOWN.

Note

UNPREDICTABLE means the event must not expose privileged information.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing PMEVTYPER<n>_EL0

PMEVTYPER<n>_EL0 can also be accessed by using PMXEVTYPER_EL0 with PMSELR_EL0.SEL set to n.

If FEAT_FGT is implemented and <n> is greater than or equal to the number of accessible event counters, then the behavior of permitted reads and writes of PMEVTYPER<n>_EL0 is as follows:

- If <n> is greater than or equal to the Effective value of PMCCR.EPMN, the access is UNDEFINED.
- Otherwise, the access is trapped to EL2.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

If FEAT_FGT is not implemented and <n> is greater than or equal to the number of accessible event counters, then reads and writes of PMEVTYPER<n>_EL0 are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP.
- Accesses to the register behave as if <n> is an UNKNOWN value less-than-or-equal-to the index of the highest accessible event counter.
- If EL2 is implemented and enabled in the current Security state, and <n> is less than the number of implemented event counters, accesses from EL1 or permitted accesses from EL0 are trapped to EL2.

Permitted reads and writes of PMEVTYPER<n>_EL0 are RAZ/WI if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- PMUSERENR_EL0.UEN == 1.
- PMUACR_EL1.P<n> == 0.

Permitted writes of PMEVTYPER<n>_EL0 are ignored if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- PMUSERENR_EL0.{UEN,ER} == {1,1}.

Note

In EL0, an access is permitted if it is enabled by PMUSERENR_EL0.{UEN,EN}.
If EL2 is implemented and enabled in the current Security state, in EL1 and EL0, MDCR_EL2.HPMN identifies the number of accessible event counters. Otherwise, the number of accessible event counters is the number of implemented event counters. For more information, see MDCR_EL2.HPMN.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMEVTYPER<m>_EL0 ; Where m = 0-30
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b11:m[4:3]	m[2:0]

```
integer m = UInt(CRm<1:0>:op2<2:0>);

if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif m >= GetNumEventCountersSelfHosted() then
    if IsFeatureImplemented(FEAT_FGT) then
        UNDEFINED;
    else
        ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')
         ↪then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMEVTYPERn_EL0 == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && m >= GetNumEventCountersAccessible() then
            if !IsFeatureImplemented(FEAT_FGT) then
                ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
            else
                AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1' && PMUACR_EL1[m] == '0' then
            X[t, 64] = Zeros(64);
        else
            X[t, 64] = PMEVTYPER_EL0[m];
    elsif PSTATE.EL == EL1 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
             ↪HDFGRTR_EL2.PMEVTYPERn_EL0 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && m >= GetNumEventCountersAccessible() then
            if !IsFeatureImplemented(FEAT_FGT) then
                ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
            else
                AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = PMEVTYPER_EL0[m];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = PMEVTYPER_EL0[m];
    elsif PSTATE.EL == EL3 then
        X[t, 64] = PMEVTYPER_EL0[m];
```

```
MSR PMEVTYPER<m>_EL0, <Xt> ; Where m = 0-30
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b11:m[4:3]	m[2:0]

```
integer m = UInt(CRm<1:0>:op2<2:0>);

if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif m >= GetNumEventCountersSelfHosted() then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    if IsFeatureImplemented(FEAT_FGT) then
        UNDEFINED;
    else
        ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')
         ↪then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMEVTYPERn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && m >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1' && (PMUACR_EL1[m] == '0' ||
         ↪PMUSERENR_EL0.ER == '1') then
        return;
    else
        PMEVTYPER_EL0[m] = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.PMEVTYPERn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && m >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMEVTYPER_EL0[m] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMEVTYPER_EL0[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMEVTYPER_EL0[m] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.13 PMIAR_EL1, Performance Monitors Instruction Address Register

The PMIAR_EL1 characteristics are:

Purpose

Captures the address of the instruction generating a PMU Profiling exception.

Configuration

This register is present only when FEAT_SEBEP is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMIAR_EL1 are UNDEFINED.

Attributes

PMIAR_EL1 is a 64-bit register.

Field descriptions

ADDRESS, bits [63:0]

Instruction virtual address.

For writes to PMIAR_EL1, PMIAR_EL1.ADDRESS[63:P] is RESS. P is defined as:

- 56, when FEAT_LVA3 is implemented.
- 52, when FEAT_LVA is implemented.
- 48, otherwise.

PMIAR_EL1.ADDRESS[1:0] is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMIAR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMIAR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1110	0b111

```
if !(IsFeatureImplemented(FEAT_SEBEP) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGRTR2_EL2.nPMIAR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMIAR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMIAR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMIAR_EL1;
```

```
MSR PMIAR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1110	0b111

```
if !(IsFeatureImplemented(FEAT_SEBEP) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGWTR2_EL2.nPMIAR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMIAR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMIAR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMIAR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.14 PMICFILTR_EL0, Performance Monitors Instruction Counter Filter Register

The PMICFILTR_EL0 characteristics are:

Purpose

Configures the Instruction Counter.

Configuration

AArch64 System register PMICFILTR_EL0 bits [63:0] are architecturally mapped to External register PMICFILTR_EL0[63:0].

This register is present only when FEAT_PMUv3_ICNTR is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMICFILTR_EL0 are UNDEFINED.

Attributes

PMICFILTR_EL0 is a 64-bit register.

Field descriptions

Bits [63:59]

Reserved, RES0.

SYNC, bit [58]

When FEAT_SEBEP is implemented:

Synchronous mode. Controls whether a PMU Profiling exception generated by the counter is synchronous or asynchronous.

SYNC	Meaning
0b0	Asynchronous PMU Profiling exception is enabled.
0b1	Synchronous PMU Profiling exception is enabled.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

VS, bits [57:56]

When FEAT_PMUv3_SME is implemented:

SVE mode filtering. Controls counting instructions in Streaming and Non-streaming SVE modes.

VS	Meaning
0b00	This mechanism has no effect on the filtering of instructions.
0b01	The PE does not count instructions in Streaming SVE mode.
0b10	The PE does not count instructions in Non-streaming SVE mode.

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [55:32]

Reserved, RES0.

P, bit [31]

EL1 filtering. Controls counting instructions in EL1.

P	Meaning
0b0	This mechanism has no effect on filtering of instructions.
0b1	The PE does not count instructions in EL1.

If Secure and Non-secure states are implemented, then counting instructions in Non-secure EL1 is further controlled by PMICFILTR_EL0.NSK.

If FEAT_RME is implemented, then counting instructions in Realm EL1 is further controlled by PMICFILTR_EL0.RLK.

If EL3 is implemented, then counting instructions in EL3 is further controlled by PMICFILTR_EL0.M.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

U, bit [30]

EL0 filtering. Controls counting instructions in EL0.

U	Meaning
0b0	This mechanism has no effect on filtering of instructions.
0b1	The PE does not count instructions in EL0.

If Secure and Non-secure states are implemented, then counting instructions in Non-secure EL0 is further controlled by PMICFILTR_EL0.NSU.

If FEAT_RME is implemented, then counting instructions in Realm EL0 is further controlled by PMICFILTR_EL0.RLU.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

NSK, bit [29]

When EL3 is implemented:

Non-secure EL1 filtering. Controls counting instructions in Non-secure EL1. If PMICFILTR_EL0.NSK is not equal to PMICFILTR_EL0.P, then the PE does not count instructions in Non-secure EL1. Otherwise, this mechanism has no effect on filtering of instructions in Non-secure EL1.

NSK	Meaning
0b0	When PMICFILTR_EL0.P == 0, this mechanism has no effect on filtering of instructions. When PMICFILTR_EL0.P == 1, the PE does not count instructions in Non-secure EL1.
0b1	When PMICFILTR_EL0.P == 0, the PE does not count instructions in Non-secure EL1. When PMICFILTR_EL0.P == 1, this mechanism has no effect on filtering of instructions.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSU, bit [28]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When EL3 is implemented:

Non-secure EL0 filtering. Controls counting instructions in Non-secure EL0. If PMICFILTR_EL0.NSU is not equal to PMICFILTR_EL0.U, then the PE does not count instructions in Non-secure EL0. Otherwise, this mechanism has no effect on filtering of instructions in Non-secure EL0.

NSU	Meaning
`0b0`	When PMICFILTR_EL0.U == 0, this mechanism has no effect on filtering of instructions. When PMICFILTR_EL0.U == 1, the PE does not count instructions in Non-secure EL0.
`0b1`	When PMICFILTR_EL0.U == 0, the PE does not count instructions in Non-secure EL0. When PMICFILTR_EL0.U == 1, this mechanism has no effect on filtering of instructions.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSH, bit [27]

When EL2 is implemented:

EL2 filtering. Controls counting instructions in EL2.

NSH	Meaning
`0b0`	The PE does not count instructions in EL2.
`0b1`	This mechanism has no effect on filtering of instructions.

If EL3 is implemented and FEAT_SEL2 is implemented, then counting instructions in Secure EL2 is further controlled by PMICFILTR_EL0.SH.

If FEAT_RME is implemented, then counting instructions in Realm EL2 is further controlled by PMICFILTR_EL0.RLH.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

M, bit [26]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When EL3 is implemented:

EL3 filtering. Controls counting instructions in EL3. If PMICFILTR_EL0.M is not equal to PMICFILTR_EL0.P, then the PE does not count instructions in EL3. Otherwise, this mechanism has no effect on filtering of instructions in EL3.

M	Meaning
0b0	When PMICFILTR_EL0.P == 0, this mechanism has no effect on filtering of instructions. When PMICFILTR_EL0.P == 1, the PE does not count instructions in EL3.
0b1	When PMICFILTR_EL0.P == 0, the PE does not count instructions in EL3. When PMICFILTR_EL0.P == 1, this mechanism has no effect on filtering of instructions.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [25]

Reserved, RES0.

SH, bit [24]

When EL3 is implemented and FEAT_SEL2 is implemented:

Secure EL2 filtering. Controls counting instructions in Secure EL2. If PMICFILTR_EL0.SH is equal to PMICFILTR_EL0.NSH, then the PE does not count instructions in Secure EL2. Otherwise, this mechanism has no effect on filtering of instructions in Secure EL2.

SH	Meaning
0b0	When PMICFILTR_EL0.NSH == 0, the PE does not count instructions in Secure EL2. When PMICFILTR_EL0.NSH == 1, this mechanism has no effect on filtering of instructions.
0b1	When PMICFILTR_EL0.NSH == 0, this mechanism has no effect on filtering of instructions. When PMICFILTR_EL0.NSH == 1, the PE does not count instructions in Secure EL2.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

When Secure EL2 is not implemented, access to this field is RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

Bit [23]

Reserved, RES0.

RLK, bit [22]

When FEAT_RME is implemented:

Realm EL1 filtering. Controls counting instructions in Realm EL1. If PMICFILTR_EL0.RLK is not equal to PMICFILTR_EL0.P, then the PE does not count instructions in Realm EL1. Otherwise, this mechanism has no effect on filtering of instructions in Realm EL1.

RLK	Meaning
0b0	When PMICFILTR_EL0.P == 0, this mechanism has no effect on filtering of instructions. When PMICFILTR_EL0.P == 1, the PE does not count instructions in Realm EL1.
0b1	When PMICFILTR_EL0.P == 0, the PE does not count instructions in Realm EL1. When PMICFILTR_EL0.P == 1, this mechanism has no effect on filtering of instructions.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLU, bit [21]

When FEAT_RME is implemented:

Realm EL0 filtering. Controls counting instructions in Realm EL0. If PMICFILTR_EL0.RLU is not equal to PMICFILTR_EL0.U, then the PE does not count instructions in Realm EL0. Otherwise, this mechanism has no effect on filtering of instructions in Realm EL0.

RLU	Meaning
0b0	When PMICFILTR_EL0.U == 0, this mechanism has no effect on filtering of instructions. When PMICFILTR_EL0.U == 1, the PE does not count instructions in Realm EL0.
0b1	When PMICFILTR_EL0.U == 0, the PE does not count instructions in Realm EL0. When PMICFILTR_EL0.U == 1, this mechanism has no effect on filtering of instructions.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLH, bit [20]

When FEAT_RME is implemented:

Realm EL2 filtering. Controls counting instructions in Realm EL2. If PMICFILTR_EL0.RLH is equal to PMICFILTR_EL0.NSH, then the PE does not count instructions in Realm EL2. Otherwise, this mechanism has no effect on filtering of instructions in Realm EL2.

RLH	Meaning
`0b0`	When PMICFILTR_EL0.NSH == 0, the PE does not count instructions in Realm EL2. When PMICFILTR_EL0.NSH == 1, this mechanism has no effect on filtering of instructions.
`0b1`	When PMICFILTR_EL0.NSH == 0, this mechanism has no effect on filtering of instructions. When PMICFILTR_EL0.NSH == 1, the PE does not count instructions in Realm EL2.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [19:16]

Reserved, RES0.

evtCount, bits [15:0]

Event to count.

Reads as `0x0008`

Access to this field is RO.

Accessing PMICFILTR_EL0

Permitted reads and writes of PMICFILTR_EL0 are RAZ/WI if all of the following are true:

- PSTATE.EL == EL0.
- PMUACR_EL1.F0 == 0.

Permitted writes of PMICFILTR_EL0 are ignored if all of the following are true:

- PSTATE.EL == EL0.
- PMUSERENR_EL0.IR == 1.

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, PMICFILTR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b0110	0b000

```
if !(IsFeatureImplemented(FEAT_PMUv3_ICNTR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.UEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
     ↪SCR_EL3.FGTEn2 == '0') || HDFGRTR2_EL2.nPMICFILTR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1' && PMUACR_EL1.F0 == '0' then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = PMICFILTR_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
     ↪HDFGRTR2_EL2.nPMICFILTR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMICFILTR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMICFILTR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMICFILTR_EL0;
```

```
MSR PMICFILTR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b0110	0b000

```
if !(IsFeatureImplemented(FEAT_PMUv3_ICNTR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.UEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
         ↪SCR_EL3.FGTEn2 == '0') || HDFGWTR2_EL2.nPMICFILTR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1' && (PMUACR_EL1.F0 == '0' ||
         ↪PMUSERENR_EL0.IR == '1') then
        return;
    else
        PMICFILTR_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGWTR2_EL2.nPMICFILTR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMICFILTR_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMICFILTR_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMICFILTR_EL0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.15 PMICNTR_EL0, Performance Monitors Instruction Counter Register

The PMICNTR_EL0 characteristics are:

Purpose

If event counting is not prohibited and the instruction counter is enabled, the counter increments for each architecturally-executed instruction, according to the configuration specified by PMICFILTR_EL0.

Configuration

AArch64 System register PMICNTR_EL0 bits [63:0] are architecturally mapped to External register PMICNTR_EL0[63:0].

This register is present only when FEAT_PMUv3_ICNTR is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMICNTR_EL0 are UNDEFINED.

Attributes

PMICNTR_EL0 is a 64-bit register.

Field descriptions

ICNT, bits [63:0]

Instruction Counter.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing PMICNTR_EL0

PMICNTR_EL0 treats permitted reads-as-zero and ignores permitted writes if all of the following are true:

- PSTATE.EL == EL0.
- PMUACR_EL1.F0 == 0.

PMICNTR_EL0 ignores permitted writes if all of the following are true:

- PSTATE.EL == EL0.
- PMUSERENR_EL0.IR == 1.

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, PMICNTR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_PMUv3_ICNTR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.UEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
     ↪SCR_EL3.FGTEn2 == '0') || HDFGRTR2_EL2.nPMICNTR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1' && PMUACR_EL1.F0 == '0' then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = PMICNTR_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
     ↪HDFGRTR2_EL2.nPMICNTR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMICNTR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMICNTR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMICNTR_EL0;
```

```
MSR PMICNTR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_PMUv3_ICNTR) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.UEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
         ↪SCR_EL3.FGTEn2 == '0') || HDFGWTR2_EL2.nPMICNTR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1' && (PMUACR_EL1.F0 == '0' ||
         ↪PMUSERENR_EL0.IR == '1') then
        return;
    else
        PMICNTR_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGWTR2_EL2.nPMICNTR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMICNTR_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMICNTR_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMICNTR_EL0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.16 PMICNTSVR_EL1, Performance Monitors Instruction Count Saved Value Register

The PMICNTSVR_EL1 characteristics are:

Purpose

Captures the PMU Instruction counter, PMICNTR_EL0.

Configuration

AArch64 System register PMICNTSVR_EL1 bits [63:0] are architecturally mapped to External register PMICNTSVR_EL1[63:0].

This register is present only when FEAT_PMUv3_ICNTR is implemented, FEAT_PMUv3_SS is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to PMICNTSVR_EL1 are UNDEFINED.

Attributes

PMICNTSVR_EL1 is a 64-bit register.

Field descriptions

ICNT, bits [63:0]

Sampled Instruction Count. The value of PMICNTR_EL0 at the last successful Capture event.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing PMICNTSVR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMICNTSVR_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b1110	0b1100	0b000

```
if !(IsFeatureImplemented(FEAT_PMUv3_ICNTR) && IsFeatureImplemented(FEAT_PMUv3_SS) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMSS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGRTR2_EL2.nPMSSDATA == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPMSS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMICNTSVR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMSS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPMSS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMICNTSVR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMICNTSVR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.17 PMINTENCLR_EL1, Performance Monitors Interrupt Enable Clear Register

The PMINTENCLR_EL1 characteristics are:

Purpose

Allows software to disable the generation of interrupt requests or, when FEAT_EBEP is implemented, PMU Profiling exceptions on overflows from the following counters:

- The cycle counter PMCCNTR_EL0.
- The event counters PMEVCNTR<n>_EL0.
- When FEAT_PMUv3_ICNTR is implemented, the instruction counter PMICNTR_EL0.

Reading from this register shows which overflow interrupt requests or PMU Profiling exceptions are enabled.

Configuration

AArch64 System register PMINTENCLR_EL1 bits [63:0] are architecturally mapped to AArch64 System register PMINTENSET_EL1[63:0].

AArch64 System register PMINTENCLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register PMINTENCLR[31:0].

AArch64 System register PMINTENCLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register PMINTENSET[31:0].

AArch64 System register PMINTENCLR_EL1 bits [31:0] are architecturally mapped to External register PMINTENCLR_EL1[31:0].

AArch64 System register PMINTENCLR_EL1 bits [31:0] are architecturally mapped to External register PMINTENSET_EL1[31:0].

When FEAT_PMUv3p9 is implemented or FEAT_PMUv3_EXT64 is implemented, AArch64 System register PMINTENCLR_EL1 bits [63:32] are architecturally mapped to External register PMINTENCLR_EL1[63:32].

When FEAT_PMUv3p9 is implemented or FEAT_PMUv3_EXT64 is implemented, AArch64 System register PMINTENCLR_EL1 bits [63:32] are architecturally mapped to External register PMINTENSET_EL1[63:32].

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMINTENCLR_EL1 are UNDEFINED.

Attributes

PMINTENCLR_EL1 is a 64-bit register.

Field descriptions

Bits [63:33]

Reserved, RES0.

F0, bit [32]

When FEAT_PMUv3_ICNTR is implemented:

Interrupt request or PMU Profiling exception on unsigned overflow of PMICNTR_EL0 disable. On writes, allows software to disable the interrupt request or PMU Profiling exception on unsigned overflow of

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

PMICNTR_EL0. On reads, returns the interrupt request or PMU Profiling exception on unsigned overflow of PMICNTR_EL0 enable status.

F0	Meaning
0b0	Interrupt request or PMU Profiling exception on unsigned overflow of PMICNTR_EL0 disabled.
0b1	Interrupt request or PMU Profiling exception on unsigned overflow of PMICNTR_EL0 enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing this field has the following behavior:

- Access to this field is RAZ/WI if all of the following are true:
  - EL3 is implemented
  - PSTATE.EL != EL3
  - MDCR_EL3.EnPM2 == ‘0’
- Access to this field is RAZ/WI if all of the following are true:
  - FEAT_FGT2 is implemented
  - EL2Enabled()
  - PSTATE.EL == EL1
  - Any of the following are true:
    - All of the following are true:
      - EL3 is implemented
      - SCR_EL3.FGTEn2 == ‘0’
    - All of the following are true:
      - HDFGRTR2_EL2.nPMICFILTR_EL0 == ‘0’
      - HDFGWTR2_EL2.nPMICFILTR_EL0 == ‘0’
- Access to this field is WO/RAZ if all of the following are true:
  - FEAT_FGT2 is implemented
  - EL2Enabled()
  - PSTATE.EL == EL1
  - Any of the following are true:
    - All of the following are true:
      - EL3 is implemented
      - SCR_EL3.FGTEn2 == ‘0’
    - HDFGRTR2_EL2.nPMICFILTR_EL0 == ‘0’
- Access to this field is RO if all of the following are true:
  - FEAT_FGT2 is implemented
  - EL2Enabled()
  - PSTATE.EL == EL1
  - Any of the following are true:
    - All of the following are true:
      - EL3 is implemented
      - SCR_EL3.FGTEn2 == ‘0’
    - HDFGWTR2_EL2.nPMICFILTR_EL0 == ‘0’
- Otherwise, access to this field is W1C.

Otherwise:

Reserved, RES0.

C, bit [31]

Interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0 disable. On writes, allows software to disable the interrupt request or PMU Profiling exception on unsigned overflow of

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

PMCCNTR_EL0. On reads, returns the interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0 enable status.

C	Meaning
0b0	Interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0 disabled.
0b1	Interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Access to this field is W1C.

P<m> , bits [m], for m = 30 to 0

Interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0 disable. On writes, allows software to disable the interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0. On reads, returns the interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0 enable status.

P<m>	Meaning
0b0	Interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0 disabled.
0b1	Interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= GetNumEventCountersAccessible(), access to this field is RAZ/WI.
- Otherwise, access to this field is W1C.

Accessing PMINTENCLR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMINTENCLR_EL1
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1110	0b010

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.PMINTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMINTENCLR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMINTENCLR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMINTENCLR_EL1;
```

```
MSR PMINTENCLR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1110	0b010

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.PMINTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        PMINTENCLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMINTENCLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMINTENCLR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.18 PMINTENSET_EL1, Performance Monitors Interrupt Enable Set Register

The PMINTENSET_EL1 characteristics are:

Purpose

Allows software to enable the generation of interrupt requests or, when FEAT_EBEP is implemented, PMU Profiling exceptions on overflows from the following counters:

- The cycle counter PMCCNTR_EL0.
- The event counters PMEVCNTR<n>_EL0.
- When FEAT_PMUv3_ICNTR is implemented, the instruction counter PMICNTR_EL0.

Reading from this register shows which overflow interrupt requests or PMU Profiling exceptions are enabled.

Configuration

AArch64 System register PMINTENSET_EL1 bits [63:0] are architecturally mapped to AArch64 System register PMINTENCLR_EL1[63:0].

AArch64 System register PMINTENSET_EL1 bits [31:0] are architecturally mapped to AArch32 System register PMINTENSET[31:0].

AArch64 System register PMINTENSET_EL1 bits [31:0] are architecturally mapped to AArch32 System register PMINTENCLR[31:0].

AArch64 System register PMINTENSET_EL1 bits [31:0] are architecturally mapped to External register PMINTENSET_EL1[31:0].

AArch64 System register PMINTENSET_EL1 bits [31:0] are architecturally mapped to External register PMINTENCLR_EL1[31:0].

When FEAT_PMUv3p9 is implemented or FEAT_PMUv3_EXT64 is implemented, AArch64 System register PMINTENSET_EL1 bits [63:32] are architecturally mapped to External register PMINTENSET_EL1[63:32].

When FEAT_PMUv3p9 is implemented or FEAT_PMUv3_EXT64 is implemented, AArch64 System register PMINTENSET_EL1 bits [63:32] are architecturally mapped to External register PMINTENCLR_EL1[63:32].

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMINTENSET_EL1 are UNDEFINED.

Attributes

PMINTENSET_EL1 is a 64-bit register.

Field descriptions

Bits [63:33]

Reserved, RES0.

F0, bit [32]

When FEAT_PMUv3_ICNTR is implemented:

Interrupt request or PMU Profiling exception on unsigned overflow of PMICNTR_EL0 enable. On writes, allows software to enable the interrupt request or PMU Profiling exception on unsigned overflow of

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

PMICNTR_EL0. On reads, returns the interrupt request or PMU Profiling exception on unsigned overflow of PMICNTR_EL0 enable status.

F0	Meaning
0b0	Interrupt request or PMU Profiling exception on unsigned overflow of PMICNTR_EL0 disabled.
0b1	Interrupt request or PMU Profiling exception on unsigned overflow of PMICNTR_EL0 enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing this field has the following behavior:

- Access to this field is RAZ/WI if all of the following are true:
  - EL3 is implemented
  - PSTATE.EL != EL3
  - MDCR_EL3.EnPM2 == ‘0’
- Access to this field is RAZ/WI if all of the following are true:
  - FEAT_FGT2 is implemented
  - EL2Enabled()
  - PSTATE.EL == EL1
  - Any of the following are true:
    - All of the following are true:
      - EL3 is implemented
      - SCR_EL3.FGTEn2 == ‘0’
    - All of the following are true:
      - HDFGRTR2_EL2.nPMICFILTR_EL0 == ‘0’
      - HDFGWTR2_EL2.nPMICFILTR_EL0 == ‘0’
- Access to this field is WO/RAZ if all of the following are true:
  - FEAT_FGT2 is implemented
  - EL2Enabled()
  - PSTATE.EL == EL1
  - Any of the following are true:
    - All of the following are true:
      - EL3 is implemented
      - SCR_EL3.FGTEn2 == ‘0’
    - HDFGRTR2_EL2.nPMICFILTR_EL0 == ‘0’
- Access to this field is RO if all of the following are true:
  - FEAT_FGT2 is implemented
  - EL2Enabled()
  - PSTATE.EL == EL1
  - Any of the following are true:
    - All of the following are true:
      - EL3 is implemented
      - SCR_EL3.FGTEn2 == ‘0’
    - HDFGWTR2_EL2.nPMICFILTR_EL0 == ‘0’
- Otherwise, access to this field is W1S.

Otherwise:

Reserved, RES0.

C, bit [31]

Interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0 enable. On writes, allows software to enable the interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

On reads, returns the interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0 enable status.

C	Meaning
0b0	Interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0 disabled.
0b1	Interrupt request or PMU Profiling exception on unsigned overflow of PMCCNTR_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Access to this field is W1S.

P<m> , bits [m], for m = 30 to 0

Interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0 enable. On writes, allows software to enable the interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0. On reads, returns the interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0 enable status.

P<m>	Meaning
0b0	Interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0 disabled.
0b1	Interrupt request or PMU Profiling exception on unsigned overflow of PMEVCNTR<m>_EL0 enabled.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= GetNumEventCountersAccessible(), access to this field is RAZ/WI.
- Otherwise, access to this field is W1S.

Accessing PMINTENSET_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMINTENSET_EL1
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1110	0b001

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.PMINTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMINTENSET_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMINTENSET_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMINTENSET_EL1;
```

```
MSR PMINTENSET_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1110	0b001

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.PMINTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        PMINTENSET_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMINTENSET_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMINTENSET_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.19 PMMIR_EL1, Performance Monitors Machine Identification Register

The PMMIR_EL1 characteristics are:

Purpose

Describes Performance Monitors parameters specific to the implementation to software.

Configuration

AArch64 System register PMMIR_EL1 bits [31:0] are architecturally mapped to AArch32 System register PMMIR[31:0].

When FEAT_PMUv3_EXT is implemented and FEAT_PMUv3p4 is implemented, AArch64 System register PMMIR_EL1 bits [31:0] are architecturally mapped to External register PMMIR[31:0].

When (IsFeatureImplemented(FEAT_PMUv3_EXT) && IsFeatureImplemented(FEAT_PMUv3p4)) && (IsFeatureImplemented(FEAT_PMUv3_EXT64) || IsFeatureImplemented(FEAT_PMUv3p9)), AArch64 System register PMMIR_EL1 bits [63:32] are architecturally mapped to External register PMMIR[63:32].

This register is present only when FEAT_PMUv3p4 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMMIR_EL1 are UNDEFINED.

Attributes

PMMIR_EL1 is a 64-bit register.

Field descriptions

Bits [63:29]

Reserved, RES0.

SME, bit [28]

PMUv3 for SME. Adds support for the Streaming SVE mode filter.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SME	Meaning
0b0	Streaming SVE mode filter not implemented.
0b1	Adds support for the Streaming SVE mode filter.

All other values are reserved.

FEAT_PMUv3_SME implements the functionality identified by the value 1.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EDGE, bits [27:24]

PMU event edge detection. With PMMIR_EL1.THWIDTH, indicates implementation of event counter thresholding features.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EDGE	Meaning
0b0000	FEAT_PMUv3_EDGE is not implemented.
0b0001	FEAT_PMUv3_EDGE is implemented.
0b0010	As 0b0001, and adds support for threshold value linking between a pair of counters.

All other values are reserved.

If FEAT_PMUv3_TH is not implemented, the only permitted value is 0b0000.

FEAT_PMUv3_EDGE implements the functionality identified by the value 0b0001.

FEAT_PMUv3_TH2 implements the functionality identified by the value 0b0010.

Access to this field is RO.

THWIDTH, bits [23:20]

PMEVTYPER<n>_EL0.TH width. Indicates implementation of the FEAT_PMUv3_TH feature, and, if implemented, the size of the PMEVTYPER<n>_EL0.TH field.

The value of this field is an IMPLEMENTATION DEFINED choice of:

THWIDTH	Meaning
0b0000	FEAT_PMUv3_TH is not implemented.
0b0001	1 bit. PMEVTYPER<n>_EL0.TH[11:1] are RES0.
0b0010	2 bits. PMEVTYPER<n>_EL0.TH[11:2] are RES0.
0b0011	3 bits. PMEVTYPER<n>_EL0.TH[11:3] are RES0.
0b0100	4 bits. PMEVTYPER<n>_EL0.TH[11:4] are RES0.
0b0101	5 bits. PMEVTYPER<n>_EL0.TH[11:5] are RES0.
0b0110	6 bits. PMEVTYPER<n>_EL0.TH[11:6] are RES0.
0b0111	7 bits. PMEVTYPER<n>_EL0.TH[11:7] are RES0.
0b1000	8 bits. PMEVTYPER<n>_EL0.TH[11:8] are RES0.
0b1001	9 bits. PMEVTYPER<n>_EL0.TH[11:9] are RES0.
0b1010	10 bits. PMEVTYPER<n>_EL0.TH[11:10] are RES0.
0b1011	11 bits. PMEVTYPER<n>_EL0.TH[11] is RES0.
0b1100	12 bits.

All other values are reserved.

If FEAT_PMUv3_TH is not implemented, this field is zero.

Otherwise, the largest value that can be written to PMEVTYPER<n>_EL0.TH is $2^{(\text{PMMIR_EL1.THWIDTH})}$ minus one.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

BUS_WIDTH, bits [19:16]

Bus width. Indicates the number of bytes each BUS_ACCESS event relates to. Encoded as Log_2(number of bytes), plus one.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BUS_WIDTH	Meaning
`0b0000`	The information is not available.
`0b0011`	Four bytes.
`0b0100`	8 bytes.
`0b0101`	16 bytes.
`0b0110`	32 bytes.
`0b0111`	64 bytes.
`0b1000`	128 bytes.
`0b1001`	256 bytes.
`0b1010`	512 bytes.
`0b1011`	1024 bytes.
`0b1100`	2048 bytes.

All other values are reserved.

Each transfer is up to this number of bytes. An access might be smaller than the bus width.

When this field is nonzero, each access counted by BUS_ACCESS is at most BUS_WIDTH bytes. An implementation might treat a wide bus as multiple narrower buses, such that a wide access on the bus increments the BUS_ACCESS counter by more than one.

Access to this field is RO.

BUS_SLOTS, bits [15:8]

Bus count. The largest value by which the BUS_ACCESS event might increment in a single BUS_CYCLES cycle.

This field has an IMPLEMENTATION DEFINED value.

When this field is nonzero, the largest value by which the BUS_ACCESS event might increment in a single BUS_CYCLES cycle is BUS_SLOTS.

If the bus count information is not available, this field will read as zero.

Access to this field is RO.

SLOTS, bits [7:0]

Operation width. The largest value by which the STALL_SLOT event might increment in a single cycle. If the STALL_SLOT event is not implemented, this field might read as zero.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Accessing PMMIR_EL1

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, PMMIR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1110	0b110

```
if !(IsFeatureImplemented(FEAT_PMUv3p4) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.PMMIR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMMIR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMMIR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMMIR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.20 PMOVSCLR_EL0, Performance Monitors Overflow Flag Status Clear Register

The PMOVSCLR_EL0 characteristics are:

Purpose

Allows software to clear the unsigned overflow flags for the following counters to 0:

- The cycle counter PMCCNTR_EL0.
- The event counters PMEVCNTR<n>_EL0.
- When FEAT_PMUv3_ICNTR is implemented, the instruction counter PMICNTR_EL0.

Reading from this register shows the current unsigned overflow flag values.

Configuration

AArch64 System register PMOVSCLR_EL0 bits [63:0] are architecturally mapped to AArch64 System register PMOVSSET_EL0[63:0].

AArch64 System register PMOVSCLR_EL0 bits [63:0] are architecturally mapped to External register PMOVS[63:0].

AArch64 System register PMOVSCLR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMOVSR[31:0].

AArch64 System register PMOVSCLR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMOVSSET[31:0].

AArch64 System register PMOVSCLR_EL0 bits [31:0] are architecturally mapped to External register PMOVSCLR_EL0[31:0].

AArch64 System register PMOVSCLR_EL0 bits [31:0] are architecturally mapped to External register PMOVSSET_EL0[31:0].

When FEAT_PMUv3p9 is implemented or FEAT_PMUv3_EXT64 is implemented, AArch64 System register PMOVSCLR_EL0 bits [63:32] are architecturally mapped to External register PMOVSCLR_EL0[63:32].

When FEAT_PMUv3p9 is implemented or FEAT_PMUv3_EXT64 is implemented, AArch64 System register PMOVSCLR_EL0 bits [63:32] are architecturally mapped to External register PMOVSSET_EL0[63:32].

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMOVSCLR_EL0 are UNDEFINED.

Attributes

PMOVSCLR_EL0 is a 64-bit register.

Field descriptions

Bits [63:33]

Reserved, RES0.

F0, bit [32]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_PMUv3_ICNTR is implemented:

Unsigned overflow flag for PMICNTR_EL0 clear. On writes, allows software to clear the unsigned overflow flag for PMICNTR_EL0 to 0. On reads, returns the unsigned overflow flag for PMICNTR_EL0 overflow status.

F0	Meaning
`0b0`	PMICNTR_EL0 has not overflowed.
`0b1`	PMICNTR_EL0 has overflowed.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- Access to this field is RAZ/WI if all of the following are true:
  - EL3 is implemented
  - PSTATE.EL != EL3
  - MDCR_EL3.EnPM2 == ‘0’
- Access to this field is RAZ/WI if all of the following are true:
  - PSTATE.EL == EL0
  - Any of the following are true:
    - PMUSERENR_EL0.UEN == ‘0’
    - PMUACR_EL1.F0 == ‘0’
- Access to this field is RAZ/WI if all of the following are true:
  - FEAT_FGT2 is implemented
  - EL2Enabled()
  - PSTATE.EL IN {EL1, EL0}
  - [HCR_EL2.E2H, HCR_EL2.TGE] != ‘11’
  - Any of the following are true:
    - All of the following are true:
      - EL3 is implemented
      - SCR_EL3.FGTEn2 == ‘0’
    - All of the following are true:
      - HDFGRTR2_EL2.nPMICFILTR_EL0 == ‘0’
      - HDFGWTR2_EL2.nPMICFILTR_EL0 == ‘0’
- Access to this field is RAZ/WI if all of the following are true:
  - FEAT_FGT2 is implemented
  - EL2Enabled()
  - PSTATE.EL == EL0
  - [HCR_EL2.E2H, HCR_EL2.TGE] != ‘11’
  - Any of the following are true:
    - All of the following are true:
      - EL3 is implemented
      - SCR_EL3.FGTEn2 == ‘0’
    - HDFGRTR2_EL2.nPMICFILTR_EL0 == ‘0’
  - PMUSERENR_EL0.IR == ‘1’
- Access to this field is WO/RAZ if all of the following are true:
  - FEAT_FGT2 is implemented
  - EL2Enabled()
  - PSTATE.EL IN {EL1, EL0}
  - [HCR_EL2.E2H, HCR_EL2.TGE] != ‘11’
  - Any of the following are true:
    - All of the following are true:
      - EL3 is implemented

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

        - SCR_EL3.FGTEn2 == ‘0’
      - HDFGRTR2_EL2.nPMICFILTR_EL0 == ‘0’
    - Access to this field is RO if all of the following are true:
      - FEAT_FGT2 is implemented
      - EL2Enabled()
      - PSTATE.EL IN {EL1, EL0}
      - [HCR_EL2.E2H, HCR_EL2.TGE] != ‘11’
      - Any of the following are true:
        - All of the following are true:
          - EL3 is implemented
          - SCR_EL3.FGTEn2 == ‘0’
        - HDFGWTR2_EL2.nPMICFILTR_EL0 == ‘0’
    - Access to this field is RO if all of the following are true:
      - PSTATE.EL == EL0
      - PMUSERENR_EL0.IR == ‘1’
    - Otherwise, access to this field is W1C.

Otherwise:

Reserved, RES0.

C, bit [31]

Unsigned overflow flag for PMCCNTR_EL0 clear. On writes, allows software to clear the unsigned overflow flag for PMCCNTR_EL0 to 0. On reads, returns the unsigned overflow flag for PMCCNTR_EL0 overflow status.

C	Meaning
0b0	PMCCNTR_EL0 has not overflowed.
0b1	PMCCNTR_EL0 has overflowed.

PMCR_EL0.LC controls whether an overflow is detected from unsigned overflow of PMCCNTR_EL0[31:0] or unsigned overflow of PMCCNTR_EL0[63:0].

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is RAZ/WI if all of the following are true:
  - FEAT_PMUv3p9 is implemented
  - PSTATE.EL == EL0
  - PMUSERENR_EL0.UEN == ‘1’
  - PMUACR_EL1.C == ‘0’
- Access to this field is RO if all of the following are true:
  - FEAT_PMUv3p9 is implemented
  - PSTATE.EL == EL0
  - [PMUSERENR_EL0.UEN, PMUSERENR_EL0.CR] == ‘11’
- Otherwise, access to this field is W1C.

P<m> , bits [m], for m = 30 to 0

Unsigned overflow flag for PMEVCNTR<m>_EL0 clear. On writes, allows software to clear the unsigned overflow flag for PMEVCNTR<m>_EL0 to 0. On reads, returns the unsigned overflow flag for PMEVCNTR<m>_EL0 overflow status.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

P<m>	Meaning
0b0	PMEVCNTR<m>_EL0 has not overflowed.
0b1	PMEVCNTR<m>_EL0 has overflowed.

If FEAT_PMUv3p5 is implemented, MDCR_EL2.HLP and PMCR_EL0.LP control whether an overflow is detected from unsigned overflow of PMEVCNTR<m>_EL0[31:0] or unsigned overflow of PMEVCNTR<m>_EL0[63:0].

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= GetNumEventCountersAccessible(), access to this field is RAZ/WI.
- Access to this field is RAZ/WI if all of the following are true:
  - FEAT_PMUv3p9 is implemented
  - PSTATE.EL == EL0
  - PMUSERENR_EL0.UEN == '1'
  - PMUACR_EL1.P<m> == '0'
- Access to this field is RO if all of the following are true:
  - FEAT_PMUv3p9 is implemented
  - PSTATE.EL == EL0
  - [PMUSERENR_EL0.UEN, PMUSERENR_EL0.ER] == '11'
- Otherwise, access to this field is W1C.

Accessing PMOVSCLR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMOVSCLR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1100	0b011

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')
         ↪then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMOVS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMOVSCLR_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.PMOVS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMOVSCLR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMOVSCLR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMOVSCLR_EL0;
```

```
MSR PMOVSCLR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1100	0b011

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')
         ↪then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMOVS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMOVSCLR_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.PMOVS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMOVSCLR_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMOVSCLR_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMOVSCLR_EL0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.21 PMOVSSET_EL0, Performance Monitors Overflow Flag Status Set Register

The PMOVSSET_EL0 characteristics are:

Purpose

Allows software to set the unsigned overflow flags for the following counters to 1:

- The cycle counter PMCCNTR_EL0.
- The event counters PMEVCNTR<n>_EL0.
- When FEAT_PMUv3_ICNTR is implemented, the instruction counter PMICNTR_EL0.

Reading from this register shows the current unsigned overflow flag values.

Configuration

AArch64 System register PMOVSSET_EL0 bits [63:0] are architecturally mapped to AArch64 System register PMOVSCLR_EL0[63:0].

AArch64 System register PMOVSSET_EL0 bits [63:0] are architecturally mapped to External register PMOVS[63:0].

AArch64 System register PMOVSSET_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMOVSSET[31:0].

AArch64 System register PMOVSSET_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMOVSR[31:0].

AArch64 System register PMOVSSET_EL0 bits [31:0] are architecturally mapped to External register PMOVSCLR_EL0[31:0].

AArch64 System register PMOVSSET_EL0 bits [31:0] are architecturally mapped to External register PMOVSSET_EL0[31:0].

When FEAT_PMUv3p9 is implemented or FEAT_PMUv3_EXT64 is implemented, AArch64 System register PMOVSSET_EL0 bits [63:32] are architecturally mapped to External register PMOVSCLR_EL0[63:32].

When FEAT_PMUv3p9 is implemented or FEAT_PMUv3_EXT64 is implemented, AArch64 System register PMOVSSET_EL0 bits [63:32] are architecturally mapped to External register PMOVSSET_EL0[63:32].

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMOVSSET_EL0 are UNDEFINED.

Attributes

PMOVSSET_EL0 is a 64-bit register.

Field descriptions

Bits [63:33]

Reserved, RES0.

F0, bit [32]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_PMUv3_ICNTR is implemented:

Unsigned overflow flag for PMICNTR_EL0 set. On writes, allows software to set the unsigned overflow flag for PMICNTR_EL0 to 1. On reads, returns the unsigned overflow flag for PMICNTR_EL0 overflow status.

F0	Meaning
`0b0`	PMICNTR_EL0 has not overflowed.
`0b1`	PMICNTR_EL0 has overflowed.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- Access to this field is RAZ/WI if all of the following are true:
  - EL3 is implemented
  - PSTATE.EL != EL3
  - MDCR_EL3.EnPM2 == ‘0’
- Access to this field is RAZ/WI if all of the following are true:
  - PSTATE.EL == EL0
  - Any of the following are true:
    - PMUSERENR_EL0.UEN == ‘0’
    - PMUACR_EL1.F0 == ‘0’
- Access to this field is RAZ/WI if all of the following are true:
  - FEAT_FGT2 is implemented
  - EL2Enabled()
  - PSTATE.EL IN {EL1, EL0}
  - [HCR_EL2.E2H, HCR_EL2.TGE] != ‘11’
  - Any of the following are true:
    - All of the following are true:
      - EL3 is implemented
      - SCR_EL3.FGTEn2 == ‘0’
    - All of the following are true:
      - HDFGRTR2_EL2.nPMICFILTR_EL0 == ‘0’
      - HDFGWTR2_EL2.nPMICFILTR_EL0 == ‘0’
- Access to this field is RAZ/WI if all of the following are true:
  - FEAT_FGT2 is implemented
  - EL2Enabled()
  - PSTATE.EL == EL0
  - [HCR_EL2.E2H, HCR_EL2.TGE] != ‘11’
  - Any of the following are true:
    - All of the following are true:
      - EL3 is implemented
      - SCR_EL3.FGTEn2 == ‘0’
    - HDFGRTR2_EL2.nPMICFILTR_EL0 == ‘0’
  - PMUSERENR_EL0.IR == ‘1’
- Access to this field is WO/RAZ if all of the following are true:
  - FEAT_FGT2 is implemented
  - EL2Enabled()
  - PSTATE.EL IN {EL1, EL0}
  - [HCR_EL2.E2H, HCR_EL2.TGE] != ‘11’
  - Any of the following are true:
    - All of the following are true:
      - EL3 is implemented
      - SCR_EL3.FGTEn2 == ‘0’

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

— HDFGRTR2_EL2.nPMICFILTR_EL0 == ‘0’
- Access to this field is RO if all of the following are true:
  - FEAT_FGT2 is implemented
  - EL2Enabled()
  - PSTATE.EL IN {EL1, EL0}
  - [HCR_EL2.E2H, HCR_EL2.TGE] != ‘11’
  - Any of the following are true:
    — All of the following are true:
      - EL3 is implemented
      - SCR_EL3.FGTEn2 == ‘0’
    — HDFGWTR2_EL2.nPMICFILTR_EL0 == ‘0’
- Access to this field is RO if all of the following are true:
  - PSTATE.EL == EL0
  - PMUSERENR_EL0.IR == ‘1’
- Otherwise, access to this field is W1S.

Otherwise:

Reserved, RES0.

C, bit [31]

Unsigned overflow flag for PMCCNTR_EL0 set. On writes, allows software to set the unsigned overflow flag for PMCCNTR_EL0 to 1. On reads, returns the unsigned overflow flag for PMCCNTR_EL0 overflow status.

C	Meaning
0b0	PMCCNTR_EL0 has not overflowed.
0b1	PMCCNTR_EL0 has overflowed.

PMCR_EL0.LC controls whether an overflow is detected from unsigned overflow of PMCCNTR_EL0[31:0] or unsigned overflow of PMCCNTR_EL0[63:0].

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is RAZ/WI if all of the following are true:
  - FEAT_PMUv3p9 is implemented
  - PSTATE.EL == EL0
  - PMUSERENR_EL0.UEN == ‘1’
  - PMUACR_EL1.C == ‘0’
- Access to this field is RO if all of the following are true:
  - FEAT_PMUv3p9 is implemented
  - PSTATE.EL == EL0
  - [PMUSERENR_EL0.UEN, PMUSERENR_EL0.CR] == ‘11’
- Otherwise, access to this field is W1S.

P<m> , bits [m], for m = 30 to 0

Unsigned overflow flag for PMEVCNTR<m>_EL0 set. On writes, allows software to set the unsigned overflow flag for PMEVCNTR<m>_EL0 to 1. On reads, returns the unsigned overflow flag for PMEVCNTR<m>_EL0 overflow status.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

P<m>	Meaning
0b0	PMEVCNTR<m>_EL0 has not overflowed.
0b1	PMEVCNTR<m>_EL0 has overflowed.

If FEAT_PMUv3p5 is implemented, MDCR_EL2.HLP and PMCR_EL0.LP control whether an overflow is detected from unsigned overflow of PMEVCNTR<m>_EL0[31:0] or unsigned overflow of PMEVCNTR<m>_EL0[63:0].

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_PMUv3_EXTPMN is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_PMUv3_EXTPMN is not implemented, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= GetNumEventCountersAccessible(), access to this field is RAZ/WI.
- Access to this field is RAZ/WI if all of the following are true:
  - FEAT_PMUv3p9 is implemented
  - PSTATE.EL == EL0
  - PMUSERENR_EL0.UEN == '1'
  - PMUACR_EL1.P<m> == '0'
- Access to this field is RO if all of the following are true:
  - FEAT_PMUv3p9 is implemented
  - PSTATE.EL == EL0
  - [PMUSERENR_EL0.UEN, PMUSERENR_EL0.ER] == '11'
- Otherwise, access to this field is W1S.

Accessing PMOVSSET_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMOVSSET_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1110	0b011

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')
         ↪then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMOVS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMOVSSET_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.PMOVS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMOVSSET_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMOVSSET_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMOVSSET_EL0;
```

```
MSR PMOVSSET_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1110	0b011

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')
         ↪then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMOVS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMOVSSET_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.PMOVS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMOVSSET_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMOVSSET_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMOVSSET_EL0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.22 PMSELR_EL0, Performance Monitors Event Counter Selection Register

The PMSELR_EL0 characteristics are:

Purpose

Selects the current event counter PMEVCNTR<n>_EL0 or the cycle counter PMCCNTR_EL0.

Used in conjunction with PMXEVTYPER_EL0 to determine the event that increments a selected counter, and the modes and states in which the selected counter increments.

Used in conjunction with PMXEVCNTR_EL0 to determine the value of a selected counter.

Configuration

AArch64 System register PMSELR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMSELR[31:0].

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMSELR_EL0 are UNDEFINED.

Attributes

PMSELR_EL0 is a 64-bit register.

Field descriptions

Bits [63:5]

Reserved, RES0.

SEL, bits [4:0]

Event counter select. Selects the counter accessed by subsequent accesses to PMXEVTYPER_EL0 and PMXEVCNTR_EL0.

SEL	Meaning
0b00000..0b11110	Select event counter PMEVCNTR<n>_EL0, where n is the value of this field: • MRS and MSR of PMXEVTYPER_EL0 access PMEVTYPER<n>_EL0. • MRS and MSR of PMXEVCNTR_EL0 access PMEVCNTR<n>_EL0.
0b11111	Select the cycle counter, PMCCNTR_EL0: • MRS and MSR of PMXEVTYPER_EL0 access PMCCFILTR_EL0. • MRS and MSR of PMXEVCNTR_EL0 are CONSTRAINED UNPREDICTABLE. For more information, see PMXEVCNTR_EL0.

If FEAT_FGT is not implemented and this field is set to a value greater than or equal to the number of implemented counters, but not equal to 31, then direct reads of this field return an UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

For more information about the results of accesses to the event counters, including when PMSELR_EL0.SEL is set to the index of an unimplemented or inaccessible event counter, see PMXEVTYPER_EL0 and PMXEVCNTR_EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMSELR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMSELR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1100	0b101

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif (IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<UEN,ER,EN> == '000') ||
         ↪(!IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<ER,EN> == '00') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMSELR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMSELR_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.PMSELR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMSELR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMSELR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMSELR_EL0;
```

MSR PMSELR_EL0, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1100	0b101

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif (IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<UEN,ER,EN> == '000') ||
         ↪(!IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<ER,EN> == '00') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMSELR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSELR_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.PMSELR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSELR_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSELR_EL0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL3 then
    PMSELR_EL0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.23 PMSSCR_EL1, Performance Monitors Snapshot Status and Capture Register

The PMSSCR_EL1 characteristics are:

Purpose

Holds status information about the captured counters and provides a mechanism for software to initiate a sample.

Configuration

AArch64 System register PMSSCR_EL1 bits [63:0] are architecturally mapped to External register PMSSCR_EL1[63:0].

This register is present only when FEAT_PMUv3_SS is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMSSCR_EL1 are UNDEFINED.

Attributes

PMSSCR_EL1 is a 64-bit register.

Field descriptions

Bits [63:33]

Reserved, RES0.

NC, bit [32]

No Capture. Indicates whether the PMU counters have been captured.

NC	Meaning
0b0	PMU counters captured.
0b1	PMU counters not captured.

The reset behavior of this field is:

- On a Warm reset, this field resets to '1'.

Bits [31:1]

Reserved, RES0.

SS, bit [0]

Snapshot Capture and Status.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SS	Meaning
0b0	On a read, the Capture event has completed.
0b1	On a read, the Capture event has not completed. On a write, request a Capture event.

A write of 0 to this field is ignored.

It is CONSTRAINED UNPREDICTABLE whether a Capture event has completed if this field is modified when the Capture event is ongoing.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- When PMU capture events are disabled, access to this field is RO.
- Otherwise, access to this field is RW.

Accessing PMSSCR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMSSCR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1101	0b011

```
if !(IsFeatureImplemented(FEAT_PMUv3_SS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMSS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGRTR2_EL2.nPMSSCR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPMSS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMSSCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMSS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPMSS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMSSCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMSSCR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MSR PMSSCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1101	0b011

```
if !(IsFeatureImplemented(FEAT_PMUv3_SS) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMSS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGWTR2_EL2.nPMSSCR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPMSS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSSCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMSS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPMSS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSSCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMSSCR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.24 PMSWINC_EL0, Performance Monitors Software Increment Register

The PMSWINC_EL0 characteristics are:

Purpose

Increments a counter that is configured to count the Software increment event, event `0x00`. For more information, see 'SW_INCR'.

Configuration

AArch64 System register PMSWINC_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMSWINC[31:0].

When FEAT_PMUv3_EXT32 is implemented and FEAT_PMUv3p9 is not implemented, AArch64 System register PMSWINC_EL0 bits [31:0] are architecturally mapped to External register PMSWINC_EL0[31:0].

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMSWINC_EL0 are UNDEFINED.

Attributes

PMSWINC_EL0 is a 64-bit register.

Field descriptions

Bits [63:31]

Reserved, RES0.

P<m> , bits [m], for m = 30 to 0

Software increment.

P<m>	Meaning
`0b0`	Write is ignored.
`0b1`	Increment PMEVCNTR<m>_EL0, if PMEVCNTR<m>_EL0 is configured to count software increment events.

Accessing this field has the following behavior:

- When m >= GetNumEventCountersAccessible(), access to this field is RAZ/WI.
- Access to this field is RAZ/WI if all of the following are true:
  - FEAT_PMUv3p9 is implemented
  - PSTATE.EL == EL0
  - [PMUSERENR_EL0.UEN, PMUSERENR_EL0.SW] == '10'
  - PMUACR_EL1.P<m> == '0'
- Otherwise, access to this field is WO/RAZ.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing PMSWINC_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MSR PMSWINC_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1100	0b100

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif (IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<UEN,SW,EN> == '000') ||
         ↪(!IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<SW,EN> == '00') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMSWINC_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSWINC_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.PMSWINC_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSWINC_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSWINC_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMSWINC_EL0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.25 PMUACR_EL1, Performance Monitors User Access Control Register

The PMUACR_EL1 characteristics are:

Purpose

Enables or disables EL0 access to specfic Performance Monitors.

Configuration

This register is present only when FEAT_PMUv3p9 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMUACR_EL1 are UNDEFINED.

Attributes

PMUACR_EL1 is a 64-bit register.

Field descriptions

Bits [63:33]

Reserved, RES0.

F0, bit [32]

When FEAT_PMUv3_ICNTR is implemented:

EL0 accesses to PMICNTR_EL0 enable. With PMUSERENR_EL0.UEN and PMUSERENR_EL0.IR, controls EL0 accesses to PMICNTR_EL0.

F0	Meaning
0b0	If the Effective value of PMUSERENR_EL0.UEN is 1, then EL0 accesses to PMICNTR_EL0 and associated controls are RAZ/WI, and permitted EL0 writes to PMZR_EL0.F0 are ignored.
0b1	If the Effective value of PMUSERENR_EL0.UEN is 1, then EL0 accesses to PMICNTR_EL0 and associated controls are either read-only or read/write, depending on the value of PMUSERENR_EL0.IR.

The controls associated with PMICNTR_EL0 that are accessible at EL0 are PMCNTENSET_EL0.F0, PMCNTENCLR_EL0.F0, PMOVSSET_EL0.F0, and PMOVSCLR_EL0.F0.

This field is ignored by the PE when any of the following are true:

- EL1 is using AArch32.
- PMUSERENR_EL0.UEN is 0.
- The access generates an exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

C, bit [31]

EL0 accesses to PMCCNTR_EL0 enable. With PMUSERENR_EL0.EN, PMUSERENR_EL0.UEN, and PMUSERENR_EL0.CR, controls EL0 accesses to PMCCNTR_EL0.

C	Meaning
0b0	If the Effective value of PMUSERENR_EL0.UEN is 1, then EL0 accesses to PMCCNTR_EL0 and associated controls are RAZ/WI, and permitted EL0 writes to PMZR_EL0.C are ignored.
0b1	If the Effective value of PMUSERENR_EL0.UEN is 1, then EL0 accesses to PMCCNTR_EL0 and associated controls are either read-only or read/write, depending on the value of PMUSERENR_EL0.CR.

The controls associated with PMCCNTR_EL0 that are accessible at EL0 are PMCNTENSET_EL0.C, PMCNTENCLR_EL0.C, PMOVSSET_EL0.C, and PMOVSCLR_EL0.C.

This field is ignored by the PE when any of the following are true:

- EL1 is using AArch32.
- PMUSERENR_EL0.UEN is 0.
- The access generates an exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P<m> , bits [m], for m = 30 to 0

EL0 accesses to PMEVCNTR<m>_EL0 enable. With PMUSERENR_EL0.EN, PMUSERENR_EL0.UEN, and PMUSERENR_EL0.ER, controls EL0 accesses to PMEVCNTR<m>_EL0.

P<m>	Meaning
0b0	If the Effective value of PMUSERENR_EL0.UEN is 1, then EL0 accesses to PMEVCNTR<m>_EL0 and associated controls are RAZ/WI, and permitted EL0 writes to PMZR_EL0.P<m> are ignored.
0b1	If the Effective value of PMUSERENR_EL0.UEN is 1, then EL0 accesses to PMEVCNTR<m>_EL0 and associated controls are either read-only or read/write, depending on the value of PMUSERENR_EL0.ER.

The controls associated with PMEVCNTR<m>_EL0 that are accessible at EL0 are PMCNTENSET_EL0.P<m>, PMCNTENCLR_EL0.P<m>, PMOVSSET_EL0.P<m>, and PMOVSCLR_EL0.P<m>.

This field is ignored by the PE when any of the following are true:

- EL1 is using AArch32.
- PMUSERENR_EL0.UEN is 0.
- The access generates an exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m >= GetNumEventCountersAccessible(), access to this field is RAZ/WI.
- Otherwise, access to this field is RW.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing PMUACR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMUACR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1110	0b100

```
if !(IsFeatureImplemented(FEAT_PMUv3p9) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGRTR2_EL2.nPMUACR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMUACR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMUACR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMUACR_EL1;
```

```
MSR PMUACR_EL1, <Xt>
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1110	0b100

```
if !(IsFeatureImplemented(FEAT_PMUv3p9) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGWTR2_EL2.nPMUACR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMUACR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMUACR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMUACR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.26 PMUSERENR_EL0, Performance Monitors User Enable Register

The PMUSERENR_EL0 characteristics are:

Purpose

Enables or disables EL0 access to the Performance Monitors.

Configuration

AArch64 System register PMUSERENR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMUSERENR[31:0].

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMUSERENR_EL0 are UNDEFINED.

Attributes

PMUSERENR_EL0 is a 64-bit register.

Field descriptions

Bits [63:7]

Reserved, RES0.

TID, bit [6]

When FEAT_PMUv3p9 is implemented:

Trap ID registers. Traps EL0 read access to common event identification registers.

TID	Meaning
0b0	Accesses to PMCEID<n>_EL0 and PMCEID<n> are not trapped by this mechanism.
0b1	EL0 read accesses to PMCEID<n>_EL0 and PMCEID<n> are trapped.

In AArch64 state, the register accesses affected by this control are:

- MRS reads of PMCEID0_EL0 and PMCEID1_EL0.

In AArch32 state, the register accesses affected by this control are:

- MRC reads of PMCEID0, PMCEID1, PMCEID2, and PMCEID3.

When trapped, reads generate an exception to EL1, or to EL2 when EL2 is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, and:

- AArch64 MRS reads are reported using EC syndrome value 0x18.
- AArch32 MRC reads are reported using EC syndrome value 0x03.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IR, bit [5]

When FEAT_PMUv3_ICNTR is implemented:

Instruction counter Read-only. When PMUSERENR_EL0.UEN is 1, controls whether EL0 writes to instruction counter are ignored.

IR	Meaning
0b0	EL0 accesses are not affected by this mechanism.
0b1	If the Effective value of PMUSERENR_EL0.UEN is 1, then all of the following apply at EL0: • Writes to PMICNTR_EL0 are ignored. • The controls associated with instruction counter are RAZ/WI. • Writes to PMZR_EL0.F0 are ignored.

The controls associated with PMICNTR_EL0 that are accessible at EL0 are PMCNTENSET_EL0.F0, PMCNTENCLR_EL0.F0, PMOVSSET_EL0.F0, and PMOVSCLR_EL0.F0.

Ignored writes are not trapped and do not generate an exception.

This field is ignored by the PE when any of the following are true:

- PMUSERENR_EL0.UEN is 0.
- The access generates an exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UEN, bit [4]

When FEAT_PMUv3p9 is implemented:

User Enable, with access controlled by PMUACR_EL1. Enables EL0 read/write access to PMU registers, other than PMCR_EL0.

UEN	Meaning
0b0	If FEAT_PMUv3_ICNTR is implemented, then EL0 accesses to PMICFILTR_EL0 and PMICNTR_EL0 are trapped. EL0 accesses to the other specified PMU registers, PMCR_EL0, and PMCR are trapped, unless enabled by PMUSERENR_EL0.{ER,CR,SW,EN}.
0b1	EL0 accesses to the specified PMU registers are enabled, unless trapped by another control. The behavior of permitted accesses is controlled by PMUSERENR_EL0.{ IR, ER,CR,SW} and PMUACR_EL1. EL0 accesses to PMCR_EL0 and PMCR are trapped.

In AArch64 state, the register accesses affected by this control are:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- MRS or MSR accesses to the following registers:
  - PMCCFILTR_EL0, PMCCNTR_EL0, PMCNTENCLR_EL0, PMCNTENSET_EL0, PMEVCNTR<n>_EL0, PMEVTYPER<n>_EL0, PMOVSCLR_EL0, PMOVSSET_EL0, PMSELR_EL0, PMXEVCNTR_EL0, and PMXEVTYPER_EL0.
  - If FEAT_PMUv3_ICNTR is implemented, PMICFILTR_EL0 and PMICNTR_EL0.
- MRS reads of PMCEID0_EL0 and PMCEID1_EL0.
- MSR writes to PMSWINC_EL0 and PMZR_EL0.

In AArch32 state, the register accesses affected by this control are:

- MRC or MCR accesses to PMCCFILTR, PMCCNTR, PMCNTENCLR, PMCNTENSET, PMEVCNTR<n>, PMEVTYPER<n>, PMOVSR, PMOVSSET, PMSELR, PMXEVCNTR, and PMXEVTYPER.
- MRC reads of PMCEID0, PMCEID1, PMCEID2, and PMCEID3.
- MCR writes to PMSWINC.
- MRRC or MCRR accesses to PMCCNTR.

When trapped, reads and writes generate an exception to EL1, or to EL2 when EL2 is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, and:

- AArch64 MRS and MSR accesses are reported using EC syndrome value 0x18.
- AArch32 MRC and MCR accesses are reported using EC syndrome value 0x03.
- AArch32 MRRC and MCRR accesses are reported using EC syndrome value 0x04.

This field is ignored by the PE and treated as zero when EL1 is using AArch32.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ER, bit [3]

When FEAT_PMUv3p9 is implemented:

Event counters Read enable or Read-only.

When PMUSERENR_EL0.{UEN,EN} is {0,0}, PMUSERENR_EL0.ER enables EL0 reads of the event counters and EL0 reads and writes of the select register.

When PMUSERENR_EL0.UEN is 1, EL0 reads of the event counters and EL0 writes to PMZR_EL0 are enabled by PMUSERENR_EL0.UEN, unless trapped by another control, and PMUSERENR_EL0.ER controls the behavior of EL0 writes to the event counters and PMZR_EL0.

ER	Meaning
0b0	When PMUSERENR_EL0.UEN == 0, EL0 reads of the event counters and EL0 reads and writes of the select register are disabled, unless enabled by PMUSERENR_EL0.EN. When PMUSERENR_EL0.UEN == 1, permitted EL0 writes are not affected by this mechanism.
0b1	When PMUSERENR_EL0.UEN == 0, EL0 reads of the event counters and EL0 reads and writes of the select register are enabled, unless trapped by another control. When PMUSERENR_EL0.UEN == 1, permitted EL0 writes to the event counters and PMZR_EL0.P[30:0] are ignored.

In AArch64 state, the register accesses affected by this control are:

- When PMUSERENR_EL0.{UEN,EN} is {0,0}:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- MRS reads of PMEVCNTR<n>_EL0 and PMXEVCNTR_EL0.
- MRS and MSR accesses to PMSELR_EL0.

- When PMUSERENR_EL0.UEN is 1, MSR writes to PMZR_EL0.P<n>, PMEVCNTR<n>_EL0, and PMXEVCNTR_EL0.

In AArch32 state, the register accesses affected by this control are:

- When PMUSERENR_EL0.{UEN,EN} is {0,0}:
  - MRC reads of PMEVCNTR<n> and PMXEVCNTR.
  - MRC and MCR accesses to PMSELR.
- When PMUSERENR_EL0.UEN is 1, MCR writes to PMEVCNTR<n> and PMXEVCNTR.

When disabled, reads and writes generate an exception to EL1, or to EL2 when EL2 is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, and:

- AArch64 MRS and MSR accesses are reported using EC syndrome value 0x18.
- AArch32 MRC and MCR accesses are reported using EC syndrome value 0x03.

Ignored writes are not trapped and do not generate an exception.

This field is ignored by the PE when PMUSERENR_EL0.{UEN,EN} == {0,1}.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Event counters Read enable.

When PMUSERENR_EL0.EN is 0, PMUSERENR_EL0.ER enables EL0 reads of the event counters and EL0 reads and writes of the select register.

ER	Meaning
0b0	EL0 reads of the event counters and EL0 reads and writes of the select register are disabled, unless enabled by PMUSERENR_EL0.EN.
0b1	EL0 reads of the event counters and EL0 reads and writes of the select register are enabled, unless trapped by another control.

In AArch64 state, the register accesses affected by this control are:

- MRS reads of PMEVCNTR<n>_EL0 and PMXEVCNTR_EL0.
- MRS and MSR accesses to PMSELR_EL0.

In AArch32 state, the register accesses affected by this control are:

- MRC reads of PMEVCNTR<n> and PMXEVCNTR.
- MRC and MCR accesses to PMSELR.

When disabled, reads and writes generate an exception to EL1, or to EL2 when EL2 is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, and:

- AArch64 MRS and MSR accesses are reported using EC syndrome value 0x18.
- AArch32 MRC and MCR accesses are reported using EC syndrome value 0x03.

This field is ignored by the PE when PMUSERENR_EL0.EN == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CR, bit [2]

When FEAT_PMUv3p9 is implemented:

Cycle counter Read enable or Read-only.

When PMUSERENR_EL0.{UEN,EN} is {0,0}, PMUSERENR_EL0.CR enables EL0 reads of the cycle counter.

When PMUSERENR_EL0.UEN is 1, EL0 reads of the cycle counter and EL0 writes to PMZR_EL0 are enabled by PMUSERENR_EL0.UEN, unless trapped by another control, and PMUSERENR_EL0.CR controls the behavior of EL0 writes to the cycle counter and PMZR_EL0.

CR	Meaning
`0b0`	When PMUSERENR_EL0.UEN == 0, EL0 reads of the cycle counter are disabled, unless enabled by PMUSERENR_EL0.EN. When PMUSERENR_EL0.UEN == 1, permitted EL0 writes are not affected by this mechanism.
`0b1`	When PMUSERENR_EL0.UEN == 0, EL0 reads of the cycle counter are enabled, unless trapped by another control. When PMUSERENR_EL0.UEN == 1, permitted EL0 writes to the cycle counter and PMZR_EL0.C are ignored.

In AArch64 state, the register accesses affected by this control are:

- When PMUSERENR_EL0.{UEN,EN} is {0,0}, `MRS` reads of PMCCNTR_EL0.
- When PMUSERENR_EL0.UEN is 1, `MSR` writes to PMZR_EL0.C and PMCCNTR_EL0.

In AArch32 state, the register accesses affected by this control are:

- When PMUSERENR_EL0.{UEN,EN} is {0,0}:
  - `MRC` reads of PMCCNTR.
  - `MRRC` reads of PMCCNTR.
- When PMUSERENR_EL0.UEN is 1:
  - `MCR` writes to PMCCNTR.
  - `MCRR` writes to PMCCNTR.

When disabled, reads generate an exception to EL1, or to EL2 when EL2 is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, and:

- AArch64 `MRS` reads are reported using EC syndrome value `0x18`.
- AArch32 `MRC` reads are reported using EC syndrome value `0x03`.
- AArch32 `MRRC` reads are reported using EC syndrome value `0x04`.

Ignored writes are not trapped and do not generate an exception.

This field is ignored by the PE when PMUSERENR_EL0.{UEN,EN} == {0,1}.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Cycle counter Read enable.

When PMUSERENR_EL0.EN is 0, PMUSERENR_EL0.CR enables EL0 reads of the cycle counter.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CR	Meaning
0b0	EL0 reads of the cycle counter are disabled, unless enabled by PMUSERENR_EL0.EN.
0b1	EL0 reads of the cycle counter are enabled, unless trapped by another control.

In AArch64 state, the register accesses affected by this control are:

- MRS reads of PMCCNTR_EL0.

In AArch32 state, the register accesses affected by this control are:

- MRC reads of PMCCNTR.
- MRRC reads of PMCCNTR.

When disabled, reads generate an exception to EL1, or to EL2 when EL2 is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, and:

- AArch64 MRS reads are reported using EC syndrome value 0x18.
- AArch32 MRC reads are reported using EC syndrome value 0x03.
- AArch32 MRRC reads are reported using EC syndrome value 0x04.

This field is ignored by the PE when PMUSERENR_EL0.EN == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SW, bit [1]

When FEAT_PMUv3p9 is implemented:

Software increment register Write enable.

When PMUSERENR_EL0.UEN is 0, PMUSERENR_EL0.SW enables EL0 writes to the Software increment register.

When PMUSERENR_EL0.UEN is 1, EL0 writes to the Software increment register are enabled by PMUSERENR_EL0.UEN, unless trapped by another control, and PMUSERENR_EL0.SW controls the behavior of EL0 writes to the Software increment register.

SW	Meaning
0b0	When PMUSERENR_EL0.UEN == 0, EL0 writes to the Software increment register are disabled, unless enabled by PMUSERENR_EL0.EN. When PMUSERENR_EL0.UEN == 1, permitted EL0 writes are not affected by this mechanism.
0b1	When PMUSERENR_EL0.UEN == 0, EL0 writes to the Software increment register are enabled, unless trapped by another control. When PMUSERENR_EL0.UEN == 1, permitted EL0 writes to the Software increment register ignore the value of PMUACR_EL1.

In AArch64 state, the register accesses affected by this control are:

- MSR writes to PMSWINC_EL0.

In AArch32 state, the register accesses affected by this control are:

- MCR writes to PMSWINC.

When disabled, writes generate an exception to EL1, or to EL2 when EL2 is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, and:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- AArch64 `MSR` writes are reported using EC syndrome value `0x18`.
- AArch32 `MCR` writes are reported using EC syndrome value `0x03`.

This field is ignored by the PE when PMUSERENR_EL0.{UEN,EN} == {0,1}.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Software increment register Write enable.

When PMUSERENR_EL0.EN is 0, PMUSERENR_EL0.SW enables EL0 writes to the Software increment register.

SW	Meaning
`0b0`	EL0 writes to the Software increment register are disabled, unless enabled by PMUSERENR_EL0.EN.
`0b1`	EL0 writes to the Software increment register are enabled, unless trapped by another control.

In AArch64 state, the register accesses affected by this control are:

- `MSR` writes to PMSWINC_EL0.

In AArch32 state, the register accesses affected by this control are:

- `MCR` writes to PMSWINC.

When disabled, writes generate an exception to EL1, or to EL2 when EL2 is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, and:

- AArch64 `MSR` writes are reported using EC syndrome value `0x18`.
- AArch32 `MCR` writes are reported using EC syndrome value `0x03`.

This field is ignored by the PE when PMUSERENR_EL0.EN == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EN, bit [0]

Enable.

Enables EL0 read/write access to PMU registers, other than the instruction counter.

EN	Meaning
`0b0`	EL0 accesses to the specified PMU System registers are trapped, unless enabled by PMUSERENR_EL0.{UEN,ER,CR,SW}.
`0b1`	EL0 accesses to the specified PMU System registers are enabled, unless trapped by another control.

In AArch64 state, the register accesses affected by this control are:

- `MRS` or `MSR` accesses to PMCCFILTR_EL0, PMCCNTR_EL0, PMCNTENCLR_EL0, PMCNTENSET_EL0, PMCR_EL0, PMEVCNTR<n>_EL0, PMEVTYPER<n>_EL0, PMOVSCLR_EL0, PMOVSSET_EL0, PMSELR_EL0, PMXEVCNTR_EL0, and PMXEVTYPER_EL0.
- `MRS` reads of PMCEID0_EL0 and PMCEID1_EL0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- MSR writes to the following registers:
  - PMSWINC_EL0.
  - If FEAT_PMUv3p9 is implemented, PMZR_EL0.

Note

When FEAT_PMUv3_ICNTR is implemented, this field does not affect MRS and MSR accesses to PMICNTR_EL0 and PMICFILTR_EL0.

In AArch32 state, the register accesses affected by this control are:

- MRC or MCR accesses to PMCCFILTR, PMCCNTR, PMCNTENCLR, PMCNTENSET, PMCR, PMEVCNTR<n>, PMEVTYPER<n>, PMOVSR, PMOVSSET, PMSELR, PMXEVCNTR, and PMXEVTYPER.
- MRC reads of the following registers:
  - PMCEID0 and PMCEID1.
  - If FEAT_PMUv3p1 is implemented, PMCEID2 and PMCEID3.
- MCR writes to PMSWINC.
- MRRC or MCRR accesses to PMCCNTR.

When trapped, reads and writes generate an exception to EL1, or to EL2 when EL2 is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, and:

- AArch64 MRS and MSR accesses are reported using EC syndrome value 0x18.
- AArch32 MRC and MCR accesses are reported using EC syndrome value 0x03.
- AArch32 MRRC and MCRR accesses are reported using EC syndrome value 0x04.

This field is ignored by the PE when FEAT_PMUv3p9 is implemented and PMUSERENR_EL0.UEN == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMUSERENR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMUSERENR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1110	0b000

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMUSERENR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMUSERENR_EL0;
elsif PSTATE.EL == EL1 then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.PMUSERENR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMUSERENR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMUSERENR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMUSERENR_EL0;
```

```
MSR PMUSERENR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1110	0b000

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.PMUSERENR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMUSERENR_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        PMUSERENR_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMUSERENR_EL0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.27 PMXEVCNTR_EL0, Performance Monitors Selected Event Count Register

The PMXEVCNTR_EL0 characteristics are:

Purpose

Reads or writes the value of the selected event counter, PMEVCNTR<n>_EL0. PMSELR_EL0.SEL determines which event counter is selected.

Configuration

AArch64 System register PMXEVCNTR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMXEVCNTR[31:0].

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMXEVCNTR_EL0 are UNDEFINED.

Attributes

PMXEVCNTR_EL0 is a 64-bit register.

Field descriptions

When FEAT_PMUv3p5 is implemented:

PMEVCNTR<n>, bits [63:0]

Value of the selected event counter, PMEVCNTR<n>_EL0, where n is the value stored in PMSELR_EL0.SEL.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Bits [63:32]

Reserved, RES0.

PMEVCNTR<n>, bits [31:0]

Value of the selected event counter, PMEVCNTR<n>_EL0, where n is the value stored in PMSELR_EL0.SEL.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing PMXEVCNTR_EL0

If FEAT_FGT is implemented and PMSELR_EL0.SEL is greater than or equal to the number of accessible event counters, then the behavior of permitted reads and writes of PMXEVCNTR_EL0 is as follows:

- If PMSELR_EL0.SEL is greater than or equal to the Effective value of PMCCR.EPMN, the access is UNDEFINED.
- Otherwise, the access is trapped to EL2.

If FEAT_FGT is not implemented and PMSELR_EL0.SEL is greater than or equal to the number of accessible event counters, then reads and writes of PMXEVCNTR_EL0 are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP
- Accesses to the register behave as if PMSELR_EL0.SEL has an UNKNOWN value less than the number of counters accessible at the current Exception level and Security state.
- If EL2 is implemented and enabled in the current Security state, and PMSELR_EL0.SEL is less than the number of implemented event counters, accesses from EL1 or permitted accesses from EL0 are trapped to EL2.

Permitted reads and writes of PMXEVCNTR_EL0 are RAZ/WI if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- PMUSERENR_EL0.UEN == 1.
- PMUACR_EL1.P<UInt(PMSELR_EL0.SEL)> == 0.

Permitted writes of PMXEVCNTR_EL0 are ignored if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- PMUSERENR_EL0.{UEN,ER} == {1,1}.

Note

In EL0, an access is permitted if it is enabled by PMUSERENR_EL0.{UEN,ER,EN}.
If EL2 is implemented and enabled in the current Security state, in EL1 and EL0, MDCR_EL2.HPMN identifies the number of accessible event counters.
Otherwise, the number of accessible event counters is determined by the Effective value of PMCCR.EPMN. For more information, see MDCR_EL2.HPMN and PMCCR.EPMN.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMXEVCNTR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1101	0b010

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif UInt(PMSELR_EL0.SEL) >= GetNumEventCountersSelfHosted() then
    if IsFeatureImplemented(FEAT_FGT) then
        UNDEFINED;
    else
        ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif (IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<UEN,ER,EN> == '000') ||
         ↪(!IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.<ER,EN> == '00') then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMEVCNTRn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && UInt(PMSELR_EL0.SEL) >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1' &&
         ↪PMUACR_EL1[UInt(PMSELR_EL0.SEL)] == '0' then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = PMEVCNTR_EL0[UInt(PMSELR_EL0.SEL)];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.PMEVCNTRn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && UInt(PMSELR_EL0.SEL) >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMEVCNTR_EL0[UInt(PMSELR_EL0.SEL)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMEVCNTR_EL0[UInt(PMSELR_EL0.SEL)];
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMEVCNTR_EL0[UInt(PMSELR_EL0.SEL)];
```

```
MSR PMXEVCNTR_EL0, <Xt>
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1101	0b010

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif UInt(PMSELR_EL0.SEL) >= GetNumEventCountersSelfHosted() then
    if IsFeatureImplemented(FEAT_FGT) then
        UNDEFINED;
    else
        ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')
         ↪then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMEVCNTRn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && UInt(PMSELR_EL0.SEL) >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1' &&
         ↪(PMUACR_EL1[UInt(PMSELR_EL0.SEL)] == '0' || PMUSERENR_EL0.ER == '1') then
        return;
    else
        PMEVCNTR_EL0[UInt(PMSELR_EL0.SEL)] = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.PMEVCNTRn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && UInt(PMSELR_EL0.SEL) >= GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMEVCNTR_EL0[UInt(PMSELR_EL0.SEL)] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMEVCNTR_EL0[UInt(PMSELR_EL0.SEL)] = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMEVCNTR_EL0[UInt(PMSELR_EL0.SEL)] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.28 PMXEVTYPER_EL0, Performance Monitors Selected Event Type Register

The PMXEVTYPER_EL0 characteristics are:

Purpose

When PMSELR_EL0.SEL selects an event counter, this accesses a PMEVTYPER<n>_EL0 register. When PMSELR_EL0.SEL selects the cycle counter, this accesses PMCCFILTR_EL0.

Configuration

AArch64 System register PMXEVTYPER_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMXEVTYPER[31:0].

This register is present only when FEAT_PMUv3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMXEVTYPER_EL0 are UNDEFINED.

Attributes

PMXEVTYPER_EL0 is a 64-bit register.

Field descriptions

EVTYPERn, bits [63:0]

When PMSELR_EL0.SEL == 31, this register accesses PMCCFILTR_EL0.

Otherwise, this register accesses PMEVTYPER<n>_EL0 where n is the value in PMSELR_EL0.SEL.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMXEVTYPER_EL0

If FEAT_FGT is implemented, and PMSELR_EL0.SEL is not 31 and is greater than or equal to the number of accessible event counters, then the behavior of permitted reads and writes of PMXEVTYPER_EL0 is as follows:

- If PMSELR_EL0.SEL is greater than or equal to the Effective value of PMCCR.EPMN, the access is UNDEFINED.
- Otherwise, the access is trapped to EL2.

If FEAT_FGT is not implemented, and PMSELR_EL0.SEL is not 31 and is greater than or equal to the number of accessible event counters, then reads and writes of PMXEVTYPER_EL0 are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP.
- Accesses to the register behave as if PMSELR_EL0.SEL has an UNKNOWN value less than the number of event counters accessible at the current Exception level and Security state.
- Accesses to the register behave as if PMSELR_EL0.SEL is 31.
- If EL2 is implemented and enabled in the current Security state, PMSELR_EL0 is less than the number of implemented event counters, accesses from EL1 or permitted accesses from EL0 are trapped to EL2.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Permitted reads and writes of PMXEVTYPER_EL0 are RAZ/WI if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- PMUSERENR_EL0.UEN == 1.
- Any of the following are true:
  - PMSELR_EL0.SEL != 31 and PMUACR_EL1.P<UInt(PMSELR_EL0.SEL)> == 0.
  - PMSELR_EL0.SEL == 31 and PMUACR_EL1.C == 0.

Permitted writes of PMXEVTYPER_EL0 are ignored if all of the following are true:

- FEAT_PMUv3p9 is implemented.
- PSTATE.EL == EL0.
- PMUSERENR_EL0.UEN == 1.
- Any of the following are true:
  - PMSELR_EL0.SEL != 31 and PMUSERENR_EL0.ER == 1.
  - PMSELR_EL0.SEL == 31 and PMUSERENR_EL0.CR == 1.

Note

In EL0, an access is permitted if it is enabled by PMUSERENR_EL0.{UEN,EN}.
If EL2 is implemented and enabled in the current Security state, in EL1 and EL0, MDCR_EL2.HPMN identifies the number of accessible event counters.
Otherwise, the number of accessible event counters is determined by the Effective value of PMCCR.EPMN. For more information, see MDCR_EL2.HPMN and PMCCR.EPMN.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMXEVTYPER_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1101	0b001

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif UInt(PMSELR_EL0.SEL) != 31 && UInt(PMSELR_EL0.SEL) >= GetNumEventCountersSelfHosted() then
    if IsFeatureImplemented(FEAT_FGT) then
        UNDEFINED;
    else
        ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')
         ↪then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMEVTYPERn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && UInt(PMSELR_EL0.SEL) != 31 && UInt(PMSELR_EL0.SEL) >=
         ↪GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1' && ((UInt(PMSELR_EL0.SEL) != 31
     ↪&& PMUACR_EL1[UInt(PMSELR_EL0.SEL)] == '0') || (UInt(PMSELR_EL0.SEL) == 31 && PMUACR_EL1.C ==
     ↪'0')) then
        X[t, 64] = Zeros(64);
    elsif UInt(PMSELR_EL0.SEL) == 31 then
        X[t, 64] = PMCCFILTR_EL0;
    else
        X[t, 64] = PMEVTYPER_EL0[UInt(PMSELR_EL0.SEL)];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
     ↪HDFGRTR_EL2.PMEVTYPERn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && UInt(PMSELR_EL0.SEL) != 31 && UInt(PMSELR_EL0.SEL) >=
     ↪GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif UInt(PMSELR_EL0.SEL) == 31 then
        X[t, 64] = PMCCFILTR_EL0;
    else
        X[t, 64] = PMEVTYPER_EL0[UInt(PMSELR_EL0.SEL)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif UInt(PMSELR_EL0.SEL) == 31 then
        X[t, 64] = PMCCFILTR_EL0;
    else
        X[t, 64] = PMEVTYPER_EL0[UInt(PMSELR_EL0.SEL)];
elsif PSTATE.EL == EL3 then
    if UInt(PMSELR_EL0.SEL) == 31 then
        X[t, 64] = PMCCFILTR_EL0;
    else
        X[t, 64] = PMEVTYPER_EL0[UInt(PMSELR_EL0.SEL)];
```

```
MSR PMXEVTYPER_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1101	0b001

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_PMUv3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif UInt(PMSELR_EL0.SEL) != 31 && UInt(PMSELR_EL0.SEL) >= GetNumEventCountersSelfHosted() then
    if IsFeatureImplemented(FEAT_FGT) then
        UNDEFINED;
    else
        ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')
         ↪then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMEVTYPERn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && UInt(PMSELR_EL0.SEL) != 31 && UInt(PMSELR_EL0.SEL) >=
         ↪GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif IsFeatureImplemented(FEAT_PMUv3p9) && PMUSERENR_EL0.UEN == '1' && ((UInt(PMSELR_EL0.SEL) != 31
         ↪&& (PMUACR_EL1[UInt(PMSELR_EL0.SEL)] == '0' || PMUSERENR_EL0.ER == '1')) ||
         ↪(UInt(PMSELR_EL0.SEL) == 31 && (PMUACR_EL1.C == '0' || PMUSERENR_EL0.CR == '1'))) then
        return;
    elsif UInt(PMSELR_EL0.SEL) == 31 then
        PMCCFILTR_EL0 = X[t, 64];
    else
        PMEVTYPER_EL0[UInt(PMSELR_EL0.SEL)] = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.PMEVTYPERn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && UInt(PMSELR_EL0.SEL) != 31 && UInt(PMSELR_EL0.SEL) >=
         ↪GetNumEventCountersAccessible() then
        if !IsFeatureImplemented(FEAT_FGT) then
            ConstrainUnpredictableProcedure(Unpredictable_PMUEVENTCOUNTER);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif UInt(PMSELR_EL0.SEL) == 31 then
        PMCCFILTR_EL0 = X[t, 64];
    else
        PMEVTYPER_EL0[UInt(PMSELR_EL0.SEL)] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif UInt(PMSELR_EL0.SEL) == 31 then
        PMCCFILTR_EL0 = X[t, 64];
    else
        PMEVTYPER_EL0[UInt(PMSELR_EL0.SEL)] = X[t, 64];
elsif PSTATE.EL == EL3 then
    if UInt(PMSELR_EL0.SEL) == 31 then
        PMCCFILTR_EL0 = X[t, 64];
    else
        PMEVTYPER_EL0[UInt(PMSELR_EL0.SEL)] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.29 PMZR_EL0, Performance Monitors Zero with Mask

The PMZR_EL0 characteristics are:

Purpose

Zero the set of counters specified by the mask written to PMZR_EL0.

Configuration

When FEAT_PMUv3_EXT is implemented and FEAT_PMUv3p9 is implemented, AArch64 System register PMZR_EL0 bits [63:0] are architecturally mapped to External register PMZR_EL0[63:0].

This register is present only when FEAT_PMUv3p9 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to PMZR_EL0 are UNDEFINED.

Attributes

PMZR_EL0 is a 64-bit register.

Field descriptions

Bits [63:33]

Reserved, RES0.

F0, bit [32]

When FEAT_PMUv3_ICNTR is implemented:

Zero PMICNTR_EL0.

F0	Meaning
0b0	Write is ignored.
0b1	Set PMICNTR_EL0 to zero.

Accessing this field has the following behavior:

- Access to this field is RAZ/WI if all of the following are true:
  - EL3 is implemented
  - PSTATE.EL != EL3
  - MDCR_EL3.EnPM2 == ‘0’
- Access to this field is RAZ/WI if all of the following are true:
  - PSTATE.EL == EL0
  - Any of the following are true:
    - PMUSERENR_EL0.UEN == ‘0’
    - PMUACR_EL1.F0 == ‘0’
- Access to this field is RAZ/WI if all of the following are true:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- FEAT_FGT2 is implemented
- EL2Enabled()
- PSTATE.EL IN {EL1, EL0}
- [HCR_EL2.E2H, HCR_EL2.TGE] != ‘11’
- Any of the following are true:
  - All of the following are true:
    - EL3 is implemented
    - SCR_EL3.FGTEn2 == ‘0’
  - HDFGWTR2_EL2.nPMICNTR_EL0 == ‘0’
- Access to this field is RAZ/WI if all of the following are true:
  - PSTATE.EL == EL0
  - PMUSERENR_EL0.IR == ‘1’
- Otherwise, access to this field is WO/RAZ.

Otherwise:

Reserved, RES0.

C, bit [31]

Zero PMCCNTR_EL0.

C	Meaning
0b0	Write is ignored.
0b1	Set PMCCNTR_EL0 to zero.

Accessing this field has the following behavior:

- Access to this field is RAZ/WI if all of the following are true:
  - FEAT_PMUv3p9 is implemented
  - PSTATE.EL == EL0
  - PMUSERENR_EL0.UEN == ‘1’
  - PMUACR_EL1.C == ‘0’
- Access to this field is RAZ/WI if all of the following are true:
  - FEAT_PMUv3p9 is implemented
  - PSTATE.EL == EL0
  - [PMUSERENR_EL0.UEN, PMUSERENR_EL0.CR] == ‘11’
- Otherwise, access to this field is WO/RAZ.

P<m> , bits [m], for m = 30 to 0

Zero PMEVCNTR<m>_EL0.

P<m>	Meaning
0b0	Write is ignored.
0b1	Set PMEVCNTR<m>_EL0 to zero.

Accessing this field has the following behavior:

- When m >= GetNumEventCountersAccessible(), access to this field is RAZ/WI.
- Access to this field is RAZ/WI if all of the following are true:
  - FEAT_PMUv3p9 is implemented

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

    - PSTATE.EL == EL0
    - PMUSERENR_EL0.UEN == ‘1’
    - PMUACR_EL1.P<m> == ‘0’
  - Access to this field is RAZ/WI if all of the following are true:
    - FEAT_PMUv3p9 is implemented
    - PSTATE.EL == EL0
    - [PMUSERENR_EL0.UEN, PMUSERENR_EL0.ER] == ‘11’
  - Otherwise, access to this field is WO/RAZ.

Accessing PMZR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MSR PMZR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1001	0b1101	0b100

```
if !(IsFeatureImplemented(FEAT_PMUv3p9) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' && (!IsFeatureImplemented(FEAT_PMUv3p9) || PMUSERENR_EL0.UEN == '0')
         ↪then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
         ↪SCR_EL3.FGTEn2 == '0') || HDFGWTR2_EL2.nPMZR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ZeroPMUCounters(X[t, 64]);
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGWTR2_EL2.nPMZR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ZeroPMUCounters(X[t, 64]);
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ZeroPMUCounters(X[t, 64]);
elsif PSTATE.EL == EL3 then
    ZeroPMUCounters(X[t, 64]);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.30 SPMACCESSR_EL1, System Performance Monitors Access Register (EL1)

The SPMACCESSR_EL1 characteristics are:

Purpose

Controls access to System PMUs from EL0.

Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMACCESSR_EL1 are UNDEFINED.

Attributes

SPMACCESSR_EL1 is a 64-bit register.

Field descriptions

63 62	61 60	59 58	57 56	55 54	53 52	51 50	49 48	47 46	45 44	43 42	41 40	39 38	37 36	35 34	33 32
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

P<m> , bits [2m+1:2m], for m = 31 to 0

System PMU <m> access. Controls access to System PMU <m>.

P<m>	Meaning
0b00	MRS read and MSR write System register accesses to System PMU <m> at EL0 are trapped to EL1, unless the instruction generates a higher priority exception.
0b01	MSR write System register accesses to System PMU <m> at EL0 are trapped to EL1, unless the instruction generates a higher priority exception.
0b11	This control does not cause any instructions to be trapped.

All other values are reserved.

The registers trapped by this control are:

AArch64: SPMCNTENCLR_EL0, SPMCNTENSET_EL0, SPMCR_EL0, SPMEVCNTR<n>_EL0, SPMEVFILT2R<n>_EL0, SPMEVFILTR<n>_EL0, SPMEVTYPER<n>_EL0, SPMOVSCLR_EL0, and SPMOVSSET_EL0.

This field is ignored by the PE when all of the following are true:

- EL2 is implemented and enabled in the current Security state.
- The Effective value of HCR_EL2.{E2H,TGE} is {1,1}.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m > UInt(ID_AA64DFR1_EL1.SYSPMUID), access to this field is RES0.
- Otherwise, access to this field is RW.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing SPMACCESSR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name `SPMACCESSR_EL1` or `SPMACCESSR_EL12` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMACCESSR_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b1001	0b1101	0b011

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGRTR2_EL2.nSPMACCESSR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x8E8];
    else
        X[t, 64] = SPMACCESSR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = SPMACCESSR_EL2;
    else
        X[t, 64] = SPMACCESSR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMACCESSR_EL1;
```

```
MSR SPMACCESSR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b1001	0b1101	0b011

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGWTR2_EL2.nSPMACCESSR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x8E8] = X[t, 64];
    else
        SPMACCESSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        SPMACCESSR_EL2 = X[t, 64];
    else
        SPMACCESSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPMACCESSR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, SPMACCESSR_EL12

op0	op1	CRn	CRm	op2
0b10	0b101	0b1001	0b1101	0b011

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x8E8];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = SPMACCESSR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = SPMACCESSR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented MSR SPMACCESSR_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b10	0b101	0b1001	0b1101	0b011

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x8E8] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            SPMACCESSR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        SPMACCESSR_EL1 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.31 SPMACCESSR_EL2, System Performance Monitors Access Register (EL2)

The SPMACCESSR_EL2 characteristics are:

Purpose

Controls access to System PMUs from EL1 and EL0.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMACCESSR_EL2 are UNDEFINED.

Attributes

SPMACCESSR_EL2 is a 64-bit register.

Field descriptions

63 62	61 60	59 58	57 56	55 54	53 52	51 50	49 48	47 46	45 44	43 42	41 40	39 38	37 36	35 34	33 32
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

P<m> , bits [2m+1:2m], for m = 31 to 0

System PMU <m> access. Controls access to System PMU <m>.

P<m>	Meaning
0b00	MRS read and MSR write System register accesses to System PMU <m> at EL1 and EL0 are trapped to EL2, unless the instruction generates a higher priority exception.
0b01	MSR write System register accesses to System PMU <m> at EL1 and EL0 are trapped to EL2, unless the instruction generates a higher priority exception.
0b11	This control does not cause any instructions to be trapped.

All other values are reserved.

The registers trapped by this control are:

AArch64: SPMCFGR_EL1, SPMCGCR<n>_EL1, SPMCNTENCLR_EL0, SPMCNTENSET_EL0, SPMCR_EL0, SPMDEVAFF_EL1, SPMDEVARCH_EL1, SPMEVCNTR<n>_EL0, SPMEVFILT2R<n>_EL0, SPMEVFILTR<n>_EL0, SPMEVTYPER<n>_EL0, SPMIIDR_EL1, SPMINTENCLR_EL1, SPMINTENSET_EL1, SPMOVSCLR_EL0, SPMOVSSET_EL0, and SPMSCR_EL1.

This field is ignored by the PE when EL2 is not implemented or disabled in the current Security state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m > UInt(ID_AA64DFR1_EL1.SYSPMUID), access to this field is RES0.
- Otherwise, access to this field is RW.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing SPMACCESSR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name `SPMACCESSR_EL2` or `SPMACCESSR_EL1` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMACCESSR_EL2
```

op0	op1	CRn	CRm	op2
0b10	0b100	0b1001	0b1101	0b011

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMACCESSR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMACCESSR_EL2;
```

```
MSR SPMACCESSR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b100	0b1001	0b1101	0b011

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMACCESSR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPMACCESSR_EL2 = X[t, 64];
```

MRS <Xt>, SPMACCESSR_EL1

op0	op1	CRn	CRm	op2
0b10	0b000	0b1001	0b1101	0b011

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGRTR2_EL2.nSPMACCESSR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x8E8];
    else
        X[t, 64] = SPMACCESSR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = SPMACCESSR_EL2;
    else
        X[t, 64] = SPMACCESSR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMACCESSR_EL1;
```

MSR SPMACCESSR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b10	0b000	0b1001	0b1101	0b011

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGWTR2_EL2.nSPMACCESSR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x8E8] = X[t, 64];
    else
        SPMACCESSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        SPMACCESSR_EL2 = X[t, 64];
    else
        SPMACCESSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPMACCESSR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.32 SPMACCESSR_EL3, System Performance Monitors Access Register (EL3)

The SPMACCESSR_EL3 characteristics are:

Purpose

Controls access to System PMUs from EL2, EL1 and EL0.

Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMACCESSR_EL3 are UNDEFINED.

Attributes

SPMACCESSR_EL3 is a 64-bit register.

Field descriptions

63 62	61 60	59 58	57 56	55 54	53 52	51 50	49 48	47 46	45 44	43 42	41 40	39 38	37 36	35 34	33 32
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

P<m> , bits [2m+1:2m], for m = 31 to 0

System PMU <m> access. Controls access to System PMU <m>.

P<m>	Meaning
0b00	MRS read and MSR write System register accesses to System PMU <m> at EL2, EL1, and EL0 are trapped to EL3, unless the instruction generates a higher priority exception.
0b01	MSR write System register accesses to System PMU <m> at EL2, EL1, and EL0 are trapped to EL3, unless the instruction generates a higher priority exception.
0b11	This control does not cause any instructions to be trapped.

All other values are reserved.

The registers trapped by this control are:

AArch64: SPMCFGR_EL1, SPMCGCR<n>_EL1, SPMCNTENCLR_EL0, SPMCNTENSET_EL0, SPMCR_EL0, SPMDEVAFF_EL1, SPMDEVARCH_EL1, SPMEVCNTR<n>_EL0, SPMEVFILT2R<n>_EL0, SPMEVFILTR<n>_EL0, SPMEVTYPER<n>_EL0, SPMIIDR_EL1, SPMINTENCLR_EL1, SPMINTENSET_EL1, SPMOVSCLR_EL0, SPMOVSSET_EL0, and SPMSCR_EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When m > UInt(ID_AA64DFR1_EL1.SYSPMUID), access to this field is RES0.
- Otherwise, access to this field is RW.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing SPMACCESSR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMACCESSR_EL3
```

op0	op1	CRn	CRm	op2
0b10	0b110	0b1001	0b1101	0b011

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMACCESSR_EL3;
```

```
MSR SPMACCESSR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b110	0b1001	0b1101	0b011

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    SPMACCESSR_EL3 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.33 SPMCFGR_EL1, System Performance Monitors Configuration Register

The SPMCFGR_EL1 characteristics are:

Purpose

Describes the capabilities of System PMU <s>.

Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMCFGR_EL1 are UNDEFINED.

Attributes

SPMCFGR_EL1 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

NCG, bits [31:28]

Counter Groups.

Defines the number of counter groups implemented by System PMU <s>, minus one.

If this field is zero, then one counter group is implemented and SPMCGCR<n>_EL1 read-as-zero.

Otherwise, for each counter group <m>, SPMCGCR<m DIV 8>_EL1.N<m MOD 8> defines the number of counters in the group.

Locating the first counter in each group depends on the number of implemented groups. Each counter group starts with counter:

- SPMEVTYPER<m×32>_EL0, meaning there are at most 32 counters per group, if there are 2 counter groups.
- SPMEVTYPER<m×16>_EL0, meaning there are at most 16 counters per group, if there are 3 or 4 counter groups.
- SPMEVTYPER<m×8>_EL0, meaning there are at most 8 counters per group, if there are between 5 and 8 counter groups.
- SPMEVTYPER<m×4>_EL0, meaning there are at most 4 counters per group, if there are more than 8 counter groups.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Bits [27:25]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

HDBG, bit [24]

Halt-on-debug supported. For more information on this field, see ‘CoreSight PMU Architecture’.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

TRO, bit [23]

Trace output supported. For more information on this field, see ‘CoreSight PMU Architecture’.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

SS, bit [22]

Snapshot supported. For more information on this field, see ‘CoreSight PMU Architecture’.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

FZO, bit [21]

Freeze-on-overflow supported. For more information on this field, see ‘CoreSight PMU Architecture’.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

MSI, bit [20]

Message-signaled interrupts supported. For more information on this field, see ‘CoreSight PMU Architecture’.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Bit [19]

Reserved, RAO.

Bit [18]

Reserved, RES0.

NA, bit [17]

No write access when running. For more information on this field, see ‘CoreSight PMU Architecture’.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

EX, bit [16]

Export supported. For more information on this field, see ‘CoreSight PMU Architecture’.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Bits [15:14]

Reserved, RAZ.

SIZE, bits [13:8]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Counter size. The size of the largest counter implemented by System PMU <s>.

SIZE	Meaning
0b000111	8-bit counters.
0b001001	10-bit counters.
0b001011	12-bit counters.
0b001111	16-bit counters.
0b010011	20-bit counters.
0b010111	24-bit counters.
0b011111	32-bit counters.
0b100011	36-bit counters.
0b100111	40-bit counters.
0b101011	44-bit counters.
0b101111	48-bit counters.
0b110011	52-bit counters.
0b110111	56-bit counters.
0b111111	64-bit counters.

All other values are reserved.

Not all counters must be this size. For example, a System PMU might include a mix of 32-bit and 64-bit counters.

N, bits [7:0]

Number of event counters implemented by System PMU <s>, minus 1.

N	Meaning
0x00..0x3F	Number of event counters implemented by System PMU <s>, minus 1.

All other values are reserved.

Accessing SPMCFGR_EL1

To access SPMCFGR_EL1 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s.

SPMCFGR_EL1 reads-as-zero if System PMU <s> is not implemented.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMCFGR_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b1001	0b1101	0b111

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGRTR2_EL2.nSPMID == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMCFGR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMCFGR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMCFGR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.34 SPMCGCR<n>_EL1, System PMU Counter Group Configuration Registers, n = 0 - 1

The SPMCGCR<n>_EL1 characteristics are:

Purpose

Describes the configuration of counter groups in System PMU <s>.

Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMCGCR<n>_EL1 are UNDEFINED.

Attributes

SPMCGCR<n>_EL1 is a 64-bit register.

Field descriptions

N<m> , bits [8m+7:8m], for m = 7 to 0

Number of counters in counter group 8n+m.

The maximum size of each counter group depends on the number of implemented groups and the largest implemented counter size. For more information, see SPMCFGR_EL1.NCG.

Accessing SPMCGCR<n>_EL1

To access SPMCGCR<n>_EL1 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s.

SPMCGCR<n>_EL1 reads-as-zero if any of the following are true:

- System PMU <s> implements one counter group (SPMCFGR_EL1.NCG is zero).
- System PMU <s> is not implemented.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMCGCR<m>_EL1 ; Where m = 0-1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b1001	0b1101	0b00:m[0]

```
integer m = UInt(op2<0>);

if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGRTR2_EL2.nSPMID == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMCGCR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL), m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMCGCR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL), m];
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMCGCR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL), m];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.35 SPMCNTENCLR_EL0, System Performance Monitors Count Enable Clear Register

The SPMCNTENCLR_EL0 characteristics are:

Purpose

Disable event counters in System PMU <s>.

Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMCNTENCLR_EL0 are UNDEFINED.

Attributes

SPMCNTENCLR_EL0 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
P63	P62	P61	P60	P59	P58	P57	P56	P55	P54	P53	P52	P51	P50	P49	P48	P47	P46	P45	P44	P43	P42	P41	P40	P39	P38	P37	P36	P35	P34	P33	P32

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

P<m> , bits [m], for m = 63 to 0

Event counter <m> disable.

P<m>	Meaning
0b0	Event counter <m> in System PMU <s> is disabled.
0b1	Event counter <m> in System PMU <s> is enabled.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When event counter <m> is not implemented by System PMU <s>, access to this field is RAZ/WI.
- Otherwise, access to this field is W1C.

Accessing SPMCNTENCLR_EL0

To access SPMCNTENCLR_EL0 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s.

SPMCNTENCLR_EL0 reads-as-zero and ignores writes if System PMU <s> is not implemented.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMCNTENCLR_EL0
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b10	0b011	0b1001	0b1100	0b010

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
         ↪SCR_EL3.FGTEn2 == '0') || HDFGRTR2_EL2.nSPMCNTEN == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMCNTENCLR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGRTR2_EL2.nSPMCNTEN == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        X[t, 64] = SPMCNTENCLR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
     ↪'00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMCNTENCLR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMCNTENCLR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
```

MSR SPMCNTENCLR_EL0, <Xt>

op0	op1	CRn	CRm	op2
0b10	0b011	0b1001	0b1100	0b010

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
     ↪'11' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
     ↪SCR_EL3.FGTEn2 == '0') || HDFGWTR2_EL2.nSPMCNTEN == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMCNTENCLR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
         ↪'11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGWTR2_EL2.nSPMCNTEN == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMCNTENCLR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
         ↪'11' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMCNTENCLR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPMCNTENCLR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.36 SPMCNTENSET_EL0, System Performance Monitors Count Enable Set Register

The SPMCNTENSET_EL0 characteristics are:

Purpose

Enables event counters in System PMU <s>.

Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMCNTENSET_EL0 are UNDEFINED.

Attributes

SPMCNTENSET_EL0 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
P63	P62	P61	P60	P59	P58	P57	P56	P55	P54	P53	P52	P51	P50	P49	P48	P47	P46	P45	P44	P43	P42	P41	P40	P39	P38	P37	P36	P35	P34	P33	P32

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

P<m> , bits [m], for m = 63 to 0

Event counter <m> enable.

P<m>	Meaning
0b0	Event counter <m> in System PMU <s> is disabled.
0b1	Event counter <m> in System PMU <s> is enabled.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When event counter <m> is not implemented by System PMU <s>, access to this field is RAZ/WI.
- Otherwise, access to this field is W1S.

Accessing SPMCNTENSET_EL0

To access SPMCNTENSET_EL0 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s.

SPMCNTENSET_EL0 reads-as-zero and ignores writes if System PMU <s> is not implemented.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMCNTENSET_EL0
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b10	0b011	0b1001	0b1100	0b001

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
         ↪SCR_EL3.FGTEn2 == '0') || HDFGRTR2_EL2.nSPMCNTEN == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMCNTENSET_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGRTR2_EL2.nSPMCNTEN == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        X[t, 64] = SPMCNTENSET_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
     ↪'00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMCNTENSET_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMCNTENSET_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
```

```
MSR SPMCNTENSET_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1001	0b1100	0b001

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
     ↪'11' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
     ↪SCR_EL3.FGTEn2 == '0') || HDFGWTR2_EL2.nSPMCNTEN == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMCNTENSET_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
         ↪'11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGWTR2_EL2.nSPMCNTEN == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMCNTENSET_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
         ↪'11' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMCNTENSET_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPMCNTENSET_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.37 SPMCR_EL0, System Performance Monitor Control Register

The SPMCR_EL0 characteristics are:

Purpose

Main control register for System PMU <s>.

Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMCR_EL0 are UNDEFINED.

Attributes

SPMCR_EL0 is a 64-bit register.

Field descriptions

Bits [63:12]

Reserved, RES0.

TRO, bit [11]

When SPMCFGR_EL1.TRO == '1':

Trace enable. For more information on this field, see ‘CoreSight PMU Architecture’.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When PSTATE.EL == EL0, access to this field is RO.
- Otherwise, access to this field is RW.

Otherwise:

Reserved, RES0.

HDBG, bit [10]

When SPMCFGR_EL1.HDBG == '1':

Halt-on-debug. For more information on this field, see ‘CoreSight PMU Architecture’.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When PSTATE.EL == EL0, access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- Otherwise, access to this field is RW.

Otherwise:

Reserved, RES0.

FZO, bit [9]

When SPMCFGR_EL1.FZO == '1':

Freeze-on-overflow. For more information on this field, see ‘CoreSight PMU Architecture’.

Note

If implemented by a System PMU, then freeze-on-overflow affects only the counters of System PMU <s>, not other System PMUs nor the PE PMU.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NA, bit [8]

When SPMCFGR_EL1.NA == '1':

Not accessible. For more information on this field, see ‘CoreSight PMU Architecture’.

Access to this field is RO.

Otherwise:

Reserved, RES0.

Bits [7:5]

Reserved, RES0.

EX, bit [4]

When SPMCFGR_EL1.EX == '1':

Export enable. For more information on this field, see ‘CoreSight PMU Architecture’.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [3:2]

Reserved, RES0.

P, bit [1]

Event counter reset.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

P	Meaning
0b0	Write is ignored.
0b1	Reset all event counters in System PMU <s> to zero.

Note

Resetting the event counters does not affect any overflow flags.

Access to this field is WO/RAZ.

E, bit [0]

Count enable. This field controls System PMU <s>.

E	Meaning
0b0	Monitor is disabled.
0b1	Monitor is enabled.

Performance monitor overflow IRQs are only signaled by System PMU <s> when this field is 1.

The reset behavior of this field is:

- On a System PMU reset, this field resets to `'0'`.

Accessing SPMCR_EL0

To access SPMCR_EL0 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s.

SPMCR_EL0 reads-as-zero and ignores writes if System PMU <s> is not implemented.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMCR_EL0
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1001	0b1100	0b000

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
     ↪'00' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
                AArch64.SystemAccessTrap(EL2, 0x18);
            else
                AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
             ↪SCR_EL3.FGTEn2 == '0') || HDFGRTR2_EL2.nSPMCR_EL0 == '0') then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = SPMCR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
    elsif PSTATE.EL == EL1 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
             ↪'00' then
            UNDEFINED;
        elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
             ↪HDFGRTR2_EL2.nSPMCR_EL0 == '0') then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = SPMCR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
             ↪'00' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = SPMCR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
    elsif PSTATE.EL == EL3 then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    X[t, 64] = SPMCR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
```

```
MSR SPMCR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1001	0b1100	0b000

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
         ↪'11' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
         ↪SCR_EL3.FGTEn2 == '0') || HDFGWTR2_EL2.nSPMCR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMCR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
         ↪'11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGWTR2_EL2.nSPMCR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMCR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
     ↪'11' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMCR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPMCR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.38 SPMDEVAFF_EL1, System Performance Monitors Device Affinity Register

The SPMDEVAFF_EL1 characteristics are:

Purpose

For additional information, see the CoreSight Architecture Specification.

For a System PMU that has affinity with a single PE or a group of PEs, SPMDEVAFF_EL1 is a copy of MPIDR_EL1 or part of MPIDR_EL1:

- If the System PMU has affinity with a single PE, then the affinity level is 0 and SPMDEVAFF_EL1 reads the same value as MPIDR_EL1, and SPMDEVAFF_EL1.F0V reads-as-one to indicate affinity level 0.
- If the System PMU has affinity with a group of PEs, then the affinity level is 1, 2, or 3, parts of SPMDEVAFF_EL1 reads the same value as parts of MPIDR_EL1, and the rest of SPMDEVAFF_EL1 indicates the level.

For example, if the group of PEs is a subset of the PEs at affinity level 1, then all of the following are true:

- All the PEs in the group have the same values in MPIDR_EL1.{Aff3,Aff2}, and these values are equal to SPMDEVAFF_EL1.{Aff3,Aff2}.
- SPMDEVAFF_EL1.Aff1 is nonzero and not `0x80`, and SPMDEVAFF_EL1.{Aff0,F0V} read-as-zero, to indicate at least affinity level 1. The subset of PEs at level 1 that the System PMU has affinity with is indicated by the least-significant set bit in SPMDEVAFF_EL1.Aff1. In this example, if SPMDEVAFF_EL1.Aff1[2:0] is `0b100`, then the System PMU has affinity with the up-to 8 PEs that have MPIDR_EL1.Aff1[7:3] == SPMDEVAFF_EL1.Aff1[7:3].

Depending on the IMPLEMENTATION DEFINED nature of the system, it might be possible that SPMDEVAFF_EL1 is read before system firmware has configured the System PMU and/or the PE or group of PEs that the System PMU has affinity with. When this is the case, SPMDEVAFF_EL1 reads-as-zero.

Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMDEVAFF_EL1 are UNDEFINED.

Attributes

SPMDEVAFF_EL1 is a 64-bit register.

Field descriptions

Bits [63:40]

Reserved, RES0.

Aff3, bits [39:32]

PE affinity level 3. The MPIDR_EL1.Aff3 field, viewed from the highest Exception level of the associated PE or PEs.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F0V, bit [31]

Indicates that the SPMDEVAFF_EL1.Aff0 field is valid.

The value of this field is an IMPLEMENTATION DEFINED choice of:

F0V	Meaning
0b0	SPMDEVAFF_EL1.Aff0 is not valid, and the PE affinity is above level 0 or a subset of level 0.
0b1	SPMDEVAFF_EL1.Aff0 is valid, and the PE affinity is at level 0.

Access to this field is RO.

U, bit [30]

When SPMDEVAFF_EL1.F0V == '1':

Uniprocessor. The MPIDR_EL1.U field, viewed from the highest Exception level of the associated PE.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Otherwise:

Reserved, UNKNOWN.

Bits [29:25]

Reserved, RES0.

MT, bit [24]

When SPMDEVAFF_EL1.F0V == '1':

Multithreaded. The MPIDR_EL1.MT field, viewed from the highest Exception level of the associated PE.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Otherwise:

Reserved, UNKNOWN.

Aff2, bits [23:16]

When affine with a PE or PEs at affinity level 2 or below:

PE affinity level 2. The MPIDR_EL1.Aff2 field, viewed from the highest Exception level of the associated PE or PEs.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

When affine with a sub-set of PEs at affinity level 2:

PE affinity level 2. Defines part of the MPIDR_EL1.Aff2 field, viewed from the highest Exception level of the associated PEs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Aff2	Meaning
0bxxxxxxx1	SPMDEVAFF_EL1.Aff2[7:1] is the value of MPIDR_EL1.Aff2[7:1], viewed from the highest Exception level of the associated PEs.
0bxxxxxx10	SPMDEVAFF_EL1.Aff2[7:2] is the value of MPIDR_EL1.Aff2[7:2], viewed from the highest Exception level of the associated PEs.
0bxxxxx100	SPMDEVAFF_EL1.Aff2[7:3] is the value of MPIDR_EL1.Aff2[7:3], viewed from the highest Exception level of the associated PEs.
0bxxxx1000	SPMDEVAFF_EL1.Aff2[7:4] is the value of MPIDR_EL1.Aff2[7:4], viewed from the highest Exception level of the associated PEs.
0bxxx10000	SPMDEVAFF_EL1.Aff2[7:5] is the value of MPIDR_EL1.Aff2[7:5], viewed from the highest Exception level of the associated PEs.
0bxx100000	SPMDEVAFF_EL1.Aff2[7:6] is the value of MPIDR_EL1.Aff2[7:6], viewed from the highest Exception level of the associated PEs.
0bx1000000	SPMDEVAFF_EL1.Aff2[7] is the value of MPIDR_EL1.Aff2[7], viewed from the highest Exception level of the associated PEs.

Access to this field is RO.

Otherwise:

PE affinity level NOT DEFINED. Indicates whether the PE affinity is at level 3.

Aff2	Meaning
0x80	PE affinity is at level 3.

All other values are reserved.

Access to this field is RO.

Aff1, bits [15:8]

When affine with a PE or PEs at affinity level 1 or below:

PE affinity level 1. The MPIDR_EL1.Aff1 field, viewed from the highest Exception level of the associated PE or PEs.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

When affine with a sub-set of PEs at affinity level 1:

PE affinity level 1. Defines part of the MPIDR_EL1.Aff1 field, viewed from the highest Exception level of the associated PEs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Aff1	Meaning
0bxxxxxxx1	SPMDEVAFF_EL1.Aff1[7:1] is the value of MPIDR_EL1.Aff1[7:1], viewed from the highest Exception level of the associated PEs.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Aff1	Meaning
0bxxxxxx10	SPMDEVAFF_EL1.Aff1[7:2] is the value of MPIDR_EL1.Aff1[7:2], viewed from the highest Exception level of the associated PEs.
0bxxxxx100	SPMDEVAFF_EL1.Aff1[7:3] is the value of MPIDR_EL1.Aff1[7:3], viewed from the highest Exception level of the associated PEs.
0bxxxx1000	SPMDEVAFF_EL1.Aff1[7:4] is the value of MPIDR_EL1.Aff1[7:4], viewed from the highest Exception level of the associated PEs.
0bxxx10000	SPMDEVAFF_EL1.Aff1[7:5] is the value of MPIDR_EL1.Aff1[7:5], viewed from the highest Exception level of the associated PEs.
0bxx100000	SPMDEVAFF_EL1.Aff1[7:6] is the value of MPIDR_EL1.Aff1[7:6], viewed from the highest Exception level of the associated PEs.
0bx1000000	SPMDEVAFF_EL1.Aff1[7] is the value of MPIDR_EL1.Aff1[7], viewed from the highest Exception level of the associated PEs.

Access to this field is RO.

Otherwise:

PE affinity level 1. Indicates whether the PE affinity is at level 2.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Aff1	Meaning
0x00	PE affinity is above level 2 or a subset of level 2.
0x80	PE affinity is at level 2.

Access to this field is RO.

Aff0, bits [7:0]

When affine with a PE at affinity level 0:

PE affinity level 0. The MPIDR_EL1.Aff0 field, viewed from the highest Exception level of the associated PE.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

When affine with a sub-set of PEs at affinity level 0:

PE affinity level 0. Defines part of the MPIDR_EL1.Aff0 field, viewed from the highest Exception level of the associated PEs.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Aff0	Meaning
0bxxxxxxx1	SPMDEVAFF_EL1.Aff0[7:1] is the value of MPIDR_EL1.Aff0[7:1], viewed from the highest Exception level of the associated PEs.
0bxxxxxx10	SPMDEVAFF_EL1.Aff0[7:2] is the value of MPIDR_EL1.Aff0[7:2], viewed from the highest Exception level of the associated PEs.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Aff0	Meaning
0bxxxxx100	SPMDEVAFF_EL1.Aff0[7:3] is the value of MPIDR_EL1.Aff0[7:3], viewed from the highest Exception level of the associated PEs.
0bxxxx1000	SPMDEVAFF_EL1.Aff0[7:4] is the value of MPIDR_EL1.Aff0[7:4], viewed from the highest Exception level of the associated PEs.
0bxxx10000	SPMDEVAFF_EL1.Aff0[7:5] is the value of MPIDR_EL1.Aff0[7:5], viewed from the highest Exception level of the associated PEs.
0bxx100000	SPMDEVAFF_EL1.Aff0[7:6] is the value of MPIDR_EL1.Aff0[7:6], viewed from the highest Exception level of the associated PEs.
0bx1000000	SPMDEVAFF_EL1.Aff0[7] is the value of MPIDR_EL1.Aff0[7], viewed from the highest Exception level of the associated PEs.

Access to this field is RO.

Otherwise:

PE affinity level 0. Indicates whether the PE affinity is at level 1.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Aff0	Meaning
0x00	PE affinity is above level 1 or a subset of level 1.
0x80	PE affinity is at level 1.

Access to this field is RO.

Accessing SPMDEVAFF_EL1

Reads of SPMDEVAFF_EL1 are not affected by the value of VMPIDR_EL2 at any Exception level.

If System PMU <s> has affinity only with this PE, then it is IMPLEMENTATION DEFINED whether SPMDEVAFF_EL1 reads-as-zero or reads the same value as MPIDR_EL1.

To access SPMDEVAFF_EL1 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s.

SPMDEVAFF_EL1 reads-as-zero if any of the following are true:

- System PMU <s> is not implemented.
- System PMU <s> has no affinity with the PE or cluster of PEs.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMDEVAFF_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b1001	0b1101	0b110

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGRTR2_EL2.nSPMDEVAFF_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMDEVAFF_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMDEVAFF_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMDEVAFF_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.39 SPMDEVARCH_EL1, System Performance Monitors Device Architecture Register

The SPMDEVARCH_EL1 characteristics are:

Purpose

Provides discovery information for System PMU <s>.

Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMDEVARCH_EL1 are UNDEFINED.

Attributes

SPMDEVARCH_EL1 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

ARCHITECT, bits [31:21]

Architect. Defines the architect of the component. Bits [31:28] are the JEP106 continuation code (JEP106 bank ID, minus 1) and bits [27:21] are the JEP106 ID code.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

PRESENT, bit [20]

DEVARCH present. Defines that SPMDEVARCH_EL1 register is present.

PRESENT	Meaning
0b0	Device Architecture information not present.
0b1	Device Architecture information present.

If SPMDEVARCH_EL1 is not present, the register is RES0.

REVISION, bits [19:16]

Revision. Defines the architecture revision of the component.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ARCHVER, bits [15:12]

Architecture Version. Defines the architecture version of the component.

This field has an IMPLEMENTATION DEFINED value.

SPMDEVARCH_EL1.ARCHVER and SPMDEVARCH_EL1.ARCHPART are also defined as a single field, SPMDEVARCH_EL1.ARCHID, so that SPMDEVARCH_EL1.ARCHVER is SPMDEVARCH_EL1.ARCHID[15:12].

Access to this field is RO.

ARCHPART, bits [11:0]

Architecture Part. Defines the architecture of the component.

This field has an IMPLEMENTATION DEFINED value.

SPMDEVARCH_EL1.ARCHVER and SPMDEVARCH_EL1.ARCHPART are also defined as a single field, SPMDEVARCH_EL1.ARCHID, so that SPMDEVARCH_EL1.ARCHPART is SPMDEVARCH_EL1.ARCHID[11:0].

Access to this field is RO.

Accessing SPMDEVARCH_EL1

To access SPMDEVARCH_EL1 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s.

SPMDEVARCH_EL1 reads-as-zero if any of the following are true:

- System PMU <s> is not implemented.
- System PMU <s> does not implement SPMDEVARCH_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMDEVARCH_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b1001	0b1101	0b101

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGRTR2_EL2.nSPMID == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMDEVARCH_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
     ↪'00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMDEVARCH_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMDEVARCH_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.40 SPMEVCNTR<n>_EL0, System Performance Monitors Event Count Register, n = 0 - 63

The SPMEVCNTR<n>_EL0 characteristics are:

Purpose

Event counter <n> in System PMU <s>, where n is 0 to 63.

Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMEVCNTR<n>_EL0 are UNDEFINED.

Attributes

SPMEVCNTR<n>_EL0 is a 64-bit register.

Field descriptions

CNTR, bits [63:0]

Event counter n.

The number of implemented bits for SPMEVCNTR<n>_EL0 is IMPLEMENTATION DEFINED. Unimplemented bits are RES0.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an architecturally UNKNOWN value.

Accessing SPMEVCNTR<n>_EL0

To access SPMEVCNTR<n>_EL0 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s and SPMSELR_EL0.BANK to n[5:4].

SPMEVCNTR<n>_EL0 reads-as-zero and ignores writes if any of the following are true:

- Event counter <n> is not implemented by System PMU <s>.
- System PMU <s> is not implemented.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMEVCNTR<m>_EL0 ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1110	0b000:m[3]	m[2:0]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
integer m = UInt(CRm<0>:op2<2:0>);

if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
         ↪SCR_EL3.FGTEn2 == '0') || HDFGRTR2_EL2.nSPMEVCNTRn_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m) then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = SPMEVCNTR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGRTR2_EL2.nSPMEVCNTRn_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m) then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = SPMEVCNTR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m) then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = SPMEVCNTR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m];
elsif PSTATE.EL == EL3 then
    if !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m) then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = SPMEVCNTR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m];
```

```
MSR SPMEVCNTR<m>_EL0, <Xt> ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1110	0b000:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
         ↪'11' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
         ↪SCR_EL3.FGTEn2 == '0') || HDFGWTR2_EL2.nSPMEVCNTRn_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m) then
            return;
        else
            SPMEVCNTR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m] = X[t, 64];
    elsif PSTATE.EL == EL1 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
             ↪'11' then
            UNDEFINED;
        elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
             ↪HDFGWTR2_EL2.nSPMEVCNTRn_EL0 == '0') then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m) then
            return;
        else
            SPMEVCNTR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m] = X[t, 64];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
             ↪'11' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m) then
            return;
        else
            SPMEVCNTR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m] = X[t, 64];
    elsif PSTATE.EL == EL3 then
        if !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m) then
            return;
        else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
SPMEVCNTR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.41 SPMEVFILT2R<n>_EL0, System Performance Monitors Event Filter Control Register 2, n = 0 - 63

The SPMEVFILT2R<n>_EL0 characteristics are:

Purpose

With SPMEVTYPER<n>_EL0 and SPMEVFILTR<n>_EL0, configures when event counter SPMEVCNTR<n>_EL0 in System PMU <s> increments.

The contents of this register are IMPLEMENTATION DEFINED. For more information, see SPMEVTYPER<n>_EL0.

Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMEVFILT2R<n>_EL0 are UNDEFINED.

Attributes

SPMEVFILT2R<n>_EL0 is a 64-bit register.

Field descriptions

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an IMPLEMENTATION DEFINED value.

Accessing SPMEVFILT2R<n>_EL0

To access SPMEVFILT2R<n>_EL0 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s and SPMSELR_EL0.BANK to n[5:4].

SPMEVFILT2R<n>_EL0 reads-as-zero and ignores writes if any of the following are true:

- Event counter <n> is not implemented by System PMU <s>.
- System PMU <s> is not implemented.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMEVFILT2R<m>_EL0 ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1110	0b011:m[3]	m[2:0]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
integer m = UInt(CRm<0>:op2<2:0>);

if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
         ↪SCR_EL3.FGTEn2 == '0') || HDFGRTR2_EL2.nSPMEVTYPERn_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m) then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = SPMEVFILT2R_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGRTR2_EL2.nSPMEVTYPERn_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m) then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = SPMEVFILT2R_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m) then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = SPMEVFILT2R_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m];
elsif PSTATE.EL == EL3 then
    if !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m) then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = SPMEVFILT2R_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m];
```

```
MSR SPMEVFILT2R<m>_EL0, <Xt> ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1110	0b011:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
         ↪'11' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
         ↪SCR_EL3.FGTEn2 == '0') || HDFGWTR2_EL2.nSPMEVTYPERn_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m) then
            return;
        else
            SPMEVFILT2R_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m] = X[t, 64];
    elsif PSTATE.EL == EL1 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
             ↪'11' then
            UNDEFINED;
        elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
             ↪HDFGWTR2_EL2.nSPMEVTYPERn_EL0 == '0') then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m) then
            return;
        else
            SPMEVFILT2R_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m] = X[t, 64];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
             ↪'11' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m) then
            return;
        else
            SPMEVFILT2R_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m] = X[t, 64];
    elsif PSTATE.EL == EL3 then
        if !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m) then
            return;
        else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
SPMEVFILT2R_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.42 SPMEVFILTR<n>_EL0, System Performance Monitors Event Filter Control Register, n = 0 - 63

The SPMEVFILTR<n>_EL0 characteristics are:

Purpose

With SPMEVTYPER<n>_EL0 and SPMEVFILT2R<n>_EL0, configures when event counter SPMEVCNTR<n>_EL0 in System PMU <s> increments.

The contents of this register are IMPLEMENTATION DEFINED. For more information, see SPMEVTYPER<n>_EL0.

Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMEVFILTR<n>_EL0 are UNDEFINED.

Attributes

SPMEVFILTR<n>_EL0 is a 64-bit register.

Field descriptions

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an IMPLEMENTATION DEFINED value.

Accessing SPMEVFILTR<n>_EL0

To access SPMEVFILTR<n>_EL0 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s and SPMSELR_EL0.BANK to n[5:4].

SPMEVFILTR<n>_EL0 reads-as-zero and ignores writes if any of the following are true:

- Event counter <n> is not implemented by System PMU <s>.
- System PMU <s> is not implemented.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMEVFILTR<m>_EL0 ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1110	0b010:m[3]	m[2:0]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
integer m = UInt(CRm<0>:op2<2:0>);

if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
         ↪SCR_EL3.FGTEn2 == '0') || HDFGRTR2_EL2.nSPMEVTYPERn_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m) then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = SPMEVFILTR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGRTR2_EL2.nSPMEVTYPERn_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m) then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = SPMEVFILTR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m) then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = SPMEVFILTR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m];
elsif PSTATE.EL == EL3 then
    if !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m) then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = SPMEVFILTR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m];
```

```
MSR SPMEVFILTR<m>_EL0, <Xt> ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1110	0b010:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
         ↪'11' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
         ↪SCR_EL3.FGTEn2 == '0') || HDFGWTR2_EL2.nSPMEVTYPERn_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m) then
            return;
        else
            SPMEVFILTR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m] = X[t, 64];
    elsif PSTATE.EL == EL1 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
             ↪'11' then
            UNDEFINED;
        elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
             ↪HDFGWTR2_EL2.nSPMEVTYPERn_EL0 == '0') then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m) then
            return;
        else
            SPMEVFILTR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m] = X[t, 64];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
             ↪'11' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m) then
            return;
        else
            SPMEVFILTR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m] = X[t, 64];
    elsif PSTATE.EL == EL3 then
        if !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m) then
            return;
        else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
SPMEVFILTR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.43 SPMEVTYPER<n>_EL0, System Performance Monitors Event Type Register, n = 0 - 63

The SPMEVTYPER<n>_EL0 characteristics are:

Purpose

With SPMEVFILTR<n>_EL0 and SPMEVFILT2R<n>_EL0, configures when event counter SPMEVCNTR<n>_EL0 in System PMU <s> increments.

The contents of this register are IMPLEMENTATION DEFINED. An Event Type Select Register typically contains:

- A field defining the event that the counter is responsive to, in the least-significant bits.
- Controls for per-counter filtering, such as by mode or state.
- Additional controls, such as for a per-counter state machine.

Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMEVTYPER<n>_EL0 are UNDEFINED.

Attributes

SPMEVTYPER<n>_EL0 is a 64-bit register.

Field descriptions

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an IMPLEMENTATION DEFINED value.

Accessing SPMEVTYPER<n>_EL0

To access SPMEVTYPER<n>_EL0 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s and SPMSELR_EL0.BANK to n[5:4].

SPMEVTYPER<n>_EL0 reads-as-zero and ignores writes if any of the following are true:

- Event counter <n> is not implemented by System PMU <s>.
- System PMU <s> is not implemented.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMEVTYPER<m>_EL0 ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1110	0b001:m[3]	m[2:0]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
integer m = UInt(CRm<0>:op2<2:0>);

if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
         ↪SCR_EL3.FGTEn2 == '0') || HDFGRTR2_EL2.nSPMEVTYPERn_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m) then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = SPMEVTYPER_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGRTR2_EL2.nSPMEVTYPERn_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m) then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = SPMEVTYPER_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m) then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = SPMEVTYPER_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m];
elsif PSTATE.EL == EL3 then
    if !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m) then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = SPMEVTYPER_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m];
```

```
MSR SPMEVTYPER<m>_EL0, <Xt> ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1110	0b001:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
         ↪'11' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
         ↪SCR_EL3.FGTEn2 == '0') || HDFGWTR2_EL2.nSPMEVTYPERn_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m) then
            return;
        else
            SPMEVTYPER_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m] = X[t, 64];
    elsif PSTATE.EL == EL1 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
             ↪'11' then
            UNDEFINED;
        elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
             ↪HDFGWTR2_EL2.nSPMEVTYPERn_EL0 == '0') then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m) then
            return;
        else
            SPMEVTYPER_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m] = X[t, 64];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
             ↪'11' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m) then
            return;
        else
            SPMEVTYPER_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m] = X[t, 64];
    elsif PSTATE.EL == EL3 then
        if !IsSPMUCounterImplemented(UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m) then
            return;
        else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
SPMEVTYPER_EL0[UInt(SPMSELR_EL0.SYSPMUSEL), (UInt(SPMSELR_EL0.BANK) * 16) + m] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.44 SPMIIDR_EL1, System PMU Implementation Identification Register

The SPMIIDR_EL1 characteristics are:

Purpose

Provides discovery information for System PMU <s>.

Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMIIDR_EL1 are UNDEFINED.

Attributes

SPMIIDR_EL1 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

ProductID, bits [31:20]

Part number, bits [11:0]. The part number is selected by the designer of the component.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Variant, bits [19:16]

Component major revision.

Defines either a variant of the component defined by SPMIIDR_EL1.ProductID, or the major revision of the component.

When defining a major revision, SPMIIDR_EL1.Variant and SPMIIDR_EL1.Revision together form the revision number of the component, with SPMIIDR_EL1.Variant being the most significant part and SPMIIDR_EL1.Revision the least significant part. When a component is changed, SPMIIDR_EL1.Variant or SPMIIDR_EL1.Revision is increased to ensure that software can differentiate the different revisions of the component. If SPMIIDR_EL1.Variant is increased then SPMIIDR_EL1.Revision should be set to `0b0000`.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Revision, bits [15:12]

Component minor revision.

When a component is changed:

- If SPMIIDR_EL1.Variant and SPMIIDR_EL1.Revision together form the revision number of the component then:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- SPMIIDR_EL1.Variant or SPMIIDR_EL1.Revision is increased to ensure that software can differentiate the different revisions of the component.
- If Variant is increased then Revision should be set to `0b0000`.

- Otherwise, SPMIIDR_EL1.Revision is increased to ensure that software can differentiate the different revisions of the component.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Implementer, bits [11:0]

Contains the JEP106 manufacturer's identification code of the designer of the System PMU.

The code identifies the designer of the component, which might not be the same as the implementer of the device containing the component.

For an implementation designed by Arm, this field reads as `0x43B`.

This field has an IMPLEMENTATION DEFINED value.

Bits [11:8] contain the JEP106 bank identifier of the designer minus 1.

Bit 7 is RES0.

Bits [6:0] contain bits [6:0] of the JEP106 manufacturer's identification code of the designer.

Access to this field is RO.

Accessing SPMIIDR_EL1

To access SPMIIDR_EL1 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s.

SPMIIDR_EL1 reads-as-zero if any of the following are true:

- System PMU <s> is not implemented.
- System PMU <s> does not implement SPMIIDR_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMIIDR_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b1001	0b1101	0b100

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGRTR2_EL2.nSPMID == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMIIDR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
     ↪'00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMIIDR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMIIDR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.45 SPMINTENCLR_EL1, System Performance Monitors Interrupt Enable Clear Register

The SPMINTENCLR_EL1 characteristics are:

Purpose

Disables the generation of interrupt requests on overflows from event counters in System PMU <s>.

Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMINTENCLR_EL1 are UNDEFINED.

Attributes

SPMINTENCLR_EL1 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
P63	P62	P61	P60	P59	P58	P57	P56	P55	P54	P53	P52	P51	P50	P49	P48	P47	P46	P45	P44	P43	P42	P41	P40	P39	P38	P37	P36	P35	P34	P33	P32

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

P<m> , bits [m], for m = 63 to 0

Event counter <m> overflow interrupt request disable.

P<m>	Meaning
0b0	Event counter <m> in System PMU <s> interrupt request is disabled.
0b1	Event counter <m> in System PMU <s> interrupt request is enabled.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is RAZ/WI if any of the following are true:
  - event counter <m> is not implemented by System PMU <s>
  - event counter <m> does not implement an overflow flag
  - System PMU <s> does not implement an overflow interrupt request
- Otherwise, access to this field is W1C.

Accessing SPMINTENCLR_EL1

To access SPMINTENCLR_EL1 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s.

SPMINTENCLR_EL1 reads-as-zero and ignores writes if System PMU <s> is not implemented.

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, SPMINTENCLR_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b1001	0b1110	0b010

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGRTR2_EL2.nSPMINTEN == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMINTENCLR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMINTENCLR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMINTENCLR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
```

```
MSR SPMINTENCLR_EL1, <Xt>
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b10	0b000	0b1001	0b1110	0b010

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
         ↪'11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGWTR2_EL2.nSPMINTEN == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMINTENCLR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
         ↪'11' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMINTENCLR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPMINTENCLR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.46 SPMINTENSET_EL1, System Performance Monitors Interrupt Enable Set Register

The SPMINTENSET_EL1 characteristics are:

Purpose

Enables the generation of interrupt requests on overflows from event counters in System PMU <s>.

Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMINTENSET_EL1 are UNDEFINED.

Attributes

SPMINTENSET_EL1 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
P63	P62	P61	P60	P59	P58	P57	P56	P55	P54	P53	P52	P51	P50	P49	P48	P47	P46	P45	P44	P43	P42	P41	P40	P39	P38	P37	P36	P35	P34	P33	P32

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

P<m> , bits [m], for m = 63 to 0

Event counter <m> overflow interrupt request enable.

P<m>	Meaning
0b0	Event counter <m> in System PMU <s> interrupt request is disabled.
0b1	Event counter <m> in System PMU <s> interrupt request is enabled.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is RAZ/WI if any of the following are true:
  - event counter <m> is not implemented by System PMU <s>
  - event counter <m> does not implement an overflow flag
  - System PMU <s> does not implement an overflow interrupt request
- Otherwise, access to this field is W1S.

Accessing SPMINTENSET_EL1

To access SPMINTENSET_EL1 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s.

SPMINTENSET_EL1 reads-as-zero and ignores writes if System PMU <s> is not implemented.

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, SPMINTENSET_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b000	0b1001	0b1110	0b001

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGRTR2_EL2.nSPMINTEN == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMINTENSET_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMINTENSET_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMINTENSET_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
```

```
MSR SPMINTENSET_EL1, <Xt>
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b10	0b000	0b1001	0b1110	0b001

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
         ↪'11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGWTR2_EL2.nSPMINTEN == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMINTENSET_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
         ↪'11' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMINTENSET_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPMINTENSET_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.47 SPMOVSCLR_EL0, System Performance Monitors Overflow Flag Status Clear Register

The SPMOVSCLR_EL0 characteristics are:

Purpose

Clears the state of overflow bits for event counters in System PMU <s>.

Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMOVSCLR_EL0 are UNDEFINED.

Attributes

SPMOVSCLR_EL0 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
P63	P62	P61	P60	P59	P58	P57	P56	P55	P54	P53	P52	P51	P50	P49	P48	P47	P46	P45	P44	P43	P42	P41	P40	P39	P38	P37	P36	P35	P34	P33	P32

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

P<m> , bits [m], for m = 63 to 0

Event counter <m> unsigned overflow bit clear.

P<m>	Meaning
0b0	Event counter <m> in System PMU <s> has not overflowed.
0b1	Event counter <m> in System PMU <s> has overflowed.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is RAZ/WI if any of the following are true:
  - event counter <m> is not implemented by System PMU <s>
  - event counter <m> does not implement an overflow flag
- Otherwise, access to this field is W1C.

Accessing SPMOVSCLR_EL0

To access SPMOVSCLR_EL0 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s.

SPMOVSCLR_EL0 reads-as-zero and ignores writes if System PMU <s> is not implemented.

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, SPMOVSCLR_EL0
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1001	0b1100	0b011

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
         ↪SCR_EL3.FGTEn2 == '0') || HDFGRTR2_EL2.nSPMOVS == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMOVSCLR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGRTR2_EL2.nSPMOVS == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMOVSCLR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMOVSCLR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMOVSCLR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
```

MSR SPMOVSCLR_EL0, <Xt>

op0	op1	CRn	CRm	op2
0b10	0b011	0b1001	0b1100	0b011

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
         ↪'11' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
         ↪SCR_EL3.FGTEn2 == '0') || HDFGWTR2_EL2.nSPMOVS == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMOVSCLR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
         ↪'11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGWTR2_EL2.nSPMOVS == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMOVSCLR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
         ↪'11' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMOVSCLR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPMOVSCLR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.48 SPMOVSSET_EL0, System Performance Monitors Overflow Flag Status Set Register

The SPMOVSSET_EL0 characteristics are:

Purpose

Sets the state of overflow bits for event counters in System PMU <s>.

Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMOVSSET_EL0 are UNDEFINED.

Attributes

SPMOVSSET_EL0 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
P63	P62	P61	P60	P59	P58	P57	P56	P55	P54	P53	P52	P51	P50	P49	P48	P47	P46	P45	P44	P43	P42	P41	P40	P39	P38	P37	P36	P35	P34	P33	P32

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

P<m> , bits [m], for m = 63 to 0

Event counter <m> unsigned overflow bit set.

P<m>	Meaning
0b0	Event counter <m> in System PMU <s> has not overflowed.
0b1	Event counter <m> in System PMU <s> has overflowed.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is RAZ/WI if any of the following are true:
  - event counter <m> is not implemented by System PMU <s>
  - event counter <m> does not implement an overflow flag
- Otherwise, access to this field is W1S.

Accessing SPMOVSSET_EL0

To access SPMOVSSET_EL0 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s.

SPMOVSSET_EL0 reads-as-zero and ignores writes if System PMU <s> is not implemented.

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, SPMOVSSET_EL0
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1001	0b1110	0b011

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
         ↪SCR_EL3.FGTEn2 == '0') || HDFGRTR2_EL2.nSPMOVS == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMOVSSET_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGRTR2_EL2.nSPMOVS == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMOVSSET_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMOVSSET_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMOVSSET_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)];
```

MSR SPMOVSSET_EL0, <Xt>

op0	op1	CRn	CRm	op2
0b10	0b011	0b1001	0b1110	0b011

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
         ↪'11' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
         ↪SCR_EL3.FGTEn2 == '0') || HDFGWTR2_EL2.nSPMOVS == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMOVSSET_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
         ↪'11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGWTR2_EL2.nSPMOVS == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMOVSSET_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
         ↪'11' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMOVSSET_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPMOVSSET_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.49 SPMROOTCR_EL3, System Performance Monitors Root and Realm Control Register

The SPMROOTCR_EL3 characteristics are:

Purpose

Controls observability of Root and Realm events by System PMU <s>.

Configuration

This register is present only when FEAT_RME is implemented, FEAT_SPMU is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMROOTCR_EL3 are UNDEFINED.

Attributes

SPMROOTCR_EL3 is a 64-bit register.

Field descriptions

IMPLEMENTATION DEFINED, bits [63:32]

IMPLEMENTATION DEFINED observation controls. Additional IMPLEMENTATION DEFINED bits to control certain types of filter or events by System PMU <s>.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an IMPLEMENTATION DEFINED value.

Bit [31]

Reserved, RAO.

Indicates SPMROOTCR_EL3 is implemented by System PMU <s>.

Bits [30:4]

Reserved, RES0.

NAO, bit [3]

When System PMU <s> can count or monitor non-attributable events:

Non-attributable Observation. Controls whether events or monitorable characteristics not attributable with any source can be monitored by System PMU <s>.

NAO	Meaning
0b0	Events not attributable with any event source are not counted by System PMU <s>.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

0b1	Counting non-attributable events by System PMU <s> is not prevented by this mechanism.

When both SPMROOTCR_EL3 and SPMSCR_EL1 are implemented, non-attributable events are counted only if both SPMROOTCR_EL3.NAO is 1 and SPMSCR_EL1.{NAO, SO} is nonzero.

SPMROOTCR_EL3.NAO has the opposite reset polarity to SPMSCR_EL1.NAO.

The reset behavior of this field is:

- On a System PMU reset, this field resets to `'1'`.

Otherwise:

Reserved, RES0.

Bit [2]

Reserved, RES0.

RLO, bit [1]

Realm Observation. Controls whether events or monitorable characteristics attributable to a Realm event source can be monitored by System PMU <s>.

RLO	Meaning
0b0	Events attributable to a Realm event source are not counted by System PMU <s>.
0b1	Counting events by System PMU <s> that are attributable to a Realm event source is not prevented by this mechanism.

The reset behavior of this field is:

- On a System PMU reset, this field resets to `'0'`.

RTO, bit [0]

Root Observation. Controls whether events or monitorable characteristics attributable to a Root event source can be monitored by System PMU <s>.

RTO	Meaning
0b0	Events attributable to a Root event source are not counted by System PMU <s>.
0b1	Counting events by System PMU <s> that are attributable to a Root event source is not prevented by this mechanism.

The reset behavior of this field is:

- On a System PMU reset, this field resets to `'0'`.

Accessing SPMROOTCR_EL3

To access SPMROOTCR_EL3 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s.

SPMROOTCR_EL3 reads-as-zero and ignores writes if any of the following are true:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- System PMU <s> is not implemented.
- System PMU <s> does not implement SPMROOTCR_EL3.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMROOTCR_EL3
```

op0	op1	CRn	CRm	op2
0b10	0b110	0b1001	0b1110	0b111

```
if !(IsFeatureImplemented(FEAT_RME) && IsFeatureImplemented(FEAT_SPMU) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMROOTCR_EL3[UInt(SPMSELR_EL0.SYSPMUSEL)];
```

```
MSR SPMROOTCR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b110	0b1001	0b1110	0b111

```
if !(IsFeatureImplemented(FEAT_RME) && IsFeatureImplemented(FEAT_SPMU) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.SPMROOTCR_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMROOTCR_EL3[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.50 SPMSCR_EL1, System Performance Monitors Secure Control Register

The SPMSCR_EL1 characteristics are:

Purpose

Controls observability of Secure events by System PMU <s>, and optionally controls Secure attributes for message signaled interrupts and Non-secure access to the performance monitor registers.

Configuration

This register is present only when Secure EL1 is implemented, FEAT_SPMU is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMSCR_EL1 are UNDEFINED.

Attributes

SPMSCR_EL1 is a 64-bit register.

Field descriptions

IMPLEMENTATION DEFINED, bits [63:32]

IMPLEMENTATION DEFINED observation controls. Additional IMPLEMENTATION DEFINED bits to control certain types of filter or events by System PMU <s>.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an IMPLEMENTATION DEFINED value.

Bit [31]

Reserved, RAO.

Indicates SPMSCR_EL1 is implemented by System PMU <s>.

This field reads-as-one.

Bits [30:5]

Reserved, RES0.

NAO, bit [4]

When System PMU <s> can count or monitor non-attributable events:

Non-attributable Observation. Controls whether events or monitorable characteristics not attributable with any source can be monitored by System PMU <s>.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

NAO	Meaning
0b0	Events not attributable with any event source are not counted by System PMU <s>, unless overridden by SPMSCR_EL1.SO.
0b1	Counting non-attributable events by System PMU <s> is not prevented by this mechanism.

When both SPMROOTCR_EL3 and SPMSCR_EL1 are implemented, non-attributable events are counted only if both SPMROOTCR_EL3.NAO is 1 and SPMSCR_EL1.{NAO, SO} is nonzero.

SPMSCR_EL1.NAO has the opposite reset polarity to SPMROOTCR_EL3.NAO.

This field is optional if Root and Realm states are not implemented. When this field is not implemented, System PMU <s> behaves as if SPMSCR_EL1.NAO is 0, and whether events or monitorable characteristics not attributable with any source can be monitored is controlled by SPMSCR_EL1.SO.

The reset behavior of this field is:

- On a System PMU reset, this field resets to `'0'`.

Otherwise:

Reserved, RES0.

Bits [3:1]

Reserved, RES0.

SO, bit [0]

Secure Observation. Controls whether events or monitorable characteristics attributable to a Secure event source can be monitored by System PMU <s>.

SO	Meaning
0b0	Events attributable to a Secure event source are not counted by System PMU <s>.
0b1	Counting events by System PMU <s> that are attributable to a Secure event source is not prevented by this mechanism.

Also controls whether events or monitorable characteristics not attributable with any source can be monitored by System PMU <s>. See SPMSCR_EL1.NAO.

The reset behavior of this field is:

- On a System PMU reset, this field resets to `'0'`.

Accessing SPMSCR_EL1

To access SPMSCR_EL1 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s.

SPMSCR_EL1 reads-as-zero and ignores writes if any of the following are true:

- System PMU <s> is not implemented.
- System PMU <s> does not implement SPMSCR_EL1.

SPMSCR_EL1 is UNDEFINED if accessed in Non-secure or Realm state.

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, SPMSCR_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b111	0b1001	0b1110	0b111

```
if !(HaveELUsingSecurityState(EL1, TRUE) && IsFeatureImplemented(FEAT_SPMU) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif IsCurrentSecurityState(SS_NonSecure) || (IsFeatureImplemented(FEAT_RME) &&
     ↪IsCurrentSecurityState(SS_Realm)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGRTR2_EL2.nSPMSCR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMSCR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> ==
         ↪'00' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMSCR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMSCR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MSR SPMSCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b111	0b1001	0b1110	0b111

```
if !(HaveELUsingSecurityState(EL1, TRUE) && IsFeatureImplemented(FEAT_SPMU) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif IsCurrentSecurityState(SS_NonSecure) || (IsFeatureImplemented(FEAT_RME) &&
     ↪IsCurrentSecurityState(SS_Realm)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
         ↪'11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGWTR2_EL2.nSPMSCR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMSCR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
         ↪'11' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMSCR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPMSCR_EL1[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.51 SPMSELR_EL0, System Performance Monitors Select Register

The SPMSELR_EL0 characteristics are:

Purpose

Selects the System PMU and event counter registers to access.

Configuration

This register is present only when FEAT_SPMU is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMSELR_EL0 are UNDEFINED.

Attributes

SPMSELR_EL0 is a 64-bit register.

Field descriptions

Bits [63:10]

Reserved, RES0.

SYSPMUSEL, bits [9:4]

System PMU Select. Selects a System PMU <s> to access.

Values `0x20` to `0x3F` are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [3:2]

Reserved, RES0.

BANK, bits [1:0]

System PMU bank access control. Selects a bank of 16 System PMU event counters and related controls to access.

BANK	Meaning
`0b00`	Select event counters 0 to 15.
`0b01`	Select event counters 16 to 31.
`0b10`	Select event counters 32 to 47.
`0b11`	Select event counters 48 to 63.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing SPMSELR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, SPMSELR_EL0
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1001	0b1100	0b101

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
     ↪SCR_EL3.FGTEn2 == '0') || HDFGRTR2_EL2.nSPMSELR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMSELR_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
     ↪HDFGRTR2_EL2.nSPMSELR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMSELR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = SPMSELR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = SPMSELR_EL0;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MSR SPMSELR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b011	0b1001	0b1100	0b101

```
if !(IsFeatureImplemented(FEAT_SPMU) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
         ↪SCR_EL3.FGTEn2 == '0') || HDFGWTR2_EL2.nSPMSELR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMSELR_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGWTR2_EL2.nSPMSELR_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMSELR_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMSELR_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPMSELR_EL0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.5.52 SPMZR_EL0, System Performance Monitors Zero with Mask

The SPMZR_EL0 characteristics are:

Purpose

Zero the set of System PMU event counters specified by the mask written to SPMZR_EL0.

Configuration

This register is present only when FEAT_SPMU2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to SPMZR_EL0 are UNDEFINED.

Attributes

SPMZR_EL0 is a 64-bit register.

Field descriptions

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
P63	P62	P61	P60	P59	P58	P57	P56	P55	P54	P53	P52	P51	P50	P49	P48	P47	P46	P45	P44	P43	P42	P41	P40	P39	P38	P37	P36	P35	P34	P33	P32

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

P<m> , bits [m], for m = 63 to 0

Zero event counter <m>.

P<m>	Meaning
0b0	Write is ignored.
0b1	Set event counter<m> in System PMU <s> to zero.

The reset behavior of this field is:

- On a System PMU reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When event counter <m> is not implemented by System PMU <s>, access to this field is RAZ/WI.
- Otherwise, access to this field is WO/RAZ.

Accessing SPMZR_EL0

To access SPMZR_EL0 for System PMU <s>, set SPMSELR_EL0.SYSPMUSEL to s.

SPMZR_EL0 ignores writes if System PMU <s> is not implemented.

Accesses to this register use the following encodings in the System register encoding space:

```
MSR SPMZR_EL0, <Xt>
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b10	0b011	0b1001	0b1100	0b100

```
if !(IsFeatureImplemented(FEAT_SPMU2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
         ↪'11' then
        UNDEFINED;
    elsif MDSCR_EL1.EnSPM == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !ELIsInHost(EL0) && SPMACCESSR_EL1<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) &&
         ↪SCR_EL3.FGTEn2 == '0') || HDFGWTR2_EL2.nSPMEVCNTRn_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMZR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
         ↪'11' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGWTR2_EL2.nSPMEVCNTRn_EL0 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.EnSPM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SPMACCESSR_EL2<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        SPMZR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPM2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> !=
         ↪'11' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.EnPM2 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SPMACCESSR_EL3<UInt(SPMSELR_EL0.SYSPMUSEL) * 2+:2> != '11' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SPMZR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
elsif PSTATE.EL == EL3 then
    SPMZR_EL0[UInt(SPMSELR_EL0.SYSPMUSEL)] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.6 Activity Monitors registers

This section lists the Activity Monitors registers in AArch64.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.6.1 AMCFGR_EL0, Activity Monitors Configuration Register

The AMCFGR_EL0 characteristics are:

Purpose

Global configuration register for the activity monitors.

Provides information on supported features, the number of counter groups implemented, the total number of activity monitor event counters implemented, and the size of the counters. AMCFGR_EL0 is applicable to both the architected and the auxiliary counter groups.

Configuration

AArch64 System register AMCFGR_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMCFGR[31:0].

When FEAT_AMU_EXT32 is implemented, AArch64 System register AMCFGR_EL0 bits [31:0] are architecturally mapped to External register AMCFGR[31:0].

When FEAT_AMU_EXT64 is implemented, AArch64 System register AMCFGR_EL0 bits [63:0] are architecturally mapped to External register AMCFGR[63:0].

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCFGR_EL0 are UNDEFINED.

Attributes

AMCFGR_EL0 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

NCG, bits [31:28]

Defines the number of counter groups implemented, minus one.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NCG	Meaning
0b0000	One counter group implemented.
0b0001	Two counter groups implemented.

All other values are reserved.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bits [27:25]

Reserved, RES0.

HDBG, bit [24]

Halt-on-debug supported.

This feature must be supported, and so this bit is `0b1`.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HDBG	Meaning
`0b0`	AMCR_EL0.HDBG is RES0.
`0b1`	AMCR_EL0.HDBG is read/write.

Access to this field is RO.

Bits [23:14]

Reserved, RAZ.

SIZE, bits [13:8]

Defines the size of the activity monitor event counters, minus one.

The counters are 64-bit, so the value of this field is `0b111111`.

This field is used by software to determine the spacing of the counters in the memory-map. The counters are at doubleword-aligned addresses.

Reads as `0b111111`

Access to this field is RO.

N, bits [7:0]

Defines the number of activity monitor event counters implemented in all groups, minus one.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Accessing AMCFGR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMCFGR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCFGR_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCFGR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCFGR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMCFGR_EL0;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.6.2 AMCG1IDR_EL0, Activity Monitors Counter Group 1 Identification Register

The AMCG1IDR_EL0 characteristics are:

Purpose

Defines which auxiliary counters are implemented, and which of them have a corresponding virtual offset register, AMEVCNTVOFF1<n>_EL2 implemented.

Configuration

This register is present only when FEAT_AMUv1p1 is implemented. Otherwise, direct accesses to AMCG1IDR_EL0 are UNDEFINED.

Attributes

AMCG1IDR_EL0 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

AMEVCNTOFF1<n>_EL2 , bits [n+16], for n = 15 to 0

Indicates which implemented auxiliary counters have a corresponding virtual offset register, AMEVCNTVOFF1<n>_EL2 implemented.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

AMEVCNTOFF1<n>_EL2	Meaning
0b0	AMEVCNTR1<n>_EL0 does not have an offset, or is not implemented.
0b1	The offset AMEVCNTVOFF1<n>_EL2 is implemented for AMEVCNTR1<n>_EL0.

AMEVCNTR1<n>_EL0 , bits [n], for n = 15 to 0

Indicates which auxiliary counters AMEVCNTR1<n>_EL0 are implemented.

AMEVCNTR1<n>_EL0	Meaning
0b0	AMEVCNTR1<n>_EL0 is not implemented.
0b1	AMEVCNTR1<n>_EL0 is implemented.

Accessing AMCG1IDR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMCG1IDR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0010	0b110

```
if !IsFeatureImplemented(FEAT_AMUv1p1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCG1IDR_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCG1IDR_EL0;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCG1IDR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMCG1IDR_EL0;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.6.3 AMCGCR_EL0, Activity Monitors Counter Group Configuration Register

The AMCGCR_EL0 characteristics are:

Purpose

Provides information on the number of activity monitor event counters implemented within each counter group.

Configuration

AArch64 System register AMCGCR_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMCGCR[31:0].

When FEAT_AMU_EXT32 is implemented, AArch64 System register AMCGCR_EL0 bits [31:0] are architecturally mapped to External register AMCGCR[31:0].

When FEAT_AMU_EXT64 is implemented, AArch64 System register AMCGCR_EL0 bits [63:0] are architecturally mapped to External register AMCGCR[63:0].

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCGCR_EL0 are UNDEFINED.

Attributes

AMCGCR_EL0 is a 64-bit register.

Field descriptions

Bits [63:16]

Reserved, RES0.

CG1NC, bits [15:8]

Counter Group 1 Number of Counters. The number of counters in the auxiliary counter group.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CG1NC	Meaning
0x00..0x10	The number of counters.

All other values are reserved.

Access to this field is RO.

CG0NC, bits [7:0]

Counter Group 0 Number of Counters. The number of counters in the architected counter group.

Reads as 0x04

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing AMCGCR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMCGCR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0010	0b010

```
if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCGCR_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCGCR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCGCR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMCGCR_EL0;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.6.4 AMCNTENCLR0_EL0, Activity Monitors Count Enable Clear Register 0

The AMCNTENCLR0_EL0 characteristics are:

Purpose

Disable control bits for the architected activity monitors event counters, AMEVCNTR0<n>_EL0.

Configuration

AArch64 System register AMCNTENCLR0_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMCNTENCLR0[31:0].

AArch64 System register AMCNTENCLR0_EL0 bits [31:0] are architecturally mapped to External register AMCNTENCLR0[31:0].

AArch64 System register AMCNTENCLR0_EL0 bits [31:0] are architecturally mapped to External register AMCNTENCLR[31:0].

AArch64 System register AMCNTENCLR0_EL0 bits [31:0] are architecturally mapped to External register AMCNTENSET0[31:0].

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCNTENCLR0_EL0 are UNDEFINED.

Attributes

AMCNTENCLR0_EL0 is a 64-bit register.

Field descriptions

Bits [63:16]

Reserved, RES0.

Bits [15:4]

Reserved, RAZ/WI.

This field is reserved for additional architected activity monitor event counters, which Arm might define in a future version of the Activity Monitors architecture.

P<n> , bits [n], for n = 3 to 0

Activity monitor event counter disable bit for AMEVCNTR0<n>_EL0.

Note

AMCGCR_EL0.CG0NC identifies the number of architected activity monitor event counters. In an implementation that includes FEAT_AMUv1, the number of architected activity monitor event counters is 4.

Possible values of each bit are:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

P<n>	Meaning
0b0	When read, means that AMEVCNTR0<n>_EL0 is disabled. When written, has no effect.
0b1	When read, means that AMEVCNTR0<n>_EL0 is enabled. When written, disables AMEVCNTR0<n>_EL0.

The reset behavior of this field is:

- On a AMU reset, this field resets to '0'.

Accessing this field has the following behavior:

- When n >= 4, access to this field is RAZ/WI.
- Otherwise, access to this field is W1C.

Accessing AMCNTENCLR0_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMCNTENCLR0_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0010	0b100

```
if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMCNTEN0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCNTENCLR0_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HAFGRTR_EL2.AMCNTEN0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    else
        X[t, 64] = AMCNTENCLR0_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCNTENCLR0_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMCNTENCLR0_EL0;
```

```
MSR AMCNTENCLR0_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0010	0b100

```
if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif IsHighestEL(PSTATE.EL) then
    AMCNTENCLR0_EL0 = X[t, 64];
else
    UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.6.5 AMCNTENCLR1_EL0, Activity Monitors Count Enable Clear Register 1

The AMCNTENCLR1_EL0 characteristics are:

Purpose

Disable control bits for the auxiliary activity monitors event counters, AMEVCNTR1<n>_EL0.

Configuration

AArch64 System register AMCNTENCLR1_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMCNTENCLR1[31:0].

AArch64 System register AMCNTENCLR1_EL0 bits [31:0] are architecturally mapped to External register AMCNTENCLR1[31:0].

AArch64 System register AMCNTENCLR1_EL0 bits [31:0] are architecturally mapped to External register AMCNTENSET1[31:0].

AArch64 System register AMCNTENCLR1_EL0 bits [31:0] are architecturally mapped to External register AMCNTENCLR[63:32].

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCNTENCLR1_EL0 are UNDEFINED.

Attributes

AMCNTENCLR1_EL0 is a 64-bit register.

Field descriptions

Bits [63:16]

Reserved, RES0.

P<n> , bits [n], for n = 15 to 0

Activity monitor event counter disable bit for AMEVCNTR1<n>_EL0.

Possible values of each bit are:

P<n>	Meaning
0b0	When read, means that AMEVCNTR1<n>_EL0 is disabled. When written, has no effect.
0b1	When read, means that AMEVCNTR1<n>_EL0 is enabled. When written, disables AMEVCNTR1<n>_EL0.

The reset behavior of this field is:

- On a AMU reset, this field resets to '0'.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing this field has the following behavior:

- When n >= UInt(AMCGCR_EL0.CG1NC), access to this field is RAZ/WI.
- Otherwise, access to this field is W1C.

Accessing AMCNTENCLR1_EL0

If there are no auxiliary monitor event counters implemented, reads and writes of AMCNTENCLR1_EL0 are UNDEFINED.

Note

There are no implemented auxiliary activity monitor event counters when AMCFGR_EL0.NCG == `0b0000`.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMCNTENCLR1_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMCNTEN1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCNTENCLR1_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HAFGRTR_EL2.AMCNTEN1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCNTENCLR1_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCNTENCLR1_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMCNTENCLR1_EL0;
```

```
MSR AMCNTENCLR1_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif IsHighestEL(PSTATE.EL) then
    AMCNTENCLR1_EL0 = X[t, 64];
else
    UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.6.6 AMCNTENSET0_EL0, Activity Monitors Count Enable Set Register 0

The AMCNTENSET0_EL0 characteristics are:

Purpose

Enable control bits for the architected activity monitors event counters, AMEVCNTR0<n>_EL0.

Configuration

AArch64 System register AMCNTENSET0_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMCNTENSET0[31:0].

AArch64 System register AMCNTENSET0_EL0 bits [31:0] are architecturally mapped to External register AMCNTENSET0[31:0].

AArch64 System register AMCNTENSET0_EL0 bits [31:0] are architecturally mapped to External register AMCNTENSET[31:0].

AArch64 System register AMCNTENSET0_EL0 bits [31:0] are architecturally mapped to External register AMCNTENCLR0[31:0].

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCNTENSET0_EL0 are UNDEFINED.

Attributes

AMCNTENSET0_EL0 is a 64-bit register.

Field descriptions

Bits [63:16]

Reserved, RES0.

Bits [15:4]

Reserved, RAZ/WI.

This field is reserved for additional architected activity monitor event counters, which Arm might define in a future version of the Activity Monitors architecture.

P<n> , bits [n], for n = 3 to 0

Activity monitor event counter enable bit for AMEVCNTR0<n>_EL0.

Note

AMCGCR_EL0.CG0NC identifies the number of architected activity monitor event counters. In an implementation that includes FEAT_AMUv1, the number of architected activity monitor event counters is 4.

Possible values of each bit are:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

P<n>	Meaning
0b0	When read, means that AMEVCNTR0<n>_EL0 is disabled. When written, has no effect.
0b1	When read, means that AMEVCNTR0<n>_EL0 is enabled. When written, enables AMEVCNTR0<n>_EL0.

The reset behavior of this field is:

- On a AMU reset, this field resets to '0'.

Accessing this field has the following behavior:

- When n >= 4, access to this field is RAZ/WI.
- Otherwise, access to this field is W1S.

Accessing AMCNTENSET0_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMCNTENSET0_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0010	0b101

```
if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMCNTEN0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCNTENSET0_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HAFGRTR_EL2.AMCNTEN0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    else
        X[t, 64] = AMCNTENSET0_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCNTENSET0_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMCNTENSET0_EL0;
```

```
MSR AMCNTENSET0_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0010	0b101

```
if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif IsHighestEL(PSTATE.EL) then
    AMCNTENSET0_EL0 = X[t, 64];
else
    UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.6.7 AMCNTENSET1_EL0, Activity Monitors Count Enable Set Register 1

The AMCNTENSET1_EL0 characteristics are:

Purpose

Enable control bits for the auxiliary activity monitors event counters, AMEVCNTR1<n>_EL0.

Configuration

AArch64 System register AMCNTENSET1_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMCNTENSET1[31:0].

AArch64 System register AMCNTENSET1_EL0 bits [31:0] are architecturally mapped to External register AMCNTENSET1[31:0].

AArch64 System register AMCNTENSET1_EL0 bits [31:0] are architecturally mapped to External register AMCNTENCLR1[31:0].

AArch64 System register AMCNTENSET1_EL0 bits [31:0] are architecturally mapped to External register AMCNTENSET[63:32].

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCNTENSET1_EL0 are UNDEFINED.

Attributes

AMCNTENSET1_EL0 is a 64-bit register.

Field descriptions

Bits [63:16]

Reserved, RES0.

P<n> , bits [n], for n = 15 to 0

Activity monitor event counter enable bit for AMEVCNTR1<n>_EL0.

Possible values of each bit are:

P<n>	Meaning
0b0	When read, means that AMEVCNTR1<n>_EL0 is disabled. When written, has no effect.
0b1	When read, means that AMEVCNTR1<n>_EL0 is enabled. When written, enables AMEVCNTR1<n>_EL0.

The reset behavior of this field is:

- On a AMU reset, this field resets to '0'.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing this field has the following behavior:

- When n >= UInt(AMCGCR_EL0.CG1NC), access to this field is RAZ/WI.
- Otherwise, access to this field is W1S.

Accessing AMCNTENSET1_EL0

If there are no auxiliary monitor event counters implemented, reads and writes of AMCNTENSET1_EL0 are UNDEFINED.

Note

There are no implemented auxiliary activity monitor event counters when AMCFGR_EL0.NCG == `0b0000`.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMCNTENSET1_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMCNTEN1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCNTENSET1_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HAFGRTR_EL2.AMCNTEN1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCNTENSET1_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCNTENSET1_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMCNTENSET1_EL0;
```

```
MSR AMCNTENSET1_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif IsHighestEL(PSTATE.EL) then
    AMCNTENSET1_EL0 = X[t, 64];
else
    UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.6.8 AMCR_EL0, Activity Monitors Control Register

The AMCR_EL0 characteristics are:

Purpose

Global control register for the activity monitors implementation. AMCR_EL0 is applicable to both the architected and the auxiliary counter groups.

Configuration

AArch64 System register AMCR_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMCR[31:0].

When FEAT_AMU_EXT32 is implemented, AArch64 System register AMCR_EL0 bits [31:0] are architecturally mapped to External register AMCR[31:0].

When FEAT_AMU_EXT64 is implemented, AArch64 System register AMCR_EL0 bits [63:0] are architecturally mapped to External register AMCR[63:0].

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCR_EL0 are UNDEFINED.

Attributes

AMCR_EL0 is a 64-bit register.

Field descriptions

Bits [63:18]

Reserved, RES0.

CG1RZ, bit [17]

When FEAT_AMUv1p1 is implemented:

Counter Group 1 Read Zero.

CG1RZ	Meaning
0b0	System register reads of AMEVCNTR1<n>_EL0 return the event count at all implemented and enabled Exception levels.
0b1	If the current Exception level is the highest implemented Exception level, system register reads of AMEVCNTR1<n>_EL0 return the event count. Otherwise, reads of AMEVCNTR1<n>_EL0 return a zero value.

Note

Reads from the memory-mapped view are unaffected by this field.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [16:11]

Reserved, RES0.

HDBG, bit [10]

This bit controls whether activity monitor counting is halted when the PE is halted in Debug state.

HDBG	Meaning
0b0	Activity monitors do not halt counting when the PE is halted in Debug state.
0b1	Activity monitors halt counting when the PE is halted in Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [9:0]

Reserved, RES0.

Accessing AMCR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMCR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCR_EL0;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMCR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMCR_EL0;
```

MSR AMCR_EL0, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif IsHighestEL(PSTATE.EL) then
    AMCR_EL0 = X[t, 64];
else
    UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.6.9 AMEVCNTR0<n>_EL0, Activity Monitors Event Counter Registers 0, n = 0 - 3

The AMEVCNTR0<n>_EL0 characteristics are:

Purpose

Provides access to the architected activity monitor event counters.

Configuration

AArch64 System register AMEVCNTR0<n>_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMEVCNTR0<n>[31:0].

AArch64 System register AMEVCNTR0<n>_EL0 bits [63:0] are architecturally mapped to External register AMEVCNTR0[63:0].

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMEVCNTR0<n>_EL0 are UNDEFINED.

Attributes

AMEVCNTR0<n>_EL0 is a 64-bit register.

Field descriptions

ACNT, bits [63:0]

Architected activity monitor event counter n.

Value of architected activity monitor event counter n, where n is the number of this register and is a number from 0 to 3.

If all of the following are true, reads of the AMEVCNTR0<n>_EL0 registers from EL0 or EL1 return (PCount<63:0> - AMEVCNTVOFF0<n>_EL2<63:0>), where PCount is the physical count returned when AMEVCNTR0<n>_EL0 is read from EL2 or EL3:

- FEAT_AMUv1p1 is implemented.
- HCR_EL2.AMVOFFEN and SCR_EL3.AMVOFFEN are 1.
- EL2 is implemented and enabled in the current Security state, and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.

If the counter is enabled, writes to this register have UNPREDICTABLE results.

The reset behavior of this field is:

- On a AMU reset, this field resets to `0x0000000000000000`.

Accessing AMEVCNTR0<n>_EL0

If <n> is greater than or equal to the number of architected activity monitor event counters, reads and writes of AMEVCNTR0<n>_EL0 are UNDEFINED.

Note

AMCGCR_EL0.CG0NC identifies the number of architected activity monitor event counters.

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, AMEVCNTR0<m>_EL0 ; Where m = 0-3
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b010:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif m >= 4 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMEVCNTR0<m>_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMEVCNTR0_EL0[m];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HAFGRTR_EL2.AMEVCNTR0<m>_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMEVCNTR0_EL0[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMEVCNTR0_EL0[m];
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMEVCNTR0_EL0[m];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MSR AMEVCNTR0<m>_EL0, <Xt> ; Where m = 0-3
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b010:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif m >= 4 then
    UNDEFINED;
elsif IsHighestEL(PSTATE.EL) then
    AMEVCNTR0_EL0[m] = X[t, 64];
else
    UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.6.10 AMEVCNTR1<n>_EL0, Activity Monitors Event Counter Registers 1, n = 0 - 15

The AMEVCNTR1<n>_EL0 characteristics are:

Purpose

Provides access to the auxiliary activity monitor event counters.

Configuration

AArch64 System register AMEVCNTR1<n>_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMEVCNTR1<n>[31:0].

AArch64 System register AMEVCNTR1<n>_EL0 bits [63:0] are architecturally mapped to External register AMEVCNTR1[63:0].

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMEVCNTR1<n>_EL0 are UNDEFINED.

Attributes

AMEVCNTR1<n>_EL0 is a 64-bit register.

Field descriptions

ACNT, bits [63:0]

Auxiliary activity monitor event counter n.

Value of auxiliary activity monitor event counter n, where n is the number of this register and is a number from 0 to 15.

If all of the following are true, reads of the AMEVCNTR1<n>_EL0 registers from EL0 or EL1 return (PCount<63:0> - AMEVCNTVOFF1<n>_EL2<63:0>), where PCount is the physical count returned when AMEVCNTR1<n>_EL0 is read from EL2 or EL3:

- FEAT_AMUv1p1 is implemented.
- HCR_EL2.AMVOFFEN and SCR_EL3.AMVOFFEN are 1.
- AMCR_EL0.CG1RZ is 0.
- EL2 is implemented and enabled in the current Security state, and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.

If the counter is enabled, writes to this register have UNPREDICTABLE results.

The reset behavior of this field is:

- On a AMU reset, this field resets to `0x0000000000000000`.

Accessing AMEVCNTR1<n>_EL0

If <n> is greater than or equal to the number of auxiliary activity monitor event counters, reads and writes of AMEVCNTR1<n>_EL0 are UNDEFINED.

Note

AMCGCR_EL0.CG1NC identifies the number of auxiliary activity monitor event counters.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMEVCNTR1<m>_EL0 ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b110:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif m >= NUM_AMU_CG1_MONITORS then
    UNDEFINED;
elsif !IsG1ActivityMonitorImplemented(m) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMEVCNTR1<m>_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif AMCR_EL0.CG1RZ == '1' then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = AMEVCNTR1_EL0[m];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HAFGRTR_EL2.AMEVCNTR1<m>_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsHighestEL(PSTATE.EL) && AMCR_EL0.CG1RZ == '1' then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = AMEVCNTR1_EL0[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsHighestEL(PSTATE.EL) && AMCR_EL0.CG1RZ == '1' then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = AMEVCNTR1_EL0[m];
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMEVCNTR1_EL0[m];
```

```
MSR AMEVCNTR1<m>_EL0, <Xt> ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b110:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif m >= NUM_AMU_CG1_MONITORS then
    UNDEFINED;
elsif !IsG1ActivityMonitorImplemented(m) then
    UNDEFINED;
elsif IsHighestEL(PSTATE.EL) then
    AMEVCNTR1_EL0[m] = X[t, 64];
else
    UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.6.11 AMEVCNTVOFF0<n>_EL2, Activity Monitors Event Counter Virtual Offset Registers 0, n = 0 - 15

The AMEVCNTVOFF0<n>_EL2 characteristics are:

Purpose

Holds the 64-bit virtual offset for architected activity monitor events.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_AMUv1p1 is implemented. Otherwise, direct accesses to AMEVCNTVOFF0<n>_EL2 are UNDEFINED.

Attributes

AMEVCNTVOFF0<n>_EL2 is a 64-bit register.

Field descriptions

VOffset, bits [63:0]

Virtual offset.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AMEVCNTVOFF0<n>_EL2

If <n> is not 0, 2 or 3, reads and writes of AMEVCNTVOFF0<n>_EL2 are UNDEFINED.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMEVCNTVOFF0<m>_EL2 ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1101	0b100:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !IsFeatureImplemented(FEAT_AMUv1p1) then
    UNDEFINED;
elsif m >= 4 then
    UNDEFINED;
elsif !(m IN {0, 2, 3}) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0xA00 + (8 * m)];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AMVOFFEN == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.AMVOFFEN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMEVCNTVOFF0_EL2[m];
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMEVCNTVOFF0_EL2[m];
```

```
MSR AMEVCNTVOFF0<m>_EL2, <Xt> ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1101	0b100:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !IsFeatureImplemented(FEAT_AMUv1p1) then
    UNDEFINED;
elsif m >= 4 then
    UNDEFINED;
elsif !(m IN {0, 2, 3}) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0xA00 + (8 * m)] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AMVOFFEN == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.AMVOFFEN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        AMEVCNTVOFF0_EL2[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    AMEVCNTVOFF0_EL2[m] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.6.12 AMEVCNTVOFF1<n>_EL2, Activity Monitors Event Counter Virtual Offset Registers 1, n = 0 - 15

The AMEVCNTVOFF1<n>_EL2 characteristics are:

Purpose

Holds the 64-bit virtual offset for auxiliary activity monitor events.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_AMUv1p1 is implemented. Otherwise, direct accesses to AMEVCNTVOFF1<n>_EL2 are UNDEFINED.

Attributes

AMEVCNTVOFF1<n>_EL2 is a 64-bit register.

Field descriptions

VOffset, bits [63:0]

Virtual offset.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AMEVCNTVOFF1<n>_EL2

Note

AMCG1IDR_EL0 identifies which auxiliary activity monitor event counters have a corresponding virtual offset implemented.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMEVCNTVOFF1<m>_EL2 ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1101	0b101:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !IsFeatureImplemented(FEAT_AMUv1p1) then
    UNDEFINED;
elsif m >= NUM_AMU_CG1_MONITORS then
    UNDEFINED;
elsif !IsG1ActivityMonitorOffsetImplemented(m) then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0xA80 + (8 * m)];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AMVOFFEN == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.AMVOFFEN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMEVCNTVOFF1_EL2[m];
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMEVCNTVOFF1_EL2[m];
```

```
MSR AMEVCNTVOFF1<m>_EL2, <Xt> ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1101	0b101:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !IsFeatureImplemented(FEAT_AMUv1p1) then
    UNDEFINED;
elsif m >= NUM_AMU_CG1_MONITORS then
    UNDEFINED;
elsif !IsG1ActivityMonitorOffsetImplemented(m) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0xA80 + (8 * m)] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.AMVOFFEN == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.AMVOFFEN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        AMEVCNTVOFF1_EL2[m] = X[t, 64];
elsif PSTATE.EL == EL3 then
    AMEVCNTVOFF1_EL2[m] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.6.13 AMEVTYPER0<n>_EL0, Activity Monitors Event Type Registers 0, n = 0 - 3

The AMEVTYPER0<n>_EL0 characteristics are:

Purpose

Provides information on the events that an architected activity monitor event counter AMEVCNTR0<n>_EL0 counts.

Configuration

AArch64 System register AMEVTYPER0<n>_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMEVTYPER0<n>[31:0].

AArch64 System register AMEVTYPER0<n>_EL0 bits [31:0] are architecturally mapped to External register AMEVTYPER0[31:0].

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMEVTYPER0<n>_EL0 are UNDEFINED.

Attributes

AMEVTYPER0<n>_EL0 is a 64-bit register.

Field descriptions

Bits [63:16]

Reserved, RES0.

evtCount, bits [15:0]

Event to count. The event number of the event that is counted by the architected activity monitor event counter AMEVCNTR0<n>_EL0. The value of this field is architecturally mandated for each architected counter.

The following table shows the mapping between required event numbers and the corresponding counters:

evtCount	Meaning	Applies when
0x0011	Processor frequency cycles.	n == 0
0x4004	Constant frequency cycles.	n == 1
0x0008	Instructions retired.	n == 2
0x4005	Memory stall cycles.	n == 3

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing AMEVTYPER0<n>_EL0

If <n> is greater than or equal to the number of architected activity monitor event counters, reads and writes of AMEVTYPER0<n>_EL0 are UNDEFINED.

Note

AMCGCR_EL0.CG0NC identifies the number of architected activity monitor event counters.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMEVTYPER0<m>_EL0 ; Where m = 0-3
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b011:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif m >= 4 then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMEVTYPER0_EL0[m];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMEVTYPER0_EL0[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMEVTYPER0_EL0[m];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMEVTYPER0_EL0[m];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.6.14 AMEVTYPER1<n>_EL0, Activity Monitors Event Type Registers 1, n = 0 - 15

The AMEVTYPER1<n>_EL0 characteristics are:

Purpose

Provides information on the events that an auxiliary activity monitor event counter AMEVCNTR1<n>_EL0 counts.

Configuration

AArch64 System register AMEVTYPER1<n>_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMEVTYPER1<n>[31:0].

AArch64 System register AMEVTYPER1<n>_EL0 bits [31:0] are architecturally mapped to External register AMEVTYPER1[31:0].

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMEVTYPER1<n>_EL0 are UNDEFINED.

Attributes

AMEVTYPER1<n>_EL0 is a 64-bit register.

Field descriptions

Bits [63:16]

Reserved, RES0.

evtCount, bits [15:0]

Event to count. The event number of the event that is counted by the auxiliary activity monitor event counter AMEVCNTR1<n>_EL0.

It is IMPLEMENTATION DEFINED what values are supported by each counter.

If software writes a value to this field which is not supported by the corresponding counter AMEVCNTR1<n>_EL0, then:

- It is UNPREDICTABLE which event will be counted.
- The value read back is UNKNOWN.

The event counted by AMEVCNTR1<n>_EL0 might be fixed at implementation. In this case, the field is read-only and writes are UNDEFINED.

If the corresponding counter AMEVCNTR1<n>_EL0 is enabled, writes to this register have UNPREDICTABLE results.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AMEVTYPER1<n>_EL0

If <n> is greater than or equal to the number of auxiliary activity monitor event counters, reads and writes of AMEVTYPER1<n>_EL0 are UNDEFINED.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Note

AMCGCR_EL0.CG1NC identifies the number of auxiliary activity monitor event counters.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMEVTYPER1<m>_EL0 ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b111:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif m >= NUM_AMU_CG1_MONITORS then
    UNDEFINED;
elsif !IsG1ActivityMonitorImplemented(m) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
     ↪SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMEVTYPER1<m>_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMEVTYPER1_EL0[m];
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
     ↪HAFGRTR_EL2.AMEVTYPER1<m>_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMEVTYPER1_EL0[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMEVTYPER1_EL0[m];
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMEVTYPER1_EL0[m];
```

```
MSR AMEVTYPER1<m>_EL0, <Xt> ; Where m = 0-15
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b111:m[3]	m[2:0]

```
integer m = UInt(CRm<0>:op2<2:0>);

if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif m >= NUM_AMU_CG1_MONITORS then
    UNDEFINED;
elsif !IsG1ActivityMonitorImplemented(m) then
    UNDEFINED;
elsif IsHighestEL(PSTATE.EL) && !boolean IMPLEMENTATION_DEFINED "AMEVCNTR1_EL0[m] is fixed" then
    AMEVTYPER1_EL0[m] = X[t, 64];
else
    UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.6.15 AMUSERENR_EL0, Activity Monitors User Enable Register

The AMUSERENR_EL0 characteristics are:

Purpose

Global user enable register for the activity monitors. Enables or disables EL0 access to the activity monitors. AMUSERENR_EL0 is applicable to both the architected and the auxiliary counter groups.

Configuration

AArch64 System register AMUSERENR_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMUSERENR[31:0].

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMUSERENR_EL0 are UNDEFINED.

Attributes

AMUSERENR_EL0 is a 64-bit register.

Field descriptions

Bits [63:1]

Reserved, RES0.

EN, bit [0]

Traps EL0 accesses to the activity monitors registers to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, as follows:

- In AArch64 state, accesses to the following registers are trapped, reported using EC syndrome value `0x18`:
  - AMCFGR_EL0, AMCGCR_EL0, AMCNTENCLR0_EL0, AMCNTENCLR1_EL0, AMCNTENSET0_EL0, AMCNTENSET1_EL0, AMCR_EL0, AMEVCNTR0<n>_EL0, AMEVCNTR1<n>_EL0, AMEVTYPER0<n>_EL0, and AMEVTYPER1<n>_EL0.
- In AArch32 state, MRC and MCR accesses to the following registers are trapped and reported using EC syndrome value `0x03`, MRRC and MCRR accesses are trapped and reported using EC syndrome value `0x04`:
  - AMCFGR, AMCGCR, AMCNTENCLR0, AMCNTENCLR1, AMCNTENSET0, AMCNTENSET1, AMCR, AMEVCNTR0<n>, AMEVCNTR1<n>, AMEVTYPER0<n>, and AMEVTYPER1<n>.

EN	Meaning
`0b0`	EL0 accesses to the activity monitors registers are trapped.
`0b1`	This control does not cause any instructions to be trapped. Software can access all activity monitor registers at EL0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Note

- AMUSERENR_EL0 can always be read at EL0 and is not governed by this bit.

Note

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AMUSERENR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, AMUSERENR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0010	0b011

```
if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMUSERENR_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMUSERENR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = AMUSERENR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = AMUSERENR_EL0;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MSR AMUSERENR_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1101	0b0010	0b011

```
if !IsFeatureImplemented(FEAT_AMUv1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        AMUSERENR_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        AMUSERENR_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    AMUSERENR_EL0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.7 Statistical Profiling Extension registers

This section lists the Statistical Profiling Extension registers in AArch64.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.7.1 PMBIDR_EL1, Profiling Buffer ID Register

The PMBIDR_EL1 characteristics are:

Purpose

Provides information to software as to whether the buffer can be programmed at the current Exception level.

Configuration

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMBIDR_EL1 are UNDEFINED.

Attributes

PMBIDR_EL1 is a 64-bit register.

Field descriptions

Bits [63:48]

Reserved, RES0.

MaxBuffSize, bits [47:32]

Maximum supported Profiling Buffer size. Reserved for software use.

Reads as `0x0000`

The only permitted value is `0x0000`, indicating there is no limit to the maximum buffer size.

Note

Permitted values relate to the values an implementation is permitted to set this field to. A hypervisor might trap accesses to this register and use other values to describe limitations of its virtualization support to a guest operating system, as follows:

- MaxBuffSize bits[8:0] encodes a mantissa value, `M`.
- MaxBuffSize bits[13:9] encodes an exponent value, `E`.
- MaxBuffSize bits[15:14] are reserved.

The maximum buffer size, in bytes, is expressed using the following function:

```
if IsZero(E), then UInt(M:Zeros(12)) else UInt('1':M:Zeros(UInt(E)+11))
```

For example:

- A value of `0x0001` means a maximum buffer size of 4KB.
- A value of `0x3FFF` means a maximum buffer size of 4092TB.

Access to this field is RO.

Bits [31:12]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EA, bits [11:8]

External Abort handling. Describes how the PE manages External aborts on writes made by the Statistical Profiling Unit to the Profiling Buffer.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EA	Meaning
0b0000	Not described.
0b0001	The PE ignores External aborts on writes made by the Statistical Profiling Unit.
0b0010	An External abort on a write made by the Statistical Profiling Unit generates an asynchronous SError exception at the PE.

All other values are reserved.

From Armv8.8, the value 0b0000 is not permitted.

The behavior described by this field does not apply for External aborts on a translation table walk, translation table update, or GPT walk made by the Statistical Profiling Unit that are reported as MMU faults using PMBSR_ELx. For more information, see 'External aborts'.

Access to this field is RO.

AddrMode, bits [7:6]

When FEAT_SPE_nVM is implemented:

Address Modes. Describes the addressing modes available for the Profiling Buffer.

AddrMode	Meaning
0b00	Only virtual address mode is supported.
0b01	Virtual and physical address modes are supported.
0b11	Reserved for software use under virtualization, to show that only physical address mode is supported.

Other values are reserved.

If the Effective value of PMSCR_EL2.EnVM is 1 and the value returned for PMBIDR_EL1.P is 0, then this field reads as 0b01. Otherwise, this field reads as 0b00.

Note

A hypervisor might trap accesses to this register to describe limitations of its virtualization support to a guest operating system.

Otherwise:

Reserved, RES0.

F, bit [5]

Flag updates. Describes how address translations performed by the Statistical Profiling Unit manage the Access flag and dirty state.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F	Meaning
0b0	Hardware management of the Access flag and dirty state for accesses made by the Statistical Profiling Unit is always disabled for all translation stages.
0b1	Hardware management of the Access flag and dirty state for accesses made by the Statistical Profiling Unit is controlled in the same way as explicit memory accesses in the Profiling Buffer owning translation regime.

Note

If hardware management of the Access flag is disabled for a stage of translation, an access to a Page or Block with the Access flag bit not set in the descriptor will generate an Access Flag fault.
If hardware management of the dirty state is disabled for a stage of translation, an access to a Page or Block will ignore the Dirty Bit Modifier in the descriptor and might generate a Permission fault, depending on the values of the access permission bits in the descriptor.

From Armv8.8, the value 0 is not permitted.

Access to this field is RO.

P, bit [4]

Programming not allowed. When read at EL3, this field reads as zero. Otherwise, indicates that the Profiling Buffer owning Exception level is a higher Exception level or the Profiling Buffer owning Security state is not the current Security state.

P	Meaning
0b0	Programming is allowed.
0b1	Programming not allowed.

The value read from this field depends on the current Exception level and the Effective values of MDCR_EL3.NSPB, MDCR_EL3.NSPBE, and MDCR_EL2.E2PB:

- If EL3 is implemented, MDCR_EL3.NSPB is 0b0x, and either FEAT_RME is not implemented, or Secure state is implemented and MDCR_EL3.NSPBE is 0, then this field reads as one from:
  - Non-secure EL1 and Non-secure EL2.
  - If FEAT_RME is implemented, Realm EL1 and Realm EL2.
  - If Secure EL2 is implemented and enabled, and MDCR_EL2.E2PB is 0b00, Secure EL1.
- If EL3 is implemented, MDCR_EL3.NSPB is 0b1x and either FEAT_RME is not implemented or MDCR_EL3.NSPBE is 0, then this field reads as one from:
  - If Secure state is implemented, Secure EL1.
  - If Secure EL2 is implemented, Secure EL2.
  - If EL2 is implemented and MDCR_EL2.E2PB is 0b00, Non-secure EL1.
  - If FEAT_RME is implemented, Realm EL1 and Realm EL2.
- If FEAT_RME is implemented, and MDCR_EL3.{NSPB, NSPBE} is {0b1x, 1}, then this field reads as one from:
  - Non-secure EL1 and Non-secure EL2.
  - If Secure state is implemented, Secure EL1 and Secure EL2.
  - If MDCR_EL2.E2PB is 0b00, Realm EL1.
- If EL3 is not implemented, EL2 is implemented, and MDCR_EL2.E2PB is 0b00, then this field reads as one from EL1.

Otherwise, this field reads as zero.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Align, bits [3:0]

Defines the minimum alignment constraint for writes to PMBPTR_EL1.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Align	Meaning
0b0000	Byte.
0b0001	Halfword.
0b0010	Word.
0b0011	Doubleword.
0b0100	16 bytes.
0b0101	32 bytes.
0b0110	64 bytes.
0b0111	128 bytes.
0b1000	256 bytes.
0b1001	512 bytes.
0b1010	1KB.
0b1011	2KB.

All other values are reserved.

For more information, see Restrictions on the current write pointer.

If this field is nonzero, then every record is a multiple of this size.

Access to this field is RO.

Accessing PMBIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMBIDR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1010	0b111

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        ↪HDFGRTR_EL2.PMBIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = PMBIDR_EL1;
elsif PSTATE.EL == EL2 then
    X[t, 64] = PMBIDR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMBIDR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.7.2 PMBLIMITR_EL1, Profiling Buffer Limit Address Register

The PMBLIMITR_EL1 characteristics are:

Purpose

Defines the upper limit for the profiling buffer, and enables the profiling buffer

Configuration

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMBLIMITR_EL1 are UNDEFINED.

Attributes

PMBLIMITR_EL1 is a 64-bit register.

Field descriptions

LIMIT, bits [63:12]

Limit address. PMBLIMITR_EL1.LIMIT:Zeros(12) is the address of the first byte in memory after the last byte in the profiling buffer. If the smallest implemented translation granule is not 4KB, then bits[N-1:12] are RES0, where N is the IMPLEMENTATION DEFINED value, Log_2(smallest implemented translation granule).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:8]

Reserved, RES0.

nVM, bit [7]

When FEAT_SPE_nVM is implemented:

Address mode.

nVM	Meaning
0b0	The Profiling Buffer pointers are virtual addresses.
0b1	The Profiling Buffer pointers are: • Physical address in the owning security state if the owning translation regime has no stage 2 translation. • Intermediate physical addresses in the owning security state if the owning translation regime has stage 2 translations.

If the Effective value of PMSCR_EL2.EnVM is 0, then the PE ignores the value of this field, and the Profiling Buffer pointers are always virtual addresses.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is RW.

Otherwise:

Reserved, RES0.

Bit [6]

Reserved, RES0.

PMFZ, bit [5]

When FEAT_SPEv1p2 is implemented:

Freeze PMU on SPE event. Stop PMU event counters when PMBSR_EL1.S == 1.

PMFZ	Meaning
0b0	Do not freeze PMU event counters on Statistical Profiling Buffer Management event.
0b1	Freeze PMU event counters on Statistical Profiling Buffer Management event.

The PMU event counters affected by this control is controlled by PMCR_EL0.FZS and, if EL2 is implemented, MDCR_EL2.HPMFZS. See the descriptions of these control bits for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [4:3]

Reserved, RES0.

FM, bits [2:1]

Fill mode.

FM	Meaning	Applies when
0b00	Fill mode. Stop collection and raise maintenance interrupt on buffer fill.	
0b10	Discard mode. All output is discarded.	FEAT_SPEv1p2 is implemented

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [0]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Profiling Buffer enable

E	Meaning
0b0	All output is discarded.
0b1	Profiling buffer enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing PMBLIMITR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMBLIMITR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1010	0b000

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.PMBLIMITR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2PB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x800];
    else
        X[t, 64] = PMBLIMITR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMBLIMITR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMBLIMITR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MSR PMBLIMITR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1010	0b000

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.PMBLIMITR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2PB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x800] = X[t, 64];
    else
        PMBLIMITR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMBLIMITR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMBLIMITR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.7.3 PMBMAR_EL1, Profiling Buffer Memory Attribute Register

The PMBMAR_EL1 characteristics are:

Purpose

Controls Statistical Profiling Unit accesses to memory.

If the Profiling Buffer pointers specify virtual addresses, the address properties are defined by the translation tables and this register is ignored.

Configuration

This register is present only when FEAT_SPE_nVM is implemented. Otherwise, direct accesses to PMBMAR_EL1 are UNDEFINED.

Attributes

PMBMAR_EL1 is a 64-bit register.

Field descriptions

Bits [63:10]

Reserved, RES0.

SH, bits [9:8]

Profiling Buffer shareability domain. Defines the shareability domain for Normal memory used by the Profiling Buffer.

SH	Meaning
0b00	Non-shareable.
0b10	Outer Shareable.
0b11	Inner Shareable.

All other values are reserved.

This field is ignored when PMBMAR_EL1.Attr specifies any of the following memory types:

- Any Device memory type.
- Normal memory, Inner Non-cacheable, Outer Non-cacheable.

All Device and Normal Inner Non-cacheable Outer Non-cacheable memory regions are always treated as Outer Shareable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Attr, bits [7:0]

Profiling Buffer memory type and attributes. Defines the memory type and, for Normal memory, the cacheability attributes, for memory addressed by the Profiling Buffer.

The encoding of this field is the same as that of a MAIR_ELx.Attr<n> field, as follows:

Attr		Meaning
`0b0000`dd00		Device memory. See encoding of ‘dd’ for the type of Device memory.
`0b0000`dd01		If FEAT_XS is implemented: Device memory with the XS attribute set to 0. See encoding of ‘dd’ for the type of Device memory. Otherwise, UNPREDICTABLE.
`0b0000`dd1x		UNPREDICTABLE.
0boooo iiii	where oooo != 0000 and iiii != 0000	Normal memory. See encoding of ‘oooo’ and ‘iiii’ for the type of Normal memory.
`0b01000000`		If FEAT_XS is implemented: Normal Inner Non-cacheable, Outer Non-cacheable memory with the XS attribute set to 0. Otherwise, UNPREDICTABLE.
`0b10100000`		If FEAT_XS is implemented: Normal Inner Write-through Cacheable, Outer Write-through Cacheable, Read-Allocate, No-Write Allocate, Non-transient memory with the XS attribute set to 0. Otherwise, UNPREDICTABLE.
`0b11110000`		If FEAT_MTE2 is implemented: Tagged Normal Inner Write-Back, Outer Write-Back, Read-Allocate, Write-Allocate Non-transient memory. Otherwise, UNPREDICTABLE.
`0bxxxx0000`	where xxxx != 0000 and xxxx != 0100 and xxxx != 1010 and xxxx != 1111	UNPREDICTABLE.

`dd` is encoded as follows:

‘dd’	Meaning
`0b00`	Device-nGnRnE memory.
`0b01`	Device-nGnRE memory.
`0b10`	Device-nGRE memory.
`0b11`	Device-GRE memory.

`oooo` is encoded as follows:

‘oooo’		Meaning
`0b0000`		See encoding of Attr.
`0b00`RW	where RW != 00	Normal memory, Outer Write-Through Transient.
`0b0100`		Normal memory, Outer Non-cacheable.
`0b01`RW	where RW != 00	Normal memory, Outer Write-Back Transient.
`0b10`RW		Normal memory, Outer Write-Through Non-transient.
`0b11`RW		Normal memory, Outer Write-Back Non-transient.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

`R` encodes the Outer Read-Allocate policy and `W` encodes the Outer Write-Allocate policy.

`iiii` is encoded as follows:

'iiii'		Meaning
`0b0000`		See encoding of Attr.
`0b00`RW	where RW != 00	Normal memory, Inner Write-Through Transient.
`0b0100`		Normal memory, Inner Non-cacheable.
`0b01`RW	where RW != 00	Normal memory, Inner Write-Back Transient.
`0b10`RW		Normal memory, Inner Write-Through Non-transient.
`0b11`RW		Normal memory, Inner Write-Back Non-transient.

`R` encodes the Inner Read-Allocate policy and `W` encodes the Inner Write-Allocate policy.

In `oooo` and `iiii`, `R` and `W` are encoded as follows:

'R' or 'W'	Meaning
`0b0`	No Allocate.
`0b1`	Allocate.

When FEAT_XS is implemented, stage 1 Inner Write-Back Cacheable, Outer Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMBMAR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMBMAR_EL1
```

op0	op1	CRn	CRm	op2
`0b11`	`0b000`	`0b1001`	`0b1010`	`0b101`

```
if !IsFeatureImplemented(FEAT_SPE_nVM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMS4 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGRTR2_EL2.nPMBMAR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        elsif EL2Enabled() && MDCR_EL2.E2PB IN {'x0'} then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.EnPMS4 == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = PMBMAR_EL1;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
            UNDEFINED;
        elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMS4 == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.EnPMS4 == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = PMBMAR_EL1;
    elsif PSTATE.EL == EL3 then
        X[t, 64] = PMBMAR_EL1;
```

```
MSR PMBMAR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1010	0b101

```
if !IsFeatureImplemented(FEAT_SPE_nVM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMS4 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGWTR2_EL2.nPMBMAR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2PB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPMS4 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMBMAR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMS4 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPMS4 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMBMAR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMBMAR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.7.4 PMBPTR_EL1, Profiling Buffer Write Pointer Register

The PMBPTR_EL1 characteristics are:

Purpose

Defines the current write pointer for the profiling buffer.

Configuration

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMBPTR_EL1 are UNDEFINED.

Attributes

PMBPTR_EL1 is a 64-bit register.

Field descriptions

PTR, bits [63:0]

Current write address. Defines the virtual address of the next entry to be written to the buffer.

If PMBIDR_EL1.Align is not zero, then it is IMPLEMENTATION DEFINED whether bits [M-1:0] are RES0 or read/write, where M is an integer between 1 and PMBIDR_EL1.Align inclusive.

The architecture places restrictions on the values software can write to the pointer when the SPU is not in Discard mode. For more information see Restrictions on the current write pointer.

On a management interrupt, PMBPTR_EL1 is frozen.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMBPTR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMBPTR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1010	0b001

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.PMBPTR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2PB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x810];
    else
        X[t, 64] = PMBPTR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMBPTR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMBPTR_EL1;
```

```
MSR PMBPTR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1010	0b001

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.PMBPTR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2PB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x810] = X[t, 64];
    else
        PMBPTR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMBPTR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMBPTR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.7.5 PMBSR_EL1, Profiling Buffer Status/syndrome Register (EL1)

The PMBSR_EL1 characteristics are:

Purpose

Provides syndrome information to software for a Profiling Buffer management event.

Configuration

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMBSR_EL1 are UNDEFINED.

Attributes

PMBSR_EL1 is a 64-bit register.

Field descriptions

Bits [63:56]

Reserved, RES0.

MSS2, bits [55:32]

Management event Specific Syndrome 2. Contains syndrome specific to the management event.

The syndrome contents for each management event are described in the following sections.

MSS2 encoding for other Profiling Buffer management events

Bits [23:0]

Reserved, RES0.

MSS2 encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bits [23:9]

Reserved, RES0.

TopLevel, bit [8]

When FEAT_THE is implemented:

TopLevel. Indicates if the fault was due to TopLevel.

TopLevel	Meaning
0b0	Fault is not due to TopLevel.
0b1	Fault is due to TopLevel.

Otherwise:

Reserved, RES0.

AssuredOnly, bit [7]

When FEAT_THE is implemented, PMBSR_EL1.EC == '100101', and GetPMBSR_EL1_FSC() IN {'0011xx'}:

AssuredOnly flag. If a memory access generates a stage 2 Data Abort, then this field holds information about the fault.

AssuredOnly	Meaning
0b0	Data Abort is not due to AssuredOnly.
0b1	Data Abort is due to AssuredOnly.

Otherwise:

Reserved, RES0.

Overlay, bit [6]

When (FEAT_S1POE is implemented or FEAT_S2POE is implemented) and GetPMBSR_EL1_FSC() IN {'0011xx'}:

Overlay flag. If a memory access generates a Data Abort for a Permission fault, then this field holds information about the fault.

Overlay	Meaning
0b0	Data Abort is not due to Overlay Permissions.
0b1	Data Abort is due to Overlay Permissions.

Otherwise:

Reserved, RES0.

DirtyBit, bit [5]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When (FEAT_S1PIE is implemented or FEAT_S2PIE is implemented) and GetPMBSR_EL1_FSC() IN {'0011xx'}:

DirtyBit flag. If a write access to memory generates a Data Abort for a Permission fault using Indirect Permission, then this field holds information about the fault.

DirtyBit	Meaning
0b0	Permission Fault is not due to dirty state.
0b1	Permission Fault is due to dirty state.

Otherwise:

Reserved, RES0.

Bits [4:0]

Reserved, RES0.

MSS2 encoding for Granule Protection Check faults on write to Profiling Buffer

23 … 0
RES0

Bits [23:0]

Reserved, RES0.

MSS2 encoding for Profiling Buffer management event for an IMPLEMENTATION DEFINED reason

23 … 0
IMPLEMENTATION DEFINED

IMPLEMENTATION DEFINED, bits [23:0]

IMPLEMENTATION DEFINED.

EC, bits [31:26]

Event class. Top-level description of the cause of the Profiling Buffer management event.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EC	Meaning	MSS	MSS2	Applies when
0b000000	Other Profiling Buffer management event. All Profiling Buffer management events other than those described by the other defined Event class codes.	MSS encoding for other Profiling Buffer management events	MSS2 encoding for other Profiling Buffer management events	
0b011110	Granule Protection Check fault on write to Profiling Buffer, other than Granule Protection Fault (GPF). That is, any of the following: • Granule Protection Table (GPT) address size fault. • GPT walk fault. • Synchronous External abort on GPT fetch. A GPF on translation table walk or update is reported as either a Stage 1 or Stage 2 Data Abort, as appropriate. Other GPFs are reported as a Stage 1 Data Abort.	MSS encoding for Granule Protection Check faults on write to Profiling Buffer	MSS2 encoding for Granule Protection Check faults on write to Profiling Buffer	FEAT_RME is implemented
0b011111	Profiling Buffer management event for an IMPLEMENTATION DEFINED reason.	MSS encoding for Profiling Buffer management event for an IMPLEMENTATION DEFINED reason	MSS2 encoding for Profiling Buffer management event for an IMPLEMENTATION DEFINED reason	
0b100100	Stage 1 Data Abort on write to Profiling Buffer.	MSS encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer	MSS2 encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer	
0b100101	Stage 2 Data Abort on write to Profiling Buffer.	MSS encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer	MSS2 encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer	

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [25:20]

Reserved, RES0.

DL, bit [19]

Partial record lost. Following a buffer management event other than an asynchronous External abort, indicates whether the last record written to the Profiling Buffer is complete.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

DL	Meaning
0b0	PMBPTR_EL1 points to the first byte after the last complete record written to the Profiling Buffer.
0b1	Part of a record was lost because of a buffer management event or synchronous External abort. PMBPTR_EL1 might not point to the first byte after the last complete record written to the buffer, and so restarting collection might result in a data record stream that software cannot parse.

When the buffer management event was because of an asynchronous External abort, this bit is set to 1 and software must not assume that any valid data has been written to the Profiling Buffer.

This bit is RES0 if the PE never sets this bit as a result of a buffer management event caused by an asynchronous External abort.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EA, bit [18]

External abort.

EA	Meaning
0b0	An External abort has not been asserted.
0b1	An External abort has been asserted and detected by the Statistical Profiling Unit.

This bit is RES0 if the PE never sets this bit as the result of an External abort.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

S, bit [17]

Service. Indicates that a Profiling Buffer management event has been recorded.

S	Meaning
0b0	No Profiling Buffer management event for EL1 has been recorded.
0b1	A Profiling Buffer management event for EL1 has been recorded.

When FEAT_SPE_EXC is implemented, this field indicates a management event for EL1.

If FEAT_SPE_EXC is implemented and the SPE Profiling exception for EL1 is enabled, then when this field is 1, an SPE Profiling exception for EL1 is pending

If FEAT_SPE_EXC is not implemented or the SPE Profiling exception for EL1 is disabled, then this field drives the **PMBIRQ** Profiling Buffer interrupt request signal.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

COLL, bit [16]

Collision detected.

COLL	Meaning
0b0	No collision events detected.
0b1	At least one collision event was recorded.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

MSS, bits [15:0]

Management Event Specific Syndrome. Contains syndrome specific to the Profiling Buffer management event.

The syndrome contents for each Profiling Buffer management event are described in the following sections.

MSS encoding for other Profiling Buffer management events

Bits [15:6]

Reserved, RES0.

BSC, bits [5:0]

Profiling Buffer status code

BSC	Meaning
0b000000	Collection not stopped, or access not allowed.
0b000001	Profiling Buffer filled.
0b000100	Buffer size. The requested Profiling Buffer size was too large.

All other values are reserved.

MSS encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer

Bits [15:6]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FSC, bits [5:0]

Fault status code

FSC	Meaning	Applies when
0b000000	Address size fault, level 0 of translation or translation table base register.	
0b000001	Address size fault, level 1.	
0b000010	Address size fault, level 2.	
0b000011	Address size fault, level 3.	
0b000100	Translation fault, level 0.	
0b000101	Translation fault, level 1.	
0b000110	Translation fault, level 2.	
0b000111	Translation fault, level 3.	
0b001001	Access flag fault, level 1.	
0b001010	Access flag fault, level 2.	
0b001011	Access flag fault, level 3.	
0b001000	Access flag fault, level 0.	FEAT_LPA2 is implemented
0b001100	Permission fault, level 0.	FEAT_LPA2 is implemented
0b001101	Permission fault, level 1.	
0b001110	Permission fault, level 2.	
0b001111	Permission fault, level 3.	
0b010000	Synchronous External abort, not on translation table walk or hardware update of translation table.	
0b010001	Asynchronous External abort.	
0b010010	Synchronous External abort on translation table walk or hardware update of translation table, level -2.	FEAT_D128 is implemented
0b010011	Synchronous External abort on translation table walk or hardware update of translation table, level -1.	FEAT_LPA2 is implemented
0b010100	Synchronous External abort on translation table walk or hardware update of translation table, level 0.	
0b010101	Synchronous External abort on translation table walk or hardware update of translation table, level 1.	
0b010110	Synchronous External abort on translation table walk or hardware update of translation table, level 2.	
0b010111	Synchronous External abort on translation table walk or hardware update of translation table, level 3.	
0b011011	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1.	FEAT_LPA2 is implemented and FEAT_RAS is not implemented

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FSC	Meaning	Applies when
0b100001	Alignment fault.	
0b100010	Granule Protection Fault on translation table walk or hardware update of translation table, level -2.	FEAT_D128 is implemented and FEAT_RME is implemented
0b100011	Granule Protection Fault on translation table walk or hardware update of translation table, level -1.	FEAT_RME is implemented and FEAT_LPA2 is implemented
0b100100	Granule Protection Fault on translation table walk or hardware update of translation table, level 0.	FEAT_RME is implemented
0b100101	Granule Protection Fault on translation table walk or hardware update of translation table, level 1.	FEAT_RME is implemented
0b100110	Granule Protection Fault on translation table walk or hardware update of translation table, level 2.	FEAT_RME is implemented
0b100111	Granule Protection Fault on translation table walk or hardware update of translation table, level 3.	FEAT_RME is implemented
0b101000	Granule Protection Fault, not on translation table walk or hardware update of translation table.	FEAT_RME is implemented
0b101001	Address size fault, level -1.	FEAT_LPA2 is implemented
0b101010	Translation fault, level -2.	FEAT_D128 is implemented
0b101011	Translation fault, level -1.	FEAT_LPA2 is implemented
0b101100	Address Size fault, level -2.	FEAT_D128 is implemented
0b110000	TLB conflict abort.	
0b110001	Unsupported atomic hardware update fault.	FEAT_HAFDBS is implemented

All other values are reserved.

It is IMPLEMENTATION DEFINED whether each of the Access Flag fault, asynchronous External abort and synchronous External abort, Alignment fault, and TLB Conflict abort values can be generated by the PE. For more information see ‘Faults and Watchpoints’.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

MSS encoding for Granule Protection Check faults on write to Profiling Buffer

Bits [15:0]

Reserved, RES0.

MSS encoding for Profiling Buffer management event for an IMPLEMENTATION DEFINED reason

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

15	0
IMPLEMENTATION DEFINED	

IMPLEMENTATION DEFINED, bits [15:0]

IMPLEMENTATION DEFINED.

Accessing PMBSR_EL1

When the Effective value of HCR_EL2.E2H is 1 and FEAT_SPE_EXC is implemented, without explicit synchronization, accesses from EL3 using the accessor name `PMBSR_EL1` or `PMBSR_EL12` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMBSR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1010	0b011

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.PMBSR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2PB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} && (EffectivePMSCR_EL2_EE() == '00' || PMSCR_EL1.EE == '00'
         ↪|| EffectiveHCR_EL2_NVx() == '111') then
        X[t, 64] = NVMem[0x820];
    else
        X[t, 64] = PMBSR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectivePMSCR_EL2_EE() != '00' && ELIsInHost(EL2) then
        X[t, 64] = PMBSR_EL2;
    else
        X[t, 64] = PMBSR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMBSR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MSR PMBSR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1010	0b011

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.PMBSR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2PB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} && (EffectivePMSCR_EL2_EE() == '00' || PMSCR_EL1.EE == '00'
         ↪|| EffectiveHCR_EL2_NVx() == '111') then
        NVMem[0x820] = X[t, 64];
    else
        PMBSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectivePMSCR_EL2_EE() != '00' && ELIsInHost(EL2) then
        PMBSR_EL2 = X[t, 64];
    else
        PMBSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMBSR_EL1 = X[t, 64];
```

When FEAT_SPE_EXC is implemented and FEAT_VHE is implemented MRS <Xt>, PMBSR_EL12

op0	op1	CRn	CRm	op2
0b11	0b101	0b1001	0b1010	0b011

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x820];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if ELIsInHost(EL2) then
            if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
                UNDEFINED;
            elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
                if EL3SDDUndef() then
                    UNDEFINED;
                else
                    AArch64.SystemAccessTrap(EL3, 0x18);
            else
                X[t, 64] = PMBSR_EL1;
        else
            UNDEFINED;
    elsif PSTATE.EL == EL3 then
        if ELIsInHost(EL2) then
            X[t, 64] = PMBSR_EL1;
        else
            UNDEFINED;
```

When FEAT_SPE_EXC is implemented and FEAT_VHE is implemented MSR PMBSR_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b1001	0b1010	0b011

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x820] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
            UNDEFINED;
        elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMBSR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        PMBSR_EL1 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.7.6 PMBSR_EL2, Profiling Buffer Syndrome Register (EL2)

The PMBSR_EL2 characteristics are:

Purpose

Provides syndrome information to software for a Profiling Buffer management event.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_SPE_EXC is implemented. Otherwise, direct accesses to PMBSR_EL2 are UNDEFINED.

Attributes

PMBSR_EL2 is a 64-bit register.

Field descriptions

Bits [63:56]

Reserved, RES0.

MSS2, bits [55:32]

Management event Specific Syndrome 2. Contains syndrome specific to the management event.

The syndrome contents for each management event are described in the following sections.

MSS2 encoding for other Profiling Buffer management events

Bits [23:0]

Reserved, RES0.

MSS2 encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bits [23:9]

Reserved, RES0.

TopLevel, bit [8]

When FEAT_THE is implemented:

TopLevel. Indicates if the fault was due to TopLevel.

TopLevel	Meaning
0b0	Fault is not due to TopLevel.
0b1	Fault is due to TopLevel.

Otherwise:

Reserved, RES0.

AssuredOnly, bit [7]

When FEAT_THE is implemented, PMBSR_EL2.EC == '100101', and GetPMBSR_EL2_FSC() IN {'0011xx'}:

AssuredOnly flag. If a memory access generates a stage 2 Data Abort, then this field holds information about the fault.

AssuredOnly	Meaning
0b0	Data Abort is not due to AssuredOnly.
0b1	Data Abort is due to AssuredOnly.

Otherwise:

Reserved, RES0.

Overlay, bit [6]

When (FEAT_S1POE is implemented or FEAT_S2POE is implemented) and GetPMBSR_EL2_FSC() IN {'0011xx'}:

Overlay flag. If a memory access generates a Data Abort for a Permission fault, then this field holds information about the fault.

Overlay	Meaning
0b0	Data Abort is not due to Overlay Permissions.
0b1	Data Abort is due to Overlay Permissions.

Otherwise:

Reserved, RES0.

DirtyBit, bit [5]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When (FEAT_S1PIE is implemented or FEAT_S2PIE is implemented) and GetPMBSR_EL2_FSC() IN {'0011xx'}:

DirtyBit flag. If a write access to memory generates a Data Abort for a Permission fault using Indirect Permission, then this field holds information about the fault.

DirtyBit	Meaning
`0b0`	Permission Fault is not due to dirty state.
`0b1`	Permission Fault is due to dirty state.

Otherwise:

Reserved, RES0.

Bits [4:0]

Reserved, RES0.

MSS2 encoding for Granule Protection Check faults on write to Profiling Buffer

23 … 0
RES0

Bits [23:0]

Reserved, RES0.

MSS2 encoding for Profiling Buffer management event for an IMPLEMENTATION DEFINED reason

23 … 0
IMPLEMENTATION DEFINED

IMPLEMENTATION DEFINED, bits [23:0]

IMPLEMENTATION DEFINED.

EC, bits [31:26]

Event class. Top-level description of the cause of the Profiling Buffer management event.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EC	Meaning	MSS	MSS2	Applies when
0b000000	Other Profiling Buffer management event. All Profiling Buffer management events other than those described by the other defined Event class codes.	MSS encoding for other Profiling Buffer management events	MSS2 encoding for other Profiling Buffer management events	
0b011110	Granule Protection Check fault on write to Profiling Buffer, other than Granule Protection Fault (GPF). That is, any of the following: • Granule Protection Table (GPT) address size fault. • GPT walk fault. • Synchronous External abort on GPT fetch. A GPF on translation table walk or update is reported as either a Stage 1 or Stage 2 Data Abort, as appropriate. Other GPFs are reported as a Stage 1 Data Abort.	MSS encoding for Granule Protection Check faults on write to Profiling Buffer	MSS2 encoding for Granule Protection Check faults on write to Profiling Buffer	FEAT_RME is implemented
0b011111	Profiling Buffer management event for an IMPLEMENTATION DEFINED reason.	MSS encoding for Profiling Buffer management event for an IMPLEMENTATION DEFINED reason	MSS2 encoding for Profiling Buffer management event for an IMPLEMENTATION DEFINED reason	
0b100100	Stage 1 Data Abort on write to Profiling Buffer.	MSS encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer	MSS2 encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer	
0b100101	Stage 2 Data Abort on write to Profiling Buffer.	MSS encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer	MSS2 encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer	

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [25:20]

Reserved, RES0.

DL, bit [19]

Partial record lost. Following a buffer management event other than an asynchronous External abort, indicates whether the last record written to the Profiling Buffer is complete.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

DL	Meaning
0b0	PMBPTR_EL1 points to the first byte after the last complete record written to the Profiling Buffer.
0b1	Part of a record was lost because of a buffer management event or synchronous External abort. PMBPTR_EL1 might not point to the first byte after the last complete record written to the buffer, and so restarting collection might result in a data record stream that software cannot parse.

When the buffer management event was because of an asynchronous External abort, this bit is set to 1 and software must not assume that any valid data has been written to the Profiling Buffer.

This bit is RES0 if the PE never sets this bit as a result of a buffer management event caused by an asynchronous External abort.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EA, bit [18]

External abort.

EA	Meaning
0b0	An External abort has not been asserted.
0b1	An External abort has been asserted and detected by the Statistical Profiling Unit.

This bit is RES0 if the PE never sets this bit as the result of an External abort.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

S, bit [17]

Service. Indicates that a Profiling Buffer management event has been recorded.

S	Meaning
0b0	No Profiling Buffer management event for EL2 has been recorded.
0b1	A Profiling Buffer management event for EL2 has been recorded.

When FEAT_SPE_EXC is implemented, this field indicates a management event for EL2.

If the SPE Profiling exception for EL2 is enabled, then when this field is 1, an SPE Profiling exception for EL2 is pending

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COLL, bit [16]

Collision detected.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

COLL	Meaning
0b0	No collision events detected.
0b1	At least one collision event was recorded.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

MSS, bits [15:0]

Management Event Specific Syndrome. Contains syndrome specific to the Profiling Buffer management event.

The syndrome contents for each Profiling Buffer management event are described in the following sections.

MSS encoding for other Profiling Buffer management events

Bits [15:6]

Reserved, RES0.

BSC, bits [5:0]

Profiling Buffer status code

BSC	Meaning
0b000000	Collection not stopped, or access not allowed.
0b000001	Profiling Buffer filled.
0b000100	Buffer size. The requested Profiling Buffer size was too large.

All other values are reserved.

MSS encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer

Bits [15:6]

Reserved, RES0.

FSC, bits [5:0]

Fault status code

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FSC	Meaning	Applies when
0b000000	Address size fault, level 0 of translation or translation table base register.	
0b000001	Address size fault, level 1.	
0b000010	Address size fault, level 2.	
0b000011	Address size fault, level 3.	
0b000100	Translation fault, level 0.	
0b000101	Translation fault, level 1.	
0b000110	Translation fault, level 2.	
0b000111	Translation fault, level 3.	
0b001001	Access flag fault, level 1.	
0b001010	Access flag fault, level 2.	
0b001011	Access flag fault, level 3.	
0b001000	Access flag fault, level 0.	FEAT_LPA2 is implemented
0b001100	Permission fault, level 0.	FEAT_LPA2 is implemented
0b001101	Permission fault, level 1.	
0b001110	Permission fault, level 2.	
0b001111	Permission fault, level 3.	
0b010000	Synchronous External abort, not on translation table walk or hardware update of translation table.	
0b010001	Asynchronous External abort.	
0b010010	Synchronous External abort on translation table walk or hardware update of translation table, level -2.	FEAT_D128 is implemented
0b010011	Synchronous External abort on translation table walk or hardware update of translation table, level -1.	FEAT_LPA2 is implemented
0b010100	Synchronous External abort on translation table walk or hardware update of translation table, level 0.	
0b010101	Synchronous External abort on translation table walk or hardware update of translation table, level 1.	
0b010110	Synchronous External abort on translation table walk or hardware update of translation table, level 2.	
0b010111	Synchronous External abort on translation table walk or hardware update of translation table, level 3.	
0b011011	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1.	FEAT_LPA2 is implemented and FEAT_RAS is not implemented
0b100001	Alignment fault.	

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FSC	Meaning	Applies when
0b100010	Granule Protection Fault on translation table walk or hardware update of translation table, level -2.	FEAT_D128 is implemented and FEAT_RME is implemented
0b100011	Granule Protection Fault on translation table walk or hardware update of translation table, level -1.	FEAT_RME is implemented and FEAT_LPA2 is implemented
0b100100	Granule Protection Fault on translation table walk or hardware update of translation table, level 0.	FEAT_RME is implemented
0b100101	Granule Protection Fault on translation table walk or hardware update of translation table, level 1.	FEAT_RME is implemented
0b100110	Granule Protection Fault on translation table walk or hardware update of translation table, level 2.	FEAT_RME is implemented
0b100111	Granule Protection Fault on translation table walk or hardware update of translation table, level 3.	FEAT_RME is implemented
0b101000	Granule Protection Fault, not on translation table walk or hardware update of translation table.	FEAT_RME is implemented
0b101001	Address size fault, level -1.	FEAT_LPA2 is implemented
0b101010	Translation fault, level -2.	FEAT_D128 is implemented
0b101011	Translation fault, level -1.	FEAT_LPA2 is implemented
0b101100	Address Size fault, level -2.	FEAT_D128 is implemented
0b110000	TLB conflict abort.	
0b110001	Unsupported atomic hardware update fault.	FEAT_HAFDBS is implemented

All other values are reserved.

It is IMPLEMENTATION DEFINED whether each of the Access Flag fault, asynchronous External abort and synchronous External abort, Alignment fault, and TLB Conflict abort values can be generated by the PE. For more information see ‘Faults and Watchpoints’.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

MSS encoding for Granule Protection Check faults on write to Profiling Buffer

Bits [15:0]

Reserved, RES0.

MSS encoding for Profiling Buffer management event for an IMPLEMENTATION DEFINED reason

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

15	0
IMPLEMENTATION DEFINED	

IMPLEMENTATION DEFINED, bits [15:0]

IMPLEMENTATION DEFINED.

Accessing PMBSR_EL2

When the Effective value of HCR_EL2.E2H is 1 and FEAT_SPE_EXC is implemented, without explicit synchronization, accesses from EL2 using the accessor name `PMBSR_EL2` or `PMBSR_EL1` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMBSR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1001	0b1010	0b011

```
if !IsFeatureImplemented(FEAT_SPE_EXC) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.PMSEE == '00' then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.PMSEE == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMBSR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMBSR_EL2;
```

```
MSR PMBSR_EL2, <Xt>
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b100	0b1001	0b1010	0b011

```
if !IsFeatureImplemented(FEAT_SPE_EXC) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.PMSEE == '00' then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.PMSEE == '00' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMBSR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMBSR_EL2 = X[t, 64];
```

MRS <Xt>, PMBSR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1010	0b011

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.PMBSR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2PB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} && (EffectivePMSCR_EL2_EE() == '00' || PMSCR_EL1.EE == '00'
         ↪|| EffectiveHCR_EL2_NVx() == '111') then
        X[t, 64] = NVMem[0x820];
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        X[t, 64] = PMBSR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectivePMSCR_EL2_EE() != '00' && ELIsInHost(EL2) then
        X[t, 64] = PMBSR_EL2;
    else
        X[t, 64] = PMBSR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMBSR_EL1;
```

```
MSR PMBSR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1010	0b011

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.PMBSR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2PB IN {'x0'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} && (EffectivePMSCR_EL2_EE() == '00' || PMSCR_EL1.EE == '00'
         ↪|| EffectiveHCR_EL2_NVx() == '111') then
        NVMem[0x820] = X[t, 64];
    else
        PMBSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectivePMSCR_EL2_EE() != '00' && ELIsInHost(EL2) then
        PMBSR_EL2 = X[t, 64];
    else
        PMBSR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMBSR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.7.7 PMBSR_EL3, Profiling Buffer Syndrome Register (EL3)

The PMBSR_EL3 characteristics are:

Purpose

Provides syndrome information to software for a Profiling Buffer management event.

Configuration

This register is present only when FEAT_SPE_EXC is implemented and EL3 is implemented. Otherwise, direct accesses to PMBSR_EL3 are UNDEFINED.

Attributes

PMBSR_EL3 is a 64-bit register.

Field descriptions

Bits [63:56]

Reserved, RES0.

MSS2, bits [55:32]

Management event Specific Syndrome 2. Contains syndrome specific to the management event.

The syndrome contents for each management event are described in the following sections.

MSS2 encoding for other Profiling Buffer management events

Bits [23:0]

Reserved, RES0.

MSS2 encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bits [23:9]

Reserved, RES0.

TopLevel, bit [8]

When FEAT_THE is implemented:

TopLevel. Indicates if the fault was due to TopLevel.

TopLevel	Meaning
0b0	Fault is not due to TopLevel.
0b1	Fault is due to TopLevel.

Otherwise:

Reserved, RES0.

AssuredOnly, bit [7]

When FEAT_THE is implemented, PMBSR_EL3.EC == '100101', and GetPMBSR_EL3_FSC() IN {'0011xx'}:

AssuredOnly flag. If a memory access generates a stage 2 Data Abort, then this field holds information about the fault.

AssuredOnly	Meaning
0b0	Data Abort is not due to AssuredOnly.
0b1	Data Abort is due to AssuredOnly.

Otherwise:

Reserved, RES0.

Overlay, bit [6]

When (FEAT_S1POE is implemented or FEAT_S2POE is implemented) and GetPMBSR_EL3_FSC() IN {'0011xx'}:

Overlay flag. If a memory access generates a Data Abort for a Permission fault, then this field holds information about the fault.

Overlay	Meaning
0b0	Data Abort is not due to Overlay Permissions.
0b1	Data Abort is due to Overlay Permissions.

Otherwise:

Reserved, RES0.

DirtyBit, bit [5]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When (FEAT_S1PIE is implemented or FEAT_S2PIE is implemented) and GetPMBSR_EL3_FSC() IN {'0011xx'}:

DirtyBit flag. If a write access to memory generates a Data Abort for a Permission fault using Indirect Permission, then this field holds information about the fault.

DirtyBit	Meaning
0b0	Permission Fault is not due to dirty state.
0b1	Permission Fault is due to dirty state.

Otherwise:

Reserved, RES0.

Bits [4:0]

Reserved, RES0.

MSS2 encoding for Granule Protection Check faults on write to Profiling Buffer

23 … 0
RES0

Bits [23:0]

Reserved, RES0.

MSS2 encoding for Profiling Buffer management event for an IMPLEMENTATION DEFINED reason

23 … 0
IMPLEMENTATION DEFINED

IMPLEMENTATION DEFINED, bits [23:0]

IMPLEMENTATION DEFINED.

EC, bits [31:26]

Event class. Top-level description of the cause of the Profiling Buffer management event.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EC	Meaning	MSS	MSS2	Applies when
0b000000	Other Profiling Buffer management event. All Profiling Buffer management events other than those described by the other defined Event class codes.	MSS encoding for other Profiling Buffer management events	MSS2 encoding for other Profiling Buffer management events	
0b011110	Granule Protection Check fault on write to Profiling Buffer, other than Granule Protection Fault (GPF). That is, any of the following: • Granule Protection Table (GPT) address size fault. • GPT walk fault. • Synchronous External abort on GPT fetch. A GPF on translation table walk or update is reported as either a Stage 1 or Stage 2 Data Abort, as appropriate. Other GPFs are reported as a Stage 1 Data Abort.	MSS encoding for Granule Protection Check faults on write to Profiling Buffer	MSS2 encoding for Granule Protection Check faults on write to Profiling Buffer	FEAT_RME is implemented
0b011111	Profiling Buffer management event for an IMPLEMENTATION DEFINED reason.	MSS encoding for Profiling Buffer management event for an IMPLEMENTATION DEFINED reason	MSS2 encoding for Profiling Buffer management event for an IMPLEMENTATION DEFINED reason	
0b100100	Stage 1 Data Abort on write to Profiling Buffer.	MSS encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer	MSS2 encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer	
0b100101	Stage 2 Data Abort on write to Profiling Buffer.	MSS encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer	MSS2 encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer	

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [25:20]

Reserved, RES0.

DL, bit [19]

Partial record lost. Following a buffer management event other than an asynchronous External abort, indicates whether the last record written to the Profiling Buffer is complete.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

DL	Meaning
0b0	PMBPTR_EL1 points to the first byte after the last complete record written to the Profiling Buffer.
0b1	Part of a record was lost because of a buffer management event or synchronous External abort. PMBPTR_EL1 might not point to the first byte after the last complete record written to the buffer, and so restarting collection might result in a data record stream that software cannot parse.

When the buffer management event was because of an asynchronous External abort, this bit is set to 1 and software must not assume that any valid data has been written to the Profiling Buffer.

This bit is RES0 if the PE never sets this bit as a result of a buffer management event caused by an asynchronous External abort.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EA, bit [18]

External abort.

EA	Meaning
0b0	An External abort has not been asserted.
0b1	An External abort has been asserted and detected by the Statistical Profiling Unit.

This bit is RES0 if the PE never sets this bit as the result of an External abort.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

S, bit [17]

Service. Indicates that a Profiling Buffer management event has been recorded.

S	Meaning
0b0	No Profiling Buffer management event for EL3 has been recorded.
0b1	A Profiling Buffer management event for EL3 has been recorded.

When FEAT_SPE_EXC is implemented, this field indicates a management event for EL3.

If the SPE Profiling exception for EL3 is enabled, then when this field is 1, an SPE Profiling exception for EL3 is pending

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COLL, bit [16]

Collision detected.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

COLL	Meaning
0b0	No collision events detected.
0b1	At least one collision event was recorded.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

MSS, bits [15:0]

Management Event Specific Syndrome. Contains syndrome specific to the Profiling Buffer management event.

The syndrome contents for each Profiling Buffer management event are described in the following sections.

MSS encoding for other Profiling Buffer management events

Bits [15:6]

Reserved, RES0.

BSC, bits [5:0]

Profiling Buffer status code

BSC	Meaning
0b000000	Collection not stopped, or access not allowed.
0b000001	Profiling Buffer filled.
0b000100	Buffer size. The requested Profiling Buffer size was too large.

All other values are reserved.

MSS encoding for stage 1 or stage 2 Data Aborts on write to Profiling Buffer

Bits [15:6]

Reserved, RES0.

FSC, bits [5:0]

Fault status code

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FSC	Meaning	Applies when
0b000000	Address size fault, level 0 of translation or translation table base register.	
0b000001	Address size fault, level 1.	
0b000010	Address size fault, level 2.	
0b000011	Address size fault, level 3.	
0b000100	Translation fault, level 0.	
0b000101	Translation fault, level 1.	
0b000110	Translation fault, level 2.	
0b000111	Translation fault, level 3.	
0b001001	Access flag fault, level 1.	
0b001010	Access flag fault, level 2.	
0b001011	Access flag fault, level 3.	
0b001000	Access flag fault, level 0.	FEAT_LPA2 is implemented
0b001100	Permission fault, level 0.	FEAT_LPA2 is implemented
0b001101	Permission fault, level 1.	
0b001110	Permission fault, level 2.	
0b001111	Permission fault, level 3.	
0b010000	Synchronous External abort, not on translation table walk or hardware update of translation table.	
0b010001	Asynchronous External abort.	
0b010010	Synchronous External abort on translation table walk or hardware update of translation table, level -2.	FEAT_D128 is implemented
0b010011	Synchronous External abort on translation table walk or hardware update of translation table, level -1.	FEAT_LPA2 is implemented
0b010100	Synchronous External abort on translation table walk or hardware update of translation table, level 0.	
0b010101	Synchronous External abort on translation table walk or hardware update of translation table, level 1.	
0b010110	Synchronous External abort on translation table walk or hardware update of translation table, level 2.	
0b010111	Synchronous External abort on translation table walk or hardware update of translation table, level 3.	
0b011011	Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1.	FEAT_LPA2 is implemented and FEAT_RAS is not implemented
0b100001	Alignment fault.	

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FSC	Meaning	Applies when
0b100010	Granule Protection Fault on translation table walk or hardware update of translation table, level -2.	FEAT_D128 is implemented and FEAT_RME is implemented
0b100011	Granule Protection Fault on translation table walk or hardware update of translation table, level -1.	FEAT_RME is implemented and FEAT_LPA2 is implemented
0b100100	Granule Protection Fault on translation table walk or hardware update of translation table, level 0.	FEAT_RME is implemented
0b100101	Granule Protection Fault on translation table walk or hardware update of translation table, level 1.	FEAT_RME is implemented
0b100110	Granule Protection Fault on translation table walk or hardware update of translation table, level 2.	FEAT_RME is implemented
0b100111	Granule Protection Fault on translation table walk or hardware update of translation table, level 3.	FEAT_RME is implemented
0b101000	Granule Protection Fault, not on translation table walk or hardware update of translation table.	FEAT_RME is implemented
0b101001	Address size fault, level -1.	FEAT_LPA2 is implemented
0b101010	Translation fault, level -2.	FEAT_D128 is implemented
0b101011	Translation fault, level -1.	FEAT_LPA2 is implemented
0b101100	Address Size fault, level -2.	FEAT_D128 is implemented
0b110000	TLB conflict abort.	
0b110001	Unsupported atomic hardware update fault.	FEAT_HAFDBS is implemented

All other values are reserved.

It is IMPLEMENTATION DEFINED whether each of the Access Flag fault, asynchronous External abort and synchronous External abort, Alignment fault, and TLB Conflict abort values can be generated by the PE. For more information see ‘Faults and Watchpoints’.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

MSS encoding for Granule Protection Check faults on write to Profiling Buffer

Bits [15:0]

Reserved, RES0.

MSS encoding for Profiling Buffer management event for an IMPLEMENTATION DEFINED reason

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

15	0
IMPLEMENTATION DEFINED	

IMPLEMENTATION DEFINED, bits [15:0]

IMPLEMENTATION DEFINED.

Accessing PMBSR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMBSR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1001	0b1010	0b011

```
if !(IsFeatureImplemented(FEAT_SPE_EXC) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMBSR_EL3;
```

```
MSR PMBSR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1001	0b1010	0b011

```
if !(IsFeatureImplemented(FEAT_SPE_EXC) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    PMBSR_EL3 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.7.8 PMSCR_EL1, Statistical Profiling Control Register (EL1)

The PMSCR_EL1 characteristics are:

Purpose

Provides EL1 controls for Statistical Profiling.

Configuration

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMSCR_EL1 are UNDEFINED.

Attributes

PMSCR_EL1 is a 64-bit register.

Field descriptions

Bits [63:12]

Reserved, RES0.

EnVM, bit [11]

When FEAT_SPE_nVM is implemented and FEAT_NV is implemented:

Reserved for software use in nested virtualization. See also PMSCR_EL2.EnVM.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

KE, bit [10]

When FEAT_SPE_EXC is implemented:

Kernel exception enable for SPE Profiling exceptions taken to EL1.

KE	Meaning
0b0	SPE Profiling exceptions taken to EL1 are always masked at EL1.
0b1	Enabled SPE Profiling exceptions taken to EL1 are masked at EL1 when PSTATE.PM is 1 and unmasked when PSTATE.PM is 0.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EE, bits [9:8]

When FEAT_SPE_EXC is implemented:

Exception Enable.

EE	Meaning	Applies when
`0b00`	Disabled. SPE Profiling exceptions for EL1 are disabled. All of the following apply: • Unless enabled by a higher Exception level, SPE Profiling exceptions are not generated. • PMBSR_EL1.S drives the interrupt request signal **PMBIRQ**. • Accesses to PMBSR_EL1 at EL1 ignore the value of HCR_EL2.NV1.	
`0b01`	Reserved for software use in nested virtualization. Behaves as `0b00` for the purpose of controlling the SPE Profiling exception and interrupt request signal **PMBIRQ**, and as `0b11` for the purpose of accesses to PMBSR_EL1.	FEAT_NV is implemented
`0b10`	Reserved for software use in nested virtualization. Behaves as `0b11` for the purposes of controlling the SPE Profiling exception and interrupt request signal **PMBIRQ**, and accesses to PMBSR_EL1.	FEAT_NV is implemented
`0b11`	Enabled. SPE Profiling exceptions for EL1 are enabled, as follows: • All Profiling Buffer management events are recorded in PMBSR_EL1, unless they are configured to be recorded in PMBSR_EL3 by MDCR_EL3.PMSEE or PMBSR_EL2 by PMSCR_EL2.EE. • SPE Profiling exceptions are generated and taken to EL1 when unmasked and PMBSR_EL1.S is 1, unless the Effective value of HCR_EL2.TGE is 1, in which case the exception is taken to EL2. • The interrupt request signal **PMBIRQ** is not asserted.	

For more information on the values reserved for software use in nested virtualization, see PMSCR_EL2.EE.

If the Effective value of PMSCR_EL2.EE is `0b00`, then the Effective value of PMSCR_EL1.EE is `0b00`.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to `'00'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PCT, bits [7:6]

When EL2 is implemented:

Physical Timestamp. If timestamp sampling is enabled and the Profiling Buffer owning Exception level is EL1, requests which timestamp counter value is collected.

If FEAT_ECV is implemented, this is a two-bit field as shown. Otherwise, bit[7] is RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

PCT	Meaning	Applies when
0b00	Virtual timestamp. The collected timestamp is the physical counter minus the value of CNTVOFF_EL2.	
0b01	Physical timestamp. The collected timestamp is the physical counter.	
0b11	Guest physical timestamp. The collected timestamp is the physical counter minus a physical offset. If any of the following are true, the physical offset is zero, otherwise the physical offset is the value of CNTPOFF_EL2: • SCR_EL3.ECVEn == 0. • CNTHCTL_EL2.ECV == 0. • FEAT_ECV_POFF is not implemented.	FEAT_ECV is implemented

When EL2 is implemented, all of the following apply:

- If the Profiling Buffer owning Exception level is EL1 and EL2 is enabled in the owning Security state, then the value of PMSCR_EL2.PCT might override or modify the meaning of this field.
- If the Profiling Buffer owning Exception level is EL2, then this field is ignored by the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Physical Timestamp. Reserved. This field reads as 0b01 and ignores writes. Software should treat this field as UNK/SBZP.

When EL2 is not implemented, the Effective values of CNTVOFF_EL2 and CNTPOFF_EL2 are zero, meaning the virtual counter and physical counter have the same value.

TS, bit [5]

Timestamp sample enable. Enables recording of a Timestamp packet when the owning Exception level is EL1.

TS	Meaning
0b0	Timestamp packet recording disabled.
0b1	Timestamp packet recording enabled.

If the Profiling Buffer owning Exception level is EL2, then this field is ignored by the PE. For more information, see Controlling the data that is collected.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PA, bit [4]

Physical Address sample enable.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

PA	Meaning
0b0	Physical addresses are not collected.
0b1	Physical addresses are collected.

When EL2 is implemented, all of the following apply:

- If the Profiling Buffer owning Exception level is EL1, then this field is combined with the Effective value PMSCR_EL2.PA to determine whether Physical addresses are collected.
- If the Profiling Buffer owning Exception level is EL2, then this field is ignored by the PE.

For more information, see Controlling the data that is collected.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CX, bit [3]

CONTEXTIDR_EL1 sample enable. Enables recording of a Context packet containing the value of CONTEXTIDR_EL1.

CX	Meaning
0b0	CONTEXTIDR_EL1 recording disabled.
0b1	CONTEXTIDR_EL1 recording enabled.

The PE ignores the value of this field and CONTEXTIDR_EL1 is not recorded when any of the following apply:

- The sampled operation executes at EL2.
- The sampled operation executes at EL0 and the Effective value of HCR_EL2.TGE is 1.

For more information, see Controlling the data that is collected.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [2]

Reserved, RES0.

E1SPE, bit [1]

EL1 Statistical Profiling Enable.

E1SPE	Meaning
0b0	Sampling disabled at EL1.
0b1	Sampling enabled at EL1.

If the Effective value of HCR_EL2.TGE is 1, then this field is ignored by the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E0SPE, bit [0]

EL0 Statistical Profiling Enable. Controls sampling at EL0 when HCR_EL2.TGE == 0 or if EL2 is disabled or not implemented.

E0SPE	Meaning
0b0	Sampling disabled at EL0.
0b1	Sampling enabled at EL0.

If the Effective value of HCR_EL2.TGE is 1, then this field is ignored by the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMSCR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMSCR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b000

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.PMSCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x828];
    else
        X[t, 64] = PMSCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = PMSCR_EL2;
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        X[t, 64] = PMSCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMSCR_EL1;
```

MSR PMSCR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b000

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.PMSCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x828] = X[t, 64];
    else
        PMSCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        PMSCR_EL2 = X[t, 64];
    else
        PMSCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMSCR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, PMSCR_EL12

op0	op1	CRn	CRm	op2
0b11	0b101	0b1001	0b1001	0b000

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x828];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
            UNDEFINED;
        elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = PMSCR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = PMSCR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented MSR PMSCR_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b1001	0b1001	0b000

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x828] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
            UNDEFINED;
        elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMSCR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        PMSCR_EL1 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.7.9 PMSCR_EL2, Statistical Profiling Control Register (EL2)

The PMSCR_EL2 characteristics are:

Purpose

Provides EL2 controls for Statistical Profiling.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMSCR_EL2 are UNDEFINED.

Attributes

PMSCR_EL2 is a 64-bit register.

Field descriptions

Bits [63:12]

Reserved, RES0.

EnVM, bit [11]

When FEAT_SPE_nVM is implemented:

Enable use of physical address Profiling Buffer pointers.

EnVM	Meaning
0b0	Use of physical address Profiling Buffer pointers is disabled. The PE behaves as if PMBLIMITR_EL1.nVM is 0.
0b1	Use of physical address Profiling Buffer pointers is permitted.

If EL2 is disabled in the owning Security state, or the Profiling Buffer owning Exception level is EL2, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

KE, bit [10]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_SPE_EXC is implemented:

Kernel exception enable for SPE Profiling exceptions taken to EL2.

KE	Meaning
0b0	SPE Profiling exceptions taken to EL2 are always masked at EL2.
0b1	Enabled SPE Profiling exceptions taken to EL2 are masked at EL2 when PSTATE.PM is 1 and unmasked when PSTATE.PM is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EE, bits [9:8]

When FEAT_SPE_EXC is implemented:

Exception Enable.

EE	Meaning
0b00	Disabled. SPE Profiling exceptions for EL2 and EL1 are disabled. All of the following apply: • No Profiling Buffer management events are recorded in PMBSR_EL2. • Unless enabled by a higher Exception level, SPE Profiling exceptions are not generated. • PMBSR_EL1.S drives the interrupt request signal **PMBIRQ**. • Accesses to PMBSR_EL1 at EL1 ignore the value of HCR_EL2.NV1 and accesses to PMBSR_EL1 at EL2 ignore the value of HCR_EL2.E2H.
0b01	Delegated. SPE Profiling exceptions for EL2 are disabled, but might be enabled for EL1 by PMSCR_EL1.EE. All of the following apply: • No Profiling Buffer management events are recorded in PMBSR_EL2. • PMBSR_EL2.S is ignored and SPE Profiling exceptions are not taken to EL2, other than for the case when the Effective value of HCR_EL2.TGE is 1.

0b10	Enabled. SPE Profiling exceptions for EL2 are enabled for Profiling Buffer management events targeting EL2, as follows:
	• Profiling Buffer management events due to a fault on a write to the Profiling Buffer that would generate a Data Abort exception taken to EL2 if generated by a store instruction executed at the owning Exception level are recorded in PMBSR_EL2, unless they are configured to be recorded in PMBSR_EL3 by MDCR_EL3.PMSEE. If the Profiling Buffer owning Exception level is EL2, then this means any fault on a write to the Profiling Buffer. If the Profiling Buffer owning Exception level is EL1, then this means any of the following faults on a write to the Profiling Buffer:
	  • Stage 2 faults.
	  • If HCR_EL2.TEA is 1, synchronous External aborts.
	  • If HCR_EL2.GPF is 1, Granule Protection Faults (GPFs).
	• Profiling Buffer management events due to Granule Protection Check faults other than GPFs on a write to the Profiling Buffer are recorded in PMBSR_EL2, unless they are configured to be recorded in PMBSR_EL3 by MDCR_EL3.PMSEE.
	• SPE Profiling exceptions are generated and taken to EL2 when unmasked and PMBSR_EL2.S is 1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EE	Meaning
0b11	Trap all. SPE Profiling exceptions for EL2 are enabled for all Profiling Buffer management events, as follows: • All Profiling Buffer management events are recorded in PMBSR_EL2, unless they are configured to be recorded in PMBSR_EL3 by MDCR_EL3.PMSEE. • SPE Profiling exceptions are generated and taken to EL2 when unmasked and PMBSR_EL2.S is 1.

If the Effective value of MDCR_EL3.PMSEE is 0b00, then the Effective value of PMSCR_EL2.EE is 0b00. Otherwise, if EL2 is not implemented or the Effective value of SCR_EL3.{NS, EEL2} is {0, 0}, then the Effective value of PMSCR_EL2.EE is 0b01.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '00'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PCT, bits [7:6]

Physical Timestamp. If timestamp sampling is enabled, determines which counter is collected. The behavior depends on the Profiling Buffer owning Exception level.

If FEAT_ECV is implemented, this is a two-bit field as shown. Otherwise, bit[7] is RES0.

PCT	Meaning	Applies when
0b00	Virtual timestamp. The collected timestamp is the physical counter minus a virtual offset. If the Profiling Buffer owning Exception level is EL2 and any of the following are true, then the virtual offset is zero: • The sampled operation executed at EL2 and the Effective value of HCR_EL2.E2H is 1. • The sampled operation executed at EL0 and the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}. Otherwise, the virtual offset is the value of CNTVOFF_EL2.	
0b01	If the Profiling Buffer owning Exception level is EL1, then the timestamp value is selected by PMSCR_EL1.PCT. Otherwise, physical timestamp. The collected timestamp is the physical counter.	
0b11	If the Profiling Buffer owning Exception level is EL1 and PMSCR_EL1.PCT is 0b00, then guest virtual timestamp. The collected timestamp is the physical counter minus the value of CNTVOFF_EL2. Otherwise, guest physical timestamp. The collected timestamp is the physical counter minus a physical offset. If any of the following are true, then the physical offset is zero, otherwise the physical offset is the value of CNTPOFF_EL2: • SCR_EL3.ECVEn is 0. • CNTHCTL_EL2.ECV is 0. • FEAT_ECV_POFF is not implemented.	FEAT_ECV is implemented

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

All other values are reserved.

If EL2 is not implemented or EL2 is disabled in the current Security state, then the Effective value of this field is `0b01`, other than for a direct read of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TS, bit [5]

Timestamp sample enable. Enables recording of a Timestamp packet when the owning Exception level is EL2.

TS	Meaning
`0b0`	Timestamp packet recording disabled.
`0b1`	Timestamp packet recording enabled.

If the Profiling Buffer owning Exception level is EL1, then this field is ignored by the PE. For more information, see Controlling the data that is collected.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PA, bit [4]

Physical Address Sample Enable.

PA	Meaning
`0b0`	Physical addresses are not collected.
`0b1`	Physical addresses are collected.

If the Profiling Buffer owning Exception level is EL2, then this field determines whether physical addresses are collected.

If EL2 is not implemented or EL2 is disabled in the owning Security state, then the Effective value of this field is 1.

For more information, see Controlling the data that is collected.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CX, bit [3]

CONTEXTIDR_EL2 sample enable. Enables recording of a Context packet containing the value of CONTEXTIDR_EL2.

CX	Meaning
`0b0`	CONTEXTIDR_EL2 recording disabled.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CX	Meaning
0b1	CONTEXTIDR_EL2 recording enabled.

The PE ignores the value of this field and CONTEXTIDR_EL2 is not recorded when EL2 is not implemented or EL2 is disabled in the current Security state.

For more information, see Controlling the data that is collected.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [2]

Reserved, RES0.

E2SPE, bit [1]

EL2 Statistical Profiling Enable.

E2SPE	Meaning
0b0	Sampling disabled at EL2.
0b1	Sampling enabled at EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When MDCR_EL2.E2PB != ‘00’, access to this field is RES0.
- Otherwise, access to this field is RW.

E0HSPE, bit [0]

EL0 Statistical Profiling Enable.

E0HSPE	Meaning
0b0	Sampling disabled at EL0.
0b1	Sampling enabled at EL0.

If the Effective value of HCR_EL2.TGE is 0, then this field is ignored by the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When MDCR_EL2.E2PB != ‘00’, access to this field is RES0.
- Otherwise, access to this field is RW.

Accessing PMSCR_EL2

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, PMSCR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1001	0b1001	0b000

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMSCR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMSCR_EL2;
```

```
MSR PMSCR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1001	0b1001	0b000

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSCR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMSCR_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, PMSCR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b000

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.PMSCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x828];
    else
        X[t, 64] = PMSCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = PMSCR_EL2;
    else
        X[t, 64] = PMSCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMSCR_EL1;
```

```
MSR PMSCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b000

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.PMSCR_EL1 == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x828] = X[t, 64];
    else
        PMSCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        PMSCR_EL2 = X[t, 64];
    else
        PMSCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMSCR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.7.10 PMSDSFR_EL1, Sampling Data Source Filter Register

The PMSDSFR_EL1 characteristics are:

Purpose

Controls sample filtering by Data Source.

Configuration

This register is present only when FEAT_SPE_FDS is implemented. Otherwise, direct accesses to PMSDSFR_EL1 are UNDEFINED.

Attributes

PMSDSFR_EL1 is a 64-bit register.

Field descriptions

S<m> , bits [m], for m = 63 to 0

When filtering on Data Source <m> is supported:

S[<m>] is the Data Source filter for IMPLEMENTATION DEFINED Data Source <m>.

S<m>	Meaning
0b0	If PMSFCR_EL1.FDS is 1, do not record load operations that have bits [5:0] of the Data Source packet set to <m>.
0b1	Load operations with Data Source <m> are unaffected by PMSFCR_EL1.FDS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

Accessing PMSDSFR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMSDSFR_EL1
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1010	0b100

```
if !IsFeatureImplemented(FEAT_SPE_FDS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMS3 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGRTR2_EL2.nPMSDSFR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPMS3 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x858];
    else
        X[t, 64] = PMSDSFR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMS3 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPMS3 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMSDSFR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMSDSFR_EL1;
```

```
MSR PMSDSFR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1010	0b100

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_SPE_FDS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMS3 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HDFGWTR2_EL2.nPMSDSFR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPMS3 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x858] = X[t, 64];
    else
        PMSDSFR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMS3 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPMS3 == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSDSFR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMSDSFR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.7.11 PMSEVFR_EL1, Sampling Event Filter Register

The PMSEVFR_EL1 characteristics are:

Purpose

Controls sample filtering by events. The overall filter is the logical AND of these filters. For example, if PMSEVFR_EL1.E[3] and PMSEVFR_EL1.E[5] are both set to 1, only samples that have both event 3 (Level 1 unified or data cache refill) and event 5 (TLB walk) set to 1 are recorded.

Configuration

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMSEVFR_EL1 are UNDEFINED.

Attributes

PMSEVFR_EL1 is a 64-bit register.

Field descriptions

E[63], bit [63]

When event 63 is implemented and filtering on event 63 is supported:

Filter on IMPLEMENTATION DEFINED event 63.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E[63]	Meaning
0b0	Event 63 is ignored.
0b1	Do not record samples that have event 63 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[62], bit [62]

When event 62 is implemented and filtering on event 62 is supported:

Filter on IMPLEMENTATION DEFINED event 62.

E[62]	Meaning
0b0	Event 62 is ignored.
0b1	Do not record samples that have event 62 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[61], bit [61]

When event 61 is implemented and filtering on event 61 is supported:

Filter on IMPLEMENTATION DEFINED event 61.

E[61]	Meaning
0b0	Event 61 is ignored.
0b1	Do not record samples that have event 61 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[60], bit [60]

When event 60 is implemented and filtering on event 60 is supported:

Filter on IMPLEMENTATION DEFINED event 60.

E[60]	Meaning
0b0	Event 60 is ignored.
0b1	Do not record samples that have event 60 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[59], bit [59]

When event 59 is implemented and filtering on event 59 is supported:

Filter on IMPLEMENTATION DEFINED event 59.

E[59]	Meaning
0b0	Event 59 is ignored.
0b1	Do not record samples that have event 59 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[58], bit [58]

When event 58 is implemented and filtering on event 58 is supported:

Filter on IMPLEMENTATION DEFINED event 58.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E[58]	Meaning
0b0	Event 58 is ignored.
0b1	Do not record samples that have event 58 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[57], bit [57]

When event 57 is implemented and filtering on event 57 is supported:

Filter on IMPLEMENTATION DEFINED event 57.

E[57]	Meaning
0b0	Event 57 is ignored.
0b1	Do not record samples that have event 57 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[56], bit [56]

When event 56 is implemented and filtering on event 56 is supported:

Filter on IMPLEMENTATION DEFINED event 56.

E[56]	Meaning
0b0	Event 56 is ignored.
0b1	Do not record samples that have event 56 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[55], bit [55]

When event 55 is implemented and filtering on event 55 is supported:

Filter on IMPLEMENTATION DEFINED event 55.

E[55]	Meaning
`0b0`	Event 55 is ignored.
`0b1`	Do not record samples that have event 55 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[54], bit [54]

When event 54 is implemented and filtering on event 54 is supported:

Filter on IMPLEMENTATION DEFINED event 54.

E[54]	Meaning
`0b0`	Event 54 is ignored.
`0b1`	Do not record samples that have event 54 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[53], bit [53]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When event 53 is implemented and filtering on event 53 is supported:

Filter on IMPLEMENTATION DEFINED event 53.

E[53]	Meaning
0b0	Event 53 is ignored.
0b1	Do not record samples that have event 53 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[52], bit [52]

When event 52 is implemented and filtering on event 52 is supported:

Filter on IMPLEMENTATION DEFINED event 52.

E[52]	Meaning
0b0	Event 52 is ignored.
0b1	Do not record samples that have event 52 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[51], bit [51]

When event 51 is implemented and filtering on event 51 is supported:

Filter on IMPLEMENTATION DEFINED event 51.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E[51]	Meaning
0b0	Event 51 is ignored.
0b1	Do not record samples that have event 51 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[50], bit [50]

When event 50 is implemented and filtering on event 50 is supported:

Filter on IMPLEMENTATION DEFINED event 50.

E[50]	Meaning
0b0	Event 50 is ignored.
0b1	Do not record samples that have event 50 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[49], bit [49]

When event 49 is implemented and filtering on event 49 is supported:

Filter on IMPLEMENTATION DEFINED event 49.

E[49]	Meaning
0b0	Event 49 is ignored.
0b1	Do not record samples that have event 49 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[48], bit [48]

When event 48 is implemented and filtering on event 48 is supported:

Filter on IMPLEMENTATION DEFINED event 48.

E[48]	Meaning
0b0	Event 48 is ignored.
0b1	Do not record samples that have event 48 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

Bits [47:32]

Reserved, RAZ/WI.

E[31], bit [31]

When FEAT_SPEv1p4 is not implemented, event 31 is implemented, and filtering on event 31 is supported:

Filter on IMPLEMENTATION DEFINED event 31.

E[31]	Meaning
0b0	Event 31 is ignored.
0b1	Do not record samples that have event 31 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[30], bit [30]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_SPEv1p4 is not implemented, event 30 is implemented, and filtering on event 30 is supported:

Filter on IMPLEMENTATION DEFINED event 30.

E[30]	Meaning
0b0	Event 30 is ignored.
0b1	Do not record samples that have event 30 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[29], bit [29]

When FEAT_SPEv1p4 is not implemented, event 29 is implemented, and filtering on event 29 is supported:

Filter on IMPLEMENTATION DEFINED event 29.

E[29]	Meaning
0b0	Event 29 is ignored.
0b1	Do not record samples that have event 29 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[28], bit [28]

When FEAT_SPEv1p4 is not implemented, event 28 is implemented, and filtering on event 28 is supported:

Filter on IMPLEMENTATION DEFINED event 28.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E[28]	Meaning
0b0	Event 28 is ignored.
0b1	Do not record samples that have event 28 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[27], bit [27]

When FEAT_SPEv1p4 is not implemented, event 27 is implemented, and filtering on event 27 is supported:

Filter on IMPLEMENTATION DEFINED event 27.

E[27]	Meaning
0b0	Event 27 is ignored.
0b1	Do not record samples that have event 27 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[26], bit [26]

When FEAT_SPEv1p4 is not implemented, event 26 is implemented, and filtering on event 26 is supported:

Filter on IMPLEMENTATION DEFINED event 26.

E[26]	Meaning
0b0	Event 26 is ignored.
0b1	Do not record samples that have event 26 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[25], bit [25]

When (FEAT_SPE_SME is implemented or FEAT_SPEv1p5 is implemented) and event 25 is implemented:

Filter on SMCU or other shared resource operation event.

E[25]	Meaning
0b0	SMCU or other shared resource operation event is ignored.
0b1	Do not record samples that have the SMCU or other shared resource operation event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_SPEv1p4 is not implemented, event 25 is implemented, and filtering on event 25 is supported:

Filter on IMPLEMENTATION DEFINED event 25.

E[25]	Meaning
0b0	Event 25 is ignored.
0b1	Do not record samples that have event 25 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[24], bit [24]

When FEAT_SPE_SME is implemented:

Filter on Streaming SVE mode event.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E[24]	Meaning
0b0	Streaming SVE mode event is ignored.
0b1	Do not record samples that have the Streaming SVE mode event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_SPEv1p4 is not implemented, event 24 is implemented, and filtering on event 24 is supported:

Filter on IMPLEMENTATION DEFINED event 24.

E[24]	Meaning
0b0	Event 24 is ignored.
0b1	Do not record samples that have event 24 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[23], bit [23]

When FEAT_SPEv1p4 is implemented and event 23 is implemented:

Filter on Data snooped event.

E[23]	Meaning
0b0	Data snooped event is ignored.
0b1	Do not record samples that have the Data snooped event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[22], bit [22]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_SPEv1p4 is implemented and event 22 is implemented:

Filter on Recently fetched event.

E[22]	Meaning
0b0	Recently fetched event is ignored.
0b1	Do not record samples that have the Recently fetched event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[21], bit [21]

When FEAT_SPEv1p4 is implemented and event 21 is implemented:

Filter on Cache data modified event.

E[21]	Meaning
0b0	Cache data modified event is ignored.
0b1	Do not record samples that have the Cache data modified event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[20], bit [20]

When FEAT_SPEv1p4 is implemented and event 20 is implemented:

Filter on Level 2 data cache miss event.

E[20]	Meaning
0b0	Level 2 data cache miss event is ignored.
0b1	Do not record samples that have the Level 2 data cache miss event == 0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[19], bit [19]

When FEAT_SPEv1p4 is implemented and event 19 is implemented:

Filter on Level 2 data cache access event.

E[19]	Meaning
`0b0`	Level 2 data cache access event is ignored.
`0b1`	Do not record samples that have the Level 2 data cache access event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[18], bit [18]

When FEAT_SPEv1p1 is implemented and (FEAT_SVE is implemented or FEAT_SME is implemented):

Filter on Empty predicate event.

E[18]	Meaning
`0b0`	Empty predicate event is ignored.
`0b1`	Do not record samples that have the Empty predicate event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[17], bit [17]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_SPEv1p1 is implemented and (FEAT_SVE is implemented or FEAT_SME is implemented):

Filter on Partial or empty predicate event.

E[17]	Meaning
`0b0`	Partial or empty predicate event is ignored.
`0b1`	Do not record samples that have the Partial or empty predicate event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

Bit [16]

Reserved, RAZ/WI.

E[15], bit [15]

When event 15 is implemented and filtering on event 15 is supported:

Filter on IMPLEMENTATION DEFINED event 15.

E[15]	Meaning
`0b0`	Event 15 is ignored.
`0b1`	Do not record samples that have event 15 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[14], bit [14]

When event 14 is implemented and filtering on event 14 is supported:

Filter on IMPLEMENTATION DEFINED event 14.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E[14]	Meaning
0b0	Event 14 is ignored.
0b1	Do not record samples that have event 14 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[13], bit [13]

When event 13 is implemented and filtering on event 13 is supported:

Filter on IMPLEMENTATION DEFINED event 13.

E[13]	Meaning
0b0	Event 13 is ignored.
0b1	Do not record samples that have event 13 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[12], bit [12]

When event 12 is implemented and filtering on event 12 is supported:

Filter on IMPLEMENTATION DEFINED event 12.

E[12]	Meaning
0b0	Event 12 is ignored.
0b1	Do not record samples that have event 12 == 0.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE is 0

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[11], bit [11]

When FEAT_SPEv1p1 is implemented:

Filter on Misalignment event.

E[11]	Meaning
0b0	Misalignment event is ignored.
0b1	Do not record samples that have the Misalignment event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[10], bit [10]

When (FEAT_SPEv1p4 is implemented or filtering on event 10 is optionally supported) and event 10 is implemented:

Filter on Remote access event.

E[10]	Meaning
0b0	Remote access event is ignored.
0b1	Do not record samples that have the Remote access event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[9], bit [9]

When (FEAT_SPEv1p4 is implemented or filtering on event 9 is optionally supported) and event 9 is implemented:

Filter on Last Level cache miss event.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E[9]	Meaning
0b0	Last Level cache miss event is ignored.
0b1	Do not record samples that have the Last Level cache miss event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[8], bit [8]

When (FEAT_SPEv1p4 is implemented or filtering on event 8 is optionally supported) and event 8 is implemented:

Filter on Last Level cache access event.

E[8]	Meaning
0b0	Last Level cache access event is ignored.
0b1	Do not record samples that have the Last Level cache access event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[7], bit [7]

Filter on Mispredicted event.

E[7]	Meaning
0b0	Mispredicted event is ignored.
0b1	Do not record samples that have the Mispredicted event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E[6], bit [6]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_SPE_FnE is implemented:

Filter on Not taken event.

E[6]	Meaning
0b0	Not taken event is ignored.
0b1	Do not record samples that have the Not taken event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[5], bit [5]

Filter on TLB walk event.

E[5]	Meaning
0b0	TLB walk event is ignored.
0b1	Do not record samples that have the TLB walk event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E[4], bit [4]

When FEAT_SPEv1p4 is implemented or filtering on event 4 is optionally supported:

Filter on TLB access event.

E[4]	Meaning
0b0	TLB access event is ignored.
0b1	Do not record samples that have the TLB access event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RAZ/WI.

E[3], bit [3]

Filter on Level 1 data cache refill or miss event.

E[3]	Meaning
0b0	Level 1 data cache refill or miss event is ignored.
0b1	Do not record samples that have the Level 1 data cache refill or miss event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E[2], bit [2]

When FEAT_SPEv1p4 is implemented or filtering on event 2 is optionally supported:

Filter on Level 1 data cache access event.

E[2]	Meaning
0b0	Level 1 data cache access event is ignored.
0b1	Do not record samples that have the Level 1 data cache access event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[1], bit [1]

When the PE supports sampling of speculative instructions:

Filter on Architecturally retired event.

E[1]	Meaning
0b0	Architecturally retired event is ignored.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

0b1	Do not record samples that have the Architecturally retired event == 0.

This field is ignored by the PE when PMSFCR_EL1.FE is 0.

If the PE does not support the sampling of speculative instructions, or always discards the sample record for speculative instructions, this bit reads as an UNKNOWN value and the PE ignores its value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, UNKNOWN.

Bit [0]

Reserved, RAZ/WI.

Accessing PMSEVFR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMSEVFR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b101

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.PMSEVFR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x830];
    else
        X[t, 64] = PMSEVFR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        X[t, 64] = PMSEVFR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMSEVFR_EL1;
```

MSR PMSEVFR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b101

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.PMSEVFR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x830] = X[t, 64];
    else
        PMSEVFR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSEVFR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMSEVFR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.7.12 PMSFCR_EL1, Sampling Filter Control Register

The PMSFCR_EL1 characteristics are:

Purpose

Controls sample filtering. The filter is the logical AND of the filters controlled by FDS, FnE, FL, FT, and FE bits. For example, if PMSFCR_EL1.FE is 1 and PMSFCR_EL1.FT is 1, then only samples including the selected types and the selected events will be recorded.

Configuration

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMSFCR_EL1 are UNDEFINED.

Attributes

PMSFCR_EL1 is a 64-bit register.

Field descriptions

Bits [63:53]

Reserved, RES0.

TYPEm, bits [52:48]

When FEAT_SPE_EFT is implemented:

SIMDm, bit [4]

SIMD filter mask.

SIMDm	Meaning
0b0	PMSFCR_EL1.SIMD controls whether SIMD operations are recorded as part of a Boolean-OR filter with other masked operation type filter controls.
0b1	PMSFCR_EL1.SIMD controls whether SIMD operations are recorded as part of a Boolean-AND filter with other unmasked operation type filter controls.

This field is ignored by the PE when PMSFCR_EL1.FT is 0.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FPm, bit [3]

Floating-point filter mask.

FPm	Meaning
0b0	PMSFCR_EL1.FP controls whether floating-point operations are recorded as part of a Boolean-OR filter with other masked operation type filter controls.
0b1	PMSFCR_EL1.FP controls whether floating-point operations are recorded as part of a Boolean-AND filter with other unmasked operation type filter controls.

This field is ignored by the PE when PMSFCR_EL1.FT is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

STm, bit [2]

Store filter mask.

STm	Meaning
0b0	PMSFCR_EL1.ST controls whether store operations are recorded as part of a Boolean-OR filter with other masked operation type filter controls.
0b1	PMSFCR_EL1.ST controls whether store operations are recorded as part of a Boolean-AND filter with other unmasked operation type filter controls.

This field is ignored by the PE when PMSFCR_EL1.FT is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

LDm, bit [1]

Load filter mask.

LDm	Meaning
0b0	PMSFCR_EL1.LD controls whether load operations are recorded as part of a Boolean-OR filter with other masked operation type filter controls.
0b1	PMSFCR_EL1.LD controls whether load operations are recorded as part of a Boolean-AND filter with other unmasked operation type filter controls.

This field is ignored by the PE when PMSFCR_EL1.FT is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bm, bit [0]

Branch filter mask.

Bm	Meaning
0b0	PMSFCR_EL1.B controls whether branch operations are recorded as part of a Boolean-OR filter with other masked operation type filter controls.
0b1	PMSFCR_EL1.B controls whether branch operations are recorded as part of a Boolean-AND filter with other unmasked operation type filter controls.

This field is ignored by the PE when PMSFCR_EL1.FT is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_SPE_EFT is not implemented:

Bits [4:0]

Reserved, RES0.

Bits [47:21]

Reserved, RES0.

TYPE, bits [20:16]

SIMD, bit [4]

When FEAT_SPE_EFT is implemented:

SIMD filter enable.

SIMD	Meaning
0b0	If PMSFCR_EL1.SIMDm is 1, then record only operations that are not SIMD operations. Otherwise, do not record SIMD operations, unless allowed by another operation type filter.
0b1	If PMSFCR_EL1.SIMDm is 1, then record only operations that are SIMD operations. Otherwise, record all SIMD operations.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

This field is ignored by the PE and no records are removed by this filter when any of the following apply:

- PMSFCR_EL1.FT is 0.
- PMSFCR_EL1.SIMDm is 0 and the values of the PMSFCR_EL1.{SIMD, FP, ST, LD, B} bits for which the corresponding PMSFCR_EL1.{SIMDm, FPm, STm, LDm, Bm} bit is zero are all zero.

For filtering purposes, SIMD operations means all Advanced SIMD, SVE, and SME SIMD operations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FP, bit [3]

When FEAT_SPE_EFT is implemented:

Floating-point filter enable.

FP	Meaning
0b0	If PMSFCR_EL1.FPm is 1, then record only operations that are not floating-point operations. Otherwise, do not record floating-point operations, unless allowed by another operation type filter.
0b1	If PMSFCR_EL1.FPm is 1, then record only operations that are floating-point operations. Otherwise, record all floating-point operations.

This field is ignored by the PE and no records are removed by this filter when any of the following apply:

- PMSFCR_EL1.FT is 0.
- PMSFCR_EL1.FPm is 0 and the values of the PMSFCR_EL1.{SIMD, FP, ST, LD, B} bits for which the corresponding PMSFCR_EL1.{SIMDm, FPm, STm, LDm, Bm} bit is zero are all zero.

For filtering purposes, floating-point operations means all scalar, Advanced SIMD, SVE, and SME floating-point operations, as defined by the FP_SPEC event.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ST, bit [2]

Store filter enable.

ST	Meaning
0b0	If FEAT_SPE_EFT is implemented and PMSFCR_EL1.STm is 1, then record only operations that are not store operations. Otherwise, do not record store operations, unless allowed by another operation type filter.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ST	Meaning
0b1	If FEAT_SPE_EFT is implemented and PMSFCR_EL1.STm is 1, then record only operations that are store operations. Otherwise, record all store operations.

This field is ignored by the PE and no records are removed by this filter when any of the following apply:

- PMSFCR_EL1.FT is 0.
- FEAT_SPE_EFT is implemented, PMSFCR_EL1.STm is 0, and the values of the PMSFCR_EL1.{SIMD, FP, ST, LD, B} bits for which the corresponding PMSFCR_EL1.{SIMDm, FPm, STm, LDm, Bm} bit is zero are all zero.

For filtering purposes, store operations includes vector stores and all atomic operations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

LD, bit [1]

Load filter enable.

LD	Meaning
0b0	If FEAT_SPE_EFT is implemented and PMSFCR_EL1.LDm is 1, then record only operations that are not load operations. Otherwise, do not record load operations, unless allowed by another operation type filter.
0b1	If FEAT_SPE_EFT is implemented and PMSFCR_EL1.LDm is 1, then record only operations that are load operations. Otherwise, record all load operations.

This field is ignored by the PE and no records are removed by this filter when any of the following apply:

- PMSFCR_EL1.FT is 0.
- FEAT_SPE_EFT is implemented, PMSFCR_EL1.LDm is 0, and the values of the PMSFCR_EL1.{SIMD, FP, ST, LD, B} bits for which the corresponding PMSFCR_EL1.{SIMDm, FPm, STm, LDm, Bm} bit is zero are all zero.

For filtering purposes, load operations includes vector loads and atomic operations that return a value to the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

B, bit [0]

Branch filter enable.

B	Meaning
0b0	If FEAT_SPE_EFT is implemented and PMSFCR_EL1.Bm is 1, then record only operations that are not branch operations. Otherwise, do not record branch operations.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

B	Meaning
0b1	If FEAT_SPE_EFT is implemented and PMSFCR_EL1.Bm is 1, then record only operations that are branch operations. Otherwise, record all branch operations.

This field is ignored by the PE and no records are removed by this filter when any of the following apply:

- PMSFCR_EL1.FT is 0.
- FEAT_SPE_EFT is implemented, PMSFCR_EL1.Bm is 0, and the values of the PMSFCR_EL1.{SIMD, FP, ST, LD, B} bits for which the corresponding PMSFCR_EL1.{SIMDm, FPm, STm, LDm, Bm} bit is zero are all zero.

For filtering purposes, branch operations includes exception returns.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [15:5]

Reserved, RES0.

FDS, bit [4]

When FEAT_SPE_FDS is implemented:

Filter by Data Source.

FDS	Meaning
0b0	Data Source filtering disabled.
0b1	Data Source filtering enabled. Samples of load instructions reporting a Data Source not selected by PMSDSFR_EL1 will not be recorded.

If PMSFCR_EL1.FDS is 1 and PMSDSFR_EL1 is zero, then no load operations with a Data Source will be recorded.

Load operations without a Data Source and other sampled operations are unaffected by this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FnE, bit [3]

When FEAT_SPE_FnE is implemented:

Filter by event, inverted.

FnE	Meaning
0b0	Inverted event filtering disabled.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FnE	Meaning
0b1	Inverted event filtering enabled. Samples including the events selected by PMSNEVFR_EL1 will not be recorded.

If any of the following are true, then it is CONSTRAINED UNPREDICTABLE whether no samples are recorded or the PE behaves as if PMSFCR_EL1.FnE is 0:

- PMSFCR_EL1.FnE is 1 and PMSNEVFR_EL1 is zero.
- PMSFCR_EL1.FnE is 1, PMSFCR_EL1.FE is 1, and there exists a value x such that PMSEVFR_EL1.E[x] is 1 and PMSNEVFR_EL1.E[x] is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FL, bit [2]

Filter by latency.

FL	Meaning
0b0	Latency filtering disabled.
0b1	Latency filtering enabled. Samples with a total latency less than PMSLATFR_EL1.MINLAT will not be recorded.

If this field is 1 and PMSLATFR_EL1.MINLAT is zero, then it is CONSTRAINED UNPREDICTABLE whether no samples are recorded or the PE behaves as if PMSFCR_EL1.FL is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FT, bit [1]

Filter by type. The filter is controlled by the TYPE and, if FEAT_SPE_EFT is implemented, TYPEm fields.

FT	Meaning
0b0	Type filtering disabled.
0b1	Type filtering enabled. Samples not one of the selected types will not be recorded.

Type filtering filters according to the type of the sampled operation.

If FEAT_SPE_EFT is not implemented, this field is 1, and the PMSFCR_EL1.{ST, LD, B} bits are all zero, then it is CONSTRAINED UNPREDICTABLE whether no samples are recorded or the PE behaves as if PMSFCR_EL1.FT is 0.

For more information, see Filtering by operation type.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FE, bit [0]

Filter by event.

FE	Meaning
0b0	Event filtering disabled.
0b1	Event filtering enabled. Samples not including the events selected by PMSEVFR_EL1 will not be recorded.

If any of the following are true, then it is CONSTRAINED UNPREDICTABLE whether no samples are recorded or the PE behaves as if PMSFCR_EL1.FE is 0:

- PMSFCR_EL1.FE is 1 and PMSEVFR_EL1 is zero.
- FEAT_SPE_FnE is implemented, PMSFCR_EL1.FnE is 1, PMSFCR_EL1.FE is 1, and there exists a value x such that PMSEVFR_EL1.E[x] is 1 and PMSNEVFR_EL1.E[x] is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMSFCR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMSFCR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b100

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.PMSFCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMSFCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMSFCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMSFCR_EL1;
```

MSR PMSFCR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b100

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.PMSFCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSFCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSFCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMSFCR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.7.13 PMSICR_EL1, Sampling Interval Counter Register

The PMSICR_EL1 characteristics are:

Purpose

Software must write zero to PMSICR_EL1 before enabling sample profiling for a sampling session. Software must then treat PMSICR_EL1 as an opaque, 64-bit, read/write register used for context switches only.

Configuration

The value of PMSICR_EL1 does not change whilst profiling is disabled.

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMSICR_EL1 are UNDEFINED.

Attributes

PMSICR_EL1 is a 64-bit register.

Field descriptions

ECOUNT, bits [63:56]

When FEAT_SPE_ERnd is implemented:

Secondary sample interval counter. Provides the secondary counter used after the primary counter reaches zero.

While the secondary counter is nonzero and profiling is enabled, the secondary counter decrements by 1 for each member of the sample population. The primary counter also continues to decrement since it is also nonzero. When the secondary counter reaches zero, a member of the sampling population is selected for sampling.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [55:32]

Reserved, RES0.

COUNT, bits [31:0]

Primary sample interval counter. Provides the primary counter used for sampling.

When the PE moves from a state or Exception level where profiling is disabled to a state or Exception level where profiling is enabled, if the value of this register is zero, then the primary counter is loaded from PMSIRR_EL1.

While the primary counter is nonzero and sampling is enabled, the primary counter decrements by 1 for each member of the sample population.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The sample interval counter counts either a number of operations or a number of instructions, depending on the IMPLEMENTATION DEFINED value of PMSIDR_EL1.ArchInst.

When the counter reaches zero, the behavior depends on the value of PMSIRR_EL1.RND and whether FEAT_SPE_ERnd is implemented:

- If PMSIRR_EL1.RND is 1 and FEAT_SPE_ERnd is implemented, then the secondary counter is set to a random or pseudorandom value in the range `0x00` to `0xFF`.
- Otherwise, a member of the sampling population is selected for sampling.
- The primary counter is loaded from PMSIRR_EL1.

The primary counter is loaded from PMSIRR_EL1 means:

- PMSICR_EL1.COUNT[31:8] is set to the value of PMSIRR_EL1.INTERVAL.
- If PMSIRR_EL1.RND is 1 and FEAT_SPE_ERnd is not implemented, then PMSICR_EL1.COUNT[7:0] is set to a random or pseudorandom value in the range `0x00` to `0xFF`.
- Otherwise, PMSICR_EL1.COUNT[7:0] is set to `0x00`.

For more information, see Initializing the sample interval counters and Behavior of the sample interval counter while profiling is enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMSICR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMSICR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b010

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.PMSICR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x838];
    else
        X[t, 64] = PMSICR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMSICR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMSICR_EL1;
```

MSR PMSICR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b010

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.PMSICR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x838] = X[t, 64];
    else
        PMSICR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSICR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMSICR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.7.14 PMSIDR_EL1, Sampling Profiling ID Register

The PMSIDR_EL1 characteristics are:

Purpose

Describes the Statistical Profiling implementation to software.

Configuration

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMSIDR_EL1 are UNDEFINED.

Attributes

PMSIDR_EL1 is a 64-bit register.

Field descriptions

Bits [63:33]

Reserved, RES0.

SME, bit [32]

SPE for SME. Describes support for the Scalable Matrix Extensions to Statistical Profiling.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SME	Meaning
0b0	The SME extensions to the Statistical Profiling Extension are not implemented.
0b1	Adds support for Statistical Profiling of the Scalable Matrix Extensions.

FEAT_SPE_SME implements the functionality identified by the value 1.

Access to this field is RO.

ALTCLK, bits [31:28]

Alternate clock domain.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ALTCLK	Meaning
0b0000	Alternate clock domain not implemented, or CPU clock domain.
0b0001	The external Streaming Mode Compute Unit provides the alternate clock domain.
0b1111	IMPLEMENTATION DEFINED clock domain.

All other values are reserved.

FEAT_SPE_ALTCLK implements the functionality identified by a nonzero value.

Access to this field is RO.

FPF, bit [27]

Floating-point flag. Describes whether Operation Type packets for sampled scalar and SIMD operations contain floating-point and Advanced SIMD indications.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FPF	Meaning
0b0	Operation Type packets for scalar and Advanced SIMD operations do not contain floating-point or Advanced SIMD information.
0b1	Operation Type packets for scalar and Advanced SIMD operations contain floating-point or Advanced SIMD information.

FEAT_SPE_FPF implements the functionality identified by the value 1.

Access to this field is RO.

EFT, bit [26]

Extended filtering by type. Describes whether Extended filtering by operation type is supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

EFT	Meaning
0b0	PMSFCR_EL1.{SIMDm, FPm, STm, Bm, LDm, SIMD, FP} are RES0.
0b1	PMSFCR_EL1.{SIMDm, FPm, STm, Bm, LDm, SIMD, FP} are implemented.

FEAT_SPE_EFT implements the functionality identified by the value 1.

Access to this field is RO.

CRR, bit [25]

Call Return branch records. Describes whether the Operation Type packets for sampled branch operations contain Call Return information.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CRR	Meaning
0b0	Operation Type packets for branch operations do not contain Call Return information.
0b1	Operation Type packets for branch operations contain Call Return information.

FEAT_SPE_CRR implements the functionality identified by the value 1.

Access to this field is RO.

PBT, bit [24]

Previous branch target Address packet. Describes whether the Previous branch target Address packet is supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

PBT	Meaning
0b0	Previous branch target Address packet not implemented.
0b1	Previous branch target Address packet implemented.

FEAT_SPE_PBT implements the OPTIONAL functionality identified by the value 1.

Access to this field is RO.

Format, bits [23:20]

Defines the format of the sample records.

Format	Meaning
0b0000	Format 0.

All other values are reserved.

Access to this field is RO.

CountSize, bits [19:16]

Defines the size of the counters.

The value of this field is an IMPLEMENTATION DEFINED choice of:

CountSize	Meaning
0b0010	12-bit saturating counters.
0b0011	16-bit saturating counters.

All other values are reserved.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

MaxSize, bits [15:12]

Defines the largest size for a single record, rounded up to a power-of-two. If this is the same as the minimum alignment (PMBIDR_EL1.Align), then each record is exactly this size.

The value of this field is an IMPLEMENTATION DEFINED choice of:

MaxSize	Meaning
`0b0100`	16 bytes.
`0b0101`	32 bytes.
`0b0110`	64 bytes.
`0b0111`	128 bytes.
`0b1000`	256 bytes.
`0b1001`	512 bytes.
`0b1010`	1KB.
`0b1011`	2KB.

All other values are reserved.

The values `0b0100` and `0b0101` are not permitted for an implementation.

Access to this field is RO.

Interval, bits [11:8]

Minimum sampling interval. This provides guidance from the implementer to the smallest sampling interval.

The interval is defined as a number of either operations or instructions, depending on the IMPLEMENTATION DEFINED value of PMSIDR_EL1.ArchInst.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Interval	Meaning
`0b0000`	256 operations or instructions, or no minimum specified.
`0b0010`	512 operations or instructions.
`0b0011`	768 operations or instructions.
`0b0100`	1,024 operations or instructions.
`0b0101`	1,536 operations or instructions.
`0b0110`	2,048 operations or instructions.
`0b0111`	3,072 operations or instructions.
`0b1000`	4,096 operations or instructions.

All other values are reserved.

If this field reads as a nonzero value, then setting the sample interval to a value less than the indicated value is likely to cause a statistically significant number of sample collisions. For example, the smallest interval might indicate the typical size of the speculation window for the implementation. That is, the number of operations or instructions that will be executing in parallel, when the processor implementation is not heavily loaded.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Setting the sample interval to a value greater than or equal to the indicated value does not guarantee a statistically insignificant number of sample collisions, as this is typically dependent on the program being profiled. For example, if the program has a large number of long latency loads due to cache misses, then this might cause more collisions when these operations or instructions are sampled.

For more information, see 'Sample collisions'.

Access to this field is RO.

FDS, bit [7]

When FEAT_SPEv1p4 is implemented:

Filtering by data source.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FDS	Meaning
0b0	PMSDSFR_EL1 is not implemented and PMSFCR_EL1.FDS is RES0.
0b1	PMSDSFR_EL1 and PMSFCR_EL1.FDS are implemented.

FEAT_SPE_FDS implements the functionality identified by the value 1.

From Armv8.9, if FEAT_SPE_LDS is implemented, the value 0 is not permitted.

Access to this field is RO.

Otherwise:

Reserved, RES0.

FnE, bit [6]

When FEAT_SPE_FnE is implemented:

Filtering by events, inverted.

The value of this field is an IMPLEMENTATION DEFINED choice of:

FnE	Meaning
0b0	PMSNEVFR_EL1 is not implemented and PMSFCR_EL1.FnE is RES0.
0b1	PMSNEVFR_EL1 and PMSFCR_EL1.FnE are implemented.

FEAT_SPE_FnE implements the functionality identified by the value 1.

From Armv8.7, if FEAT_SPE is implemented, the value 0 is not permitted.

Access to this field is RO.

Otherwise:

Reserved, RES0.

ERnd, bit [5]

Defines how the random number generator is used in determining the interval between samples, when enabled by PMSIRR_EL1.RND.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The value of this field is an IMPLEMENTATION DEFINED choice of:

ERnd	Meaning
0b0	The random number is added at the start of the interval, and the sample is taken and a new interval started when the combined interval expires.
0b1	The random number is added and the new interval started after the interval programmed in PMSIRR_EL1.INTERVAL expires, and the sample is taken when the random interval expires.

FEAT_SPE_ERnd implements the functionality identified by the value 1.

Access to this field is RO.

LDS, bit [4]

Data source indicator for sampled load instructions.

The value of this field is an IMPLEMENTATION DEFINED choice of:

LDS	Meaning
0b0	Loaded data source not implemented.
0b1	Loaded data source implemented.

FEAT_SPE_LDS implements the functionality identified by the value 1.

Access to this field is RO.

ArchInst, bit [3]

Architectural instruction profiling. Defines the subject of the profiling population.

The value of this field is an IMPLEMENTATION DEFINED choice of:

ArchInst	Meaning
0b0	Micro-op sampling implemented.
0b1	Architecture instruction sampling implemented.

FEAT_SPE_ArchInst implements the functionality identified by the value 1.

Access to this field is RO.

FL, bit [2]

Filtering by latency.

Reads as 0b1

Access to this field is RO.

FT, bit [1]

Filtering by operation type.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Reads as `0b1`

Access to this field is RO.

FE, bit [0]

Filtering by events.

Reads as `0b1`

Access to this field is RO.

Accessing PMSIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMSIDR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b111

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.PMSIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMSIDR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMSIDR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMSIDR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.7.15 PMSIRR_EL1, Sampling Interval Reload Register

The PMSIRR_EL1 characteristics are:

Purpose

Defines the interval between samples.

Configuration

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMSIRR_EL1 are UNDEFINED.

Attributes

PMSIRR_EL1 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

INTERVAL, bits [31:8]

Bits [31:8] of the PMSICR_EL1 interval counter reload value. Software must set this to a nonzero value.

If this field is zero, then an UNKNOWN sampling interval is used.

If PMSIDR_EL1.Interval is nonzero, then it provides guidance from the implementer to the smallest sampling interval that should be used.

Setting this field to a nonzero value smaller than the indicated value is likely to cause a statistically significant number of sample collisions. See Sample collisions. Setting the sample interval to a value greater than or equal to the indicated value does not guarantee a statistically insignificant number of sample collisions, as this is typically dependent on the program being profiled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [7:1]

Reserved, RES0.

RND, bit [0]

Controls randomization of the sampling interval.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

RND	Meaning
0b0	Disable randomization of sampling interval.
0b1	Add (pseudo-)random jitter to sampling interval.

The random number generator is not architected.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMSIRR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMSIRR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b011

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.PMSIRR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x840];
    else
        X[t, 64] = PMSIRR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMSIRR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMSIRR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MSR PMSIRR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b011

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.PMSIRR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x840] = X[t, 64];
    else
        PMSIRR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSIRR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMSIRR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.7.16 PMSLATFR_EL1, Sampling Latency Filter Register

The PMSLATFR_EL1 characteristics are:

Purpose

Controls sample filtering by latency

Configuration

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMSLATFR_EL1 are UNDEFINED.

Attributes

PMSLATFR_EL1 is a 64-bit register.

Field descriptions

Bits [63:16]

Reserved, RES0.

MINLAT, bits [15:0]

Minimum latency.

When PMSFCR_EL1.FL is 1, defines the value used for filtering by latency. Samples with a Total latency less than PMSLATFR_EL1.MINLAT are not recorded.

If PMSIDR_EL1.CountSize is `0b0010`, PMSLATFR_EL1.MINLAT[15:12] is RES0.

This field is ignored by the PE when PMSFCR_EL1.FL is 0.

For more information, see Filtering by latency.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMSLATFR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMSLATFR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b110

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.PMSLATFR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x848];
    else
        X[t, 64] = PMSLATFR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMSLATFR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMSLATFR_EL1;
```

MSR PMSLATFR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b110

```
if !IsFeatureImplemented(FEAT_SPE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.PMSLATFR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x848] = X[t, 64];
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        PMSLATFR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSLATFR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMSLATFR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.7.17 PMSNEVFR_EL1, Sampling Inverted Event Filter Register

The PMSNEVFR_EL1 characteristics are:

Purpose

Controls sample filtering by events. The overall inverted filter is the logical OR of these filters. For example, if PMSNEVFR_EL1.E[3] and PMSNEVFR_EL1.E[5] are both set to 1, samples that have either event 3 (Level 1 unified or data cache refill) or event 5 (TLB walk) set to 1 are not recorded.

Configuration

This register is present only when FEAT_SPE_FnE is implemented. Otherwise, direct accesses to PMSNEVFR_EL1 are UNDEFINED.

Attributes

PMSNEVFR_EL1 is a 64-bit register.

Field descriptions

E[63], bit [63]

When event 63 is implemented and filtering on event 63 is supported:

Filter on IMPLEMENTATION DEFINED event 63.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E[63]	Meaning
0b0	Event 63 is ignored.
0b1	Do not record samples that have event 63 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[62], bit [62]

When event 62 is implemented and filtering on event 62 is supported:

Filter on IMPLEMENTATION DEFINED event 62.

E[62]	Meaning
0b0	Event 62 is ignored.
0b1	Do not record samples that have event 62 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[61], bit [61]

When event 61 is implemented and filtering on event 61 is supported:

Filter on IMPLEMENTATION DEFINED event 61.

E[61]	Meaning
0b0	Event 61 is ignored.
0b1	Do not record samples that have event 61 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[60], bit [60]

When event 60 is implemented and filtering on event 60 is supported:

Filter on IMPLEMENTATION DEFINED event 60.

E[60]	Meaning
0b0	Event 60 is ignored.
0b1	Do not record samples that have event 60 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[59], bit [59]

When event 59 is implemented and filtering on event 59 is supported:

Filter on IMPLEMENTATION DEFINED event 59.

E[59]	Meaning
0b0	Event 59 is ignored.
0b1	Do not record samples that have event 59 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[58], bit [58]

When event 58 is implemented and filtering on event 58 is supported:

Filter on IMPLEMENTATION DEFINED event 58.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E[58]	Meaning
0b0	Event 58 is ignored.
0b1	Do not record samples that have event 58 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[57], bit [57]

When event 57 is implemented and filtering on event 57 is supported:

Filter on IMPLEMENTATION DEFINED event 57.

E[57]	Meaning
0b0	Event 57 is ignored.
0b1	Do not record samples that have event 57 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[56], bit [56]

When event 56 is implemented and filtering on event 56 is supported:

Filter on IMPLEMENTATION DEFINED event 56.

E[56]	Meaning
0b0	Event 56 is ignored.
0b1	Do not record samples that have event 56 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[55], bit [55]

When event 55 is implemented and filtering on event 55 is supported:

Filter on IMPLEMENTATION DEFINED event 55.

E[55]	Meaning
0b0	Event 55 is ignored.
0b1	Do not record samples that have event 55 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[54], bit [54]

When event 54 is implemented and filtering on event 54 is supported:

Filter on IMPLEMENTATION DEFINED event 54.

E[54]	Meaning
0b0	Event 54 is ignored.
0b1	Do not record samples that have event 54 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[53], bit [53]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When event 53 is implemented and filtering on event 53 is supported:

Filter on IMPLEMENTATION DEFINED event 53.

E[53]	Meaning
0b0	Event 53 is ignored.
0b1	Do not record samples that have event 53 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[52], bit [52]

When event 52 is implemented and filtering on event 52 is supported:

Filter on IMPLEMENTATION DEFINED event 52.

E[52]	Meaning
0b0	Event 52 is ignored.
0b1	Do not record samples that have event 52 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[51], bit [51]

When event 51 is implemented and filtering on event 51 is supported:

Filter on IMPLEMENTATION DEFINED event 51.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E[51]	Meaning
0b0	Event 51 is ignored.
0b1	Do not record samples that have event 51 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[50], bit [50]

When event 50 is implemented and filtering on event 50 is supported:

Filter on IMPLEMENTATION DEFINED event 50.

E[50]	Meaning
0b0	Event 50 is ignored.
0b1	Do not record samples that have event 50 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[49], bit [49]

When event 49 is implemented and filtering on event 49 is supported:

Filter on IMPLEMENTATION DEFINED event 49.

E[49]	Meaning
0b0	Event 49 is ignored.
0b1	Do not record samples that have event 49 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[48], bit [48]

When event 48 is implemented and filtering on event 48 is supported:

Filter on IMPLEMENTATION DEFINED event 48.

E[48]	Meaning
0b0	Event 48 is ignored.
0b1	Do not record samples that have event 48 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

Bits [47:32]

Reserved, RAZ/WI.

E[31], bit [31]

When FEAT_SPEv1p4 is not implemented, event 31 is implemented, and filtering on event 31 is supported:

Filter on IMPLEMENTATION DEFINED event 31.

E[31]	Meaning
0b0	Event 31 is ignored.
0b1	Do not record samples that have event 31 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[30], bit [30]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_SPEv1p4 is not implemented, event 30 is implemented, and filtering on event 30 is supported:

Filter on IMPLEMENTATION DEFINED event 30.

E[30]	Meaning
0b0	Event 30 is ignored.
0b1	Do not record samples that have event 30 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[29], bit [29]

When FEAT_SPEv1p4 is not implemented, event 29 is implemented, and filtering on event 29 is supported:

Filter on IMPLEMENTATION DEFINED event 29.

E[29]	Meaning
0b0	Event 29 is ignored.
0b1	Do not record samples that have event 29 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[28], bit [28]

When FEAT_SPEv1p4 is not implemented, event 28 is implemented, and filtering on event 28 is supported:

Filter on IMPLEMENTATION DEFINED event 28.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E[28]	Meaning
0b0	Event 28 is ignored.
0b1	Do not record samples that have event 28 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[27], bit [27]

When FEAT_SPEv1p4 is not implemented, event 27 is implemented, and filtering on event 27 is supported:

Filter on IMPLEMENTATION DEFINED event 27.

E[27]	Meaning
0b0	Event 27 is ignored.
0b1	Do not record samples that have event 27 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[26], bit [26]

When FEAT_SPEv1p4 is not implemented, event 26 is implemented, and filtering on event 26 is supported:

Filter on IMPLEMENTATION DEFINED event 26.

E[26]	Meaning
0b0	Event 26 is ignored.
0b1	Do not record samples that have event 26 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[25], bit [25]

When (FEAT_SPE_SME is implemented or FEAT_SPEv1p5 is implemented) and event 25 is implemented:

Filter on Not SMCU or other shared resource operation event.

E[25]	Meaning
0b0	SMCU or other shared resource operation event is ignored.
0b1	Do not record samples that have the SMCU or other shared resource operation event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_SPEv1p4 is not implemented, event 25 is implemented, and filtering on event 25 is supported:

Filter on IMPLEMENTATION DEFINED event 25.

E[25]	Meaning
0b0	Event 25 is ignored.
0b1	Do not record samples that have event 25 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[24], bit [24]

When FEAT_SPE_SME is implemented:

Filter on Non-streaming SVE mode event.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E[24]	Meaning
0b0	Streaming SVE mode event is ignored.
0b1	Do not record samples that have the Streaming SVE mode event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_SPEv1p4 is not implemented, event 24 is implemented, and filtering on event 24 is supported:

Filter on IMPLEMENTATION DEFINED event 24.

E[24]	Meaning
0b0	Event 24 is ignored.
0b1	Do not record samples that have event 24 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[23], bit [23]

When FEAT_SPEv1p4 is implemented and event 23 is implemented:

Filter on Data not snooped event.

E[23]	Meaning
0b0	Data snooped event is ignored.
0b1	Do not record samples that have the Data snooped event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[22], bit [22]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_SPEv1p4 is implemented and event 22 is implemented:

Filter on Not recently fetched event.

E[22]	Meaning
0b0	Recently fetched event is ignored.
0b1	Do not record samples that have the Recently fetched event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[21], bit [21]

When FEAT_SPEv1p4 is implemented and event 21 is implemented:

Filter on Cache data not modified event.

E[21]	Meaning
0b0	Cache data modified event is ignored.
0b1	Do not record samples that have the Cache data modified event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[20], bit [20]

When FEAT_SPEv1p4 is implemented and event 20 is implemented:

Filter on Level 2 data cache hit event.

E[20]	Meaning
0b0	Level 2 data cache miss event is ignored.
0b1	Do not record samples that have the Level 2 data cache miss event == 1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[19], bit [19]

When FEAT_SPEv1p4 is implemented and event 19 is implemented:

Filter on No level 2 data cache access event.

E[19]	Meaning
0b0	Level 2 data cache access event is ignored.
0b1	Do not record samples that have the Level 2 data cache access event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[18], bit [18]

When FEAT_SPEv1p1 is implemented and (FEAT_SVE is implemented or FEAT_SME is implemented):

Filter on Not empty predicate event.

E[18]	Meaning
0b0	Empty predicate event is ignored.
0b1	Do not record samples that have the Empty predicate event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[17], bit [17]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_SPEv1p1 is implemented and (FEAT_SVE is implemented or FEAT_SME is implemented):

Filter on Not partial predicate event.

E[17]	Meaning
`0b0`	Partial or empty predicate event is ignored.
`0b1`	Do not record samples that have the Partial or empty predicate event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

Bit [16]

Reserved, RAZ/WI.

E[15], bit [15]

When event 15 is implemented and filtering on event 15 is supported:

Filter on IMPLEMENTATION DEFINED event 15.

E[15]	Meaning
`0b0`	Event 15 is ignored.
`0b1`	Do not record samples that have event 15 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[14], bit [14]

When event 14 is implemented and filtering on event 14 is supported:

Filter on IMPLEMENTATION DEFINED event 14.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E[14]	Meaning
0b0	Event 14 is ignored.
0b1	Do not record samples that have event 14 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[13], bit [13]

When event 13 is implemented and filtering on event 13 is supported:

Filter on IMPLEMENTATION DEFINED event 13.

E[13]	Meaning
0b0	Event 13 is ignored.
0b1	Do not record samples that have event 13 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[12], bit [12]

When event 12 is implemented and filtering on event 12 is supported:

Filter on IMPLEMENTATION DEFINED event 12.

E[12]	Meaning
0b0	Event 12 is ignored.
0b1	Do not record samples that have event 12 == 1.

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[11], bit [11]

When FEAT_SPEv1p1 is implemented:

Filter on Aligned event.

E[11]	Meaning
0b0	Misalignment event is ignored.
0b1	Do not record samples that have the Misalignment event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[10], bit [10]

When (FEAT_SPEv1p4 is implemented or filtering on event 10 is optionally supported) and event 10 is implemented:

Filter on No remote access event.

E[10]	Meaning
0b0	Remote access event is ignored.
0b1	Do not record samples that have the Remote access event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[9], bit [9]

When (FEAT_SPEv1p4 is implemented or filtering on event 9 is optionally supported) and event 9 is implemented:

Filter on Last Level cache hit event.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E[9]	Meaning
0b0	Last Level cache miss event is ignored.
0b1	Do not record samples that have the Last Level cache miss event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[8], bit [8]

When (FEAT_SPEv1p4 is implemented or filtering on event 8 is optionally supported) and event 8 is implemented:

Filter on No Last Level cache access event.

E[8]	Meaning
0b0	Last Level cache access event is ignored.
0b1	Do not record samples that have the Last Level cache access event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[7], bit [7]

Filter on Correctly predicted event.

E[7]	Meaning
0b0	Mispredicted event is ignored.
0b1	Do not record samples that have the Mispredicted event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E[6], bit [6]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_SPE_FnE is implemented:

Filter on Taken event.

E[6]	Meaning
0b0	Not taken event is ignored.
0b1	Do not record samples that have the Not taken event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[5], bit [5]

Filter on TLB hit event.

E[5]	Meaning
0b0	TLB walk event is ignored.
0b1	Do not record samples that have the TLB walk event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E[4], bit [4]

When FEAT_SPEv1p4 is implemented or filtering on event 4 is optionally supported:

Filter on No TLB access event.

E[4]	Meaning
0b0	TLB access event is ignored.
0b1	Do not record samples that have the TLB access event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RAZ/WI.

E[3], bit [3]

Filter on Level 1 data cache hit event.

E[3]	Meaning
0b0	Level 1 data cache refill or miss event is ignored.
0b1	Do not record samples that have the Level 1 data cache refill or miss event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E[2], bit [2]

When FEAT_SPEv1p4 is implemented or filtering on event 2 is optionally supported:

Filter on No Level 1 data cache access event.

E[2]	Meaning
0b0	Level 1 data cache access event is ignored.
0b1	Do not record samples that have the Level 1 data cache access event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[1], bit [1]

When the PE supports sampling of speculative instructions:

Filter on Speculative event.

E[1]	Meaning
0b0	Architecturally retired event is ignored.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

0b1	Do not record samples that have the Architecturally retired event == 1.

This field is ignored by the PE when PMSFCR_EL1.FnE is 0.

If the PE does not support the sampling of speculative instructions, or always discards the sample record for speculative instructions, this bit reads as an UNKNOWN value and the PE ignores its value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

Bit [0]

Reserved, RAZ/WI.

Accessing PMSNEVFR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, PMSNEVFR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b001

```
if !IsFeatureImplemented(FEAT_SPE_FnE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMSN == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.nPMSNEVFR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPMSN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x850];
    else
        X[t, 64] = PMSNEVFR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMSN == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPMSN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = PMSNEVFR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = PMSNEVFR_EL1;
```

MSR PMSNEVFR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1001	0b1001	0b001

```
if !IsFeatureImplemented(FEAT_SPE_FnE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMSN == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.nPMSNEVFR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPMSN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x850] = X[t, 64];
    else
        PMSNEVFR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && CheckMDCR_EL3_NSPBTrap() then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.EnPMSN == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && CheckMDCR_EL3_NSPBTrap() then
        if EL3SDDUndef() then
            UNDEFINED;
        else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.EnPMSN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSNEVFR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    PMSNEVFR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.8 Branch Record Buffer Extension registers

This section lists the Branch Record Buffer registers in AArch64.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.8.1 BRBCR_EL1, Branch Record Buffer Control Register (EL1)

The BRBCR_EL1 characteristics are:

Purpose

Controls the Branch Record Buffer.

Configuration

This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBCR_EL1 are UNDEFINED.

Attributes

BRBCR_EL1 is a 64-bit register.

Field descriptions

Bits [63:24]

Reserved, RES0.

EXCEPTION, bit [23]

Enable the recording of entry to EL1 via an exception.

EXCEPTION	Meaning
0b0	Disable the recording of Branch records for exceptions when taken to EL1.
0b1	Enable the recording of Branch records for exceptions when taken to EL1.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

ERTN, bit [22]

Allow the recording Branch records for exception return instructions from EL1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ERTN	Meaning
0b0	Disable the recording Branch records for exception return instructions from EL1.
0b1	Enable the recording Branch records for exception return instructions from EL1.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

Bits [21:10]

Reserved, RES0.

FZPSS, bit [9]

When FEAT_PMUv3_SS is implemented:

Freeze BRBE on PMU Snapshot.

FZPSS	Meaning
0b0	Branch recording is not affected by this control.
0b1	If either EL2 is not implemented or BRBCR_EL2.FZPSS is 1, then a BRBE freeze event occurs when a successful Capture event occurs.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FZP, bit [8]

When FEAT_PMUv3 is implemented:

Freeze BRBE on PMU overflow.

FZP	Meaning
0b0	Branch recording is not affected by this control.
0b1	A BRBE freeze event occurs when the PE is in a Non-prohibited region, BRBFCR_EL1.PAUSED is 0, and any of the following applies: • For any event counter PMEVCNTR<m>_EL0 in the first range, PMOVSCLR_EL0[m] is 1, and either FEAT_SEBEP is not implemented or PMEVTYPER<m>_EL0.SYNC is 0. • FEAT_PMUv3_ICNTR is implemented, PMOVSCLR_EL0.F0 is 1, and either FEAT_SEBEP is not implemented or PMICFILTR_EL0.SYNC is 0. For more information about event counter ranges, see MDCR_EL2.HPMN.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Cold reset:
  - When FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [7]

Reserved, RES0.

TS, bits [6:5]

Timestamp Control.

TS	Meaning	Applies when
0b01	Virtual timestamp. The BRBE recorded timestamp is the physical counter value, minus the value of CNTVOFF_EL2.	
0b10	Guest physical timestamp. The BRBE recorded timestamp is the physical counter value minus a physical offset. If any of the following are true, the physical offset is zero, otherwise the physical offset is the value of CNTPOFF_EL2: • EL3 is implemented and SCR_EL3.ECVEn == 0. • EL2 is implemented and CNTHCTL_EL2.ECV == 0. • FEAT_ECV_POFF is not implemented.	FEAT_ECV is implemented
0b11	Physical timestamp. The BRBE recorded timestamp is the physical counter value.	

All other values are reserved.

This field is ignored by the PE when EL2 is implemented and BRBCR_EL2.TS != 0b00.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

MPRED, bit [4]

Mask the recording of mispredicts.

MPRED	Meaning
0b0	Disable the recording of mispredict information.
0b1	Allow the recording of mispredict information.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

CC, bit [3]

Enable the recording of cycle count information.

CC	Meaning
0b0	Disable the recording of cycle count information.
0b1	Allow the recording of cycle count information.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

Bit [2]

Reserved, RES0.

E1BRE, bit [1]

EL1 Branch recording enable.

E1BRE	Meaning
0b0	Branch recording prohibited at EL1.
0b1	Branch recording enabled at EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

E0BRE, bit [0]

EL0 Branch recording enable.

E0BRE	Meaning
0b0	Branch recording prohibited at EL0.
0b1	Branch recording enabled at EL0.

This field is ignored by the PE when all of the following are true:

- HCR_EL2.TGE == 1.
- EL2 is implemented and enabled in the current Security state.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing BRBCR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name `BRBCR_EL1` or `BRBCR_EL12` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, BRBCR_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b1001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.nBRBCTL == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x8E0];
    else
        X[t, 64] = BRBCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = BRBCR_EL2;
    else
        X[t, 64] = BRBCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = BRBCR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MSR BRBCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b1001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.nBRBCTL == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x8E0] = X[t, 64];
    else
        BRBCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        BRBCR_EL2 = X[t, 64];
    else
        BRBCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    BRBCR_EL1 = X[t, 64];
```

```
When FEAT_VHE is implemented MRS <Xt>, BRBCR_EL12
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b10	0b101	0b1001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x8E0];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
             ↪then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = BRBCR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = BRBCR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented MSR BRBCR_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b10	0b101	0b1001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x8E0] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1'
             ↪then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            BRBCR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        BRBCR_EL1 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.8.2 BRBCR_EL2, Branch Record Buffer Control Register (EL2)

The BRBCR_EL2 characteristics are:

Purpose

Controls the Branch Record Buffer.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBCR_EL2 are UNDEFINED.

Attributes

BRBCR_EL2 is a 64-bit register.

Field descriptions

Bits [63:24]

Reserved, RES0.

EXCEPTION, bit [23]

Enable the recording of entry to EL2 via an exception.

EXCEPTION	Meaning
0b0	Disable the recording of Branch records for exceptions when taken to EL2.
0b1	Enable the recording of Branch records for exceptions when taken to EL2.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

ERTN, bit [22]

Allow the recording Branch records for exception return instructions from EL2.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ERTN	Meaning
0b0	Disable the recording Branch records for exception return instructions from EL2.
0b1	Enable the recording Branch records for exception return instructions from EL2.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

Bits [21:10]

Reserved, RES0.

FZPSS, bit [9]

When FEAT_PMUv3_SS is implemented:

Freeze BRBE on PMU Snapshot.

FZPSS	Meaning
0b0	Branch recording is not affected by this control.
0b1	If BRBCR_EL1.FZPSS is 1, then a BRBE freeze event occurs when a PMU snapshot occurs.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FZP, bit [8]

When FEAT_PMUv3 is implemented:

Freeze BRBE on PMU overflow.

FZP	Meaning
0b0	Branch recording is not affected by this control.
0b1	A BRBE freeze event occurs when the PE is in a Non-prohibited region, BRBFCR_EL1.PAUSED is 0, and all the following are true for any event counter PMEVCNTR<m>_EL0 in the second range: • PMOVSCLR_EL0[m] is 1. • Either FEAT_SEBEP is not implemented or PMEVTYPER<m>_EL0.SYNC is 0. For more information about event counter ranges, see MDCR_EL2.HPMN.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Cold reset:
  - When FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [7]

Reserved, RES0.

TS, bits [6:5]

Timestamp Control.

TS	Meaning	Applies when
0b00	Timestamp controlled by BRBCR_EL1.TS.	
0b01	Virtual timestamp. The BRBE recorded timestamp is the physical counter value, minus the value of CNTVOFF_EL2.	
0b10	Guest physical timestamp. The BRBE recorded timestamp is the physical counter value minus a physical offset. If any of the following are true, the physical offset is zero, otherwise the physical offset is the value of CNTPOFF_EL2: • EL3 is implemented and SCR_EL3.ECVEn == 0. • EL2 is implemented and CNTHCTL_EL2.ECV == 0. • FEAT_ECV_POFF is not implemented.	FEAT_ECV is implemented
0b11	Physical timestamp. The BRBE recorded timestamp is the physical counter value.	

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

MPRED, bit [4]

Mask the recording of mispredicts.

MPRED	Meaning
0b0	Disable the recording of mispredict information.
0b1	Allow the recording of mispredict information.

If EL2 is not implemented, then the Effective value of this field is 1, other than for a direct read of the register.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

CC, bit [3]

Enable the recording of cycle count information.

CC	Meaning
0b0	Disable the recording of cycle count information.
0b1	Allow the recording of cycle count information.

If EL2 is not implemented, then the Effective value of this field is 1, other than for a direct read of the register.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

Bit [2]

Reserved, RES0.

E2BRE, bit [1]

EL2 Branch recording enable.

E2BRE	Meaning
0b0	Branch recording prohibited at EL2.
0b1	Branch recording enabled at EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

E0HBRE, bit [0]

EL0 Branch recording enable.

E0HBRE	Meaning
0b0	Branch recording prohibited at EL0 when HCR_EL2.TGE == 1.
0b1	Branch recording enabled at EL0 when HCR_EL2.TGE == 1.

This field is ignored by the PE when any of the following are true:

- HCR_EL2.TGE == 0.
- EL2 is disabled in the current Security state.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing BRBCR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name `BRBCR_EL2` or `BRBCR_EL1` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, BRBCR_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b1001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.nBRBCTL == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x8E0];
    else
        X[t, 64] = BRBCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = BRBCR_EL2;
    else
        X[t, 64] = BRBCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = BRBCR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, BRBCR_EL2
```

op0	op1	CRn	CRm	op2
0b10	0b100	0b1001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = BRBCR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = BRBCR_EL2;
```

```
MSR BRBCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b1001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.nBRBCTL == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EffectiveHCR_EL2_NVx() IN {'111'} then
            NVMem[0x8E0] = X[t, 64];
        else
            BRBCR_EL1 = X[t, 64];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif ELIsInHost(EL2) then
            BRBCR_EL2 = X[t, 64];
        else
            BRBCR_EL1 = X[t, 64];
    elsif PSTATE.EL == EL3 then
        BRBCR_EL1 = X[t, 64];
```

MSR BRBCR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b10	0b100	0b1001	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBCR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    BRBCR_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.8.3 BRBFCR_EL1, Branch Record Buffer Function Control Register

The BRBFCR_EL1 characteristics are:

Purpose

Functional controls for the Branch Record Buffer.

Configuration

This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBFCR_EL1 are UNDEFINED.

Attributes

BRBFCR_EL1 is a 64-bit register.

Field descriptions

Bits [63:30]

Reserved, RES0.

BANK, bits [29:28]

Branch record buffer bank access control.

BANK	Meaning
0b00	Select branch records 0 to 31.
0b01	Select branch records 32 to 63.

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [27:23]

Reserved, RES0.

CONDDIR, bit [22]

Match on conditional direct branch instructions.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CONDDIR	Meaning
0b0	Do not match on conditional direct branch instructions.
0b1	Match on conditional direct branch instructions.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

DIRCALL, bit [21]

Match on direct branch with link instructions.

DIRCALL	Meaning
0b0	Do not match on direct branch with link instructions.
0b1	Match on direct branch with link instructions.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

INDCALL, bit [20]

Match on indirect branch with link instructions.

INDCALL	Meaning
0b0	Do not match on indirect branch with link instructions.
0b1	Match on indirect branch with link instructions.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

RTN, bit [19]

Match on function return instructions.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

RTN	Meaning
0b0	Do not match on function return instructions.
0b1	Match on function return instructions.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

INDIRECT, bit [18]

Match on indirect branch instructions.

INDIRECT	Meaning
0b0	Do not match on indirect branch instructions.
0b1	Match on indirect branch instructions.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

DIRECT, bit [17]

Match on unconditional direct branch instructions.

DIRECT	Meaning
0b0	Do not match on unconditional direct branch instructions.
0b1	Match on unconditional direct branch instructions.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

EnI, bit [16]

Include or exclude matches.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EnI	Meaning
0b0	Include records for matches, and exclude records for non-matches.
0b1	Exclude records for matches, and include records for non-matches.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

Bits [15:8]

Reserved, RES0.

PAUSED, bit [7]

Branch recording Paused status.

PAUSED	Meaning
0b0	Branch recording is not Paused.
0b1	Branch recording is Paused.

The reset behavior of this field is:

- On a Cold reset:
  - When FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset:
  - When FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

Bits [6:0]

Reserved, RES0.

Accessing BRBFCR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, BRBFCR_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b1001	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.nBRBCTL == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = BRBFCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = BRBFCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = BRBFCR_EL1;
```

```
MSR BRBFCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b1001	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.nBRBCTL == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBFCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBFCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    BRBFCR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.8.4 BRBIDR0_EL1, Branch Record Buffer ID0 Register

The BRBIDR0_EL1 characteristics are:

Purpose

Indicates the features of the branch buffer unit.

Configuration

This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBIDR0_EL1 are UNDEFINED.

Attributes

BRBIDR0_EL1 is a 64-bit register.

Field descriptions

Bits [63:16]

Reserved, RES0.

CC, bits [15:12]

Cycle counter support. Defined values are:

CC	Meaning
0b0101	20-bit cycle counter implemented.

All other values are reserved.

Access to this field is RO.

FORMAT, bits [11:8]

Data format of records of the Branch record buffer. Defined values are:

FORMAT	Meaning
0b0000	Format 0.

All other values are reserved.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

NUMREC, bits [7:0]

Number of records supported.

The value of this field is an IMPLEMENTATION DEFINED choice of:

NUMREC	Meaning
0x08	8 branch records implemented.
0x10	16 branch records implemented.
0x20	32 branch records implemented.
0x40	64 branch records implemented.

All other values are reserved.

Access to this field is RO.

Accessing BRBIDR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, BRBIDR0_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b1001	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.nBRBIDR == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = BRBIDR0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = BRBIDR0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = BRBIDR0_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.8.5 BRBINFINJ_EL1, Branch Record Buffer Information Injection Register

The BRBINFINJ_EL1 characteristics are:

Purpose

The information of a Branch record for injection.

Configuration

This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBINFINJ_EL1 are UNDEFINED.

Attributes

BRBINFINJ_EL1 is a 64-bit register.

Field descriptions

Bits [63:47]

Reserved, RES0.

CCU, bit [46]

The number of PE clock cycles since the last Branch record entry is UNKNOWN.

CCU	Meaning
0b0	Indicates that the number of PE clock cycles since the last Branch record is indicated by BRBINFINJ_EL1.CC.
0b1	Indicates that the number of PE clock cycles since the last Branch record is UNKNOWN.

The value in this field is only valid when BRBINFINJ_EL1.VALID != 0b00.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When BRBINFINJ_EL1.VALID == ‘00’, access to this field is RES0.
- Otherwise, access to this field is RW.

CC, bits [45:32]

The number of PE clock cycles since the last Branch record entry.

The format of this field uses a mantissa and exponent to express the cycle count value, as follows:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- CC bits[7:0] indicate the mantissa M.
- CC bits[13:8] indicate the exponent E.

The cycle count is expressed using the following function:

```
if IsZero(E), then UInt(M) else UInt('1':M:Zeros(UInt(E)-1))
```

If required, the cycle count is rounded to a multiple of $2^{(E-1)}$ towards zero before being encoded.

A value of all ones in both the mantissa and exponent indicates the cycle count value exceeded the size of the cycle counter.

The value in this field is only valid when BRBINFINJ_EL1.VALID != `0b00`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is RES0 if any of the following are true:
  - BRBINFINJ_EL1.CCU == '1'
  - BRBINFINJ_EL1.VALID == '00'
- Otherwise, access to this field is RW.

Bits [31:14]

Reserved, RES0.

TYPE, bits [13:8]

Branch type.

TYPE	Meaning
`0b000000`	Unconditional direct branch, excluding Branch with link.
`0b000001`	Indirect branch, excluding Branch with link, Return from subroutine, and Exception return.
`0b000010`	Direct Branch with link.
`0b000011`	Indirect Branch with link.
`0b000101`	Return from subroutine.
`0b000111`	Exception return.
`0b001000`	Conditional direct branch.
`0b100001`	Debug halt.
`0b100010`	Call.
`0b100011`	Trap.
`0b100100`	SError.
`0b100110`	Instruction debug.
`0b100111`	Data debug.
`0b101010`	Alignment.
`0b101011`	Inst Fault.
`0b101100`	Data Fault.
`0b101110`	IRQ.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TYPE	Meaning
0b101111	FIQ.
0b110000	IMPLEMENTATION DEFINED exception to EL3.
0b111001	Debug State Exit.

All other values are reserved.

The value in this field is only valid when BRBINFINJ_EL1.VALID != 0b00.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When BRBINFINJ_EL1.VALID == '00', access to this field is RES0.
- Otherwise, access to this field is RW.

EL, bits [7:6]

The Exception level at the target address.

EL	Meaning	Applies when
0b00	EL0.	
0b01	EL1.	
0b10	EL2.	
0b11	EL3.	FEAT_BRBEv1p1 is implemented

The value in this field is only valid when BRBINFINJ_EL1.VALID == 0b11 or BRBINFINJ_EL1.VALID == 0b01.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is RES0 if any of the following are true:
  - BRBINFINJ_EL1.VALID == '00'
  - BRBINFINJ_EL1.VALID == '10'
- Otherwise, access to this field is RW.

MPRED, bit [5]

Branch mispredict.

MPRED	Meaning
0b0	Branch was correctly predicted or the result of the prediction was not captured.
0b1	Branch was incorrectly predicted.

The value in this field is only valid when BRBINFINJ_EL1.VALID == 0b11 or BRBINFINJ_EL1.VALID == 0b10.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is RES0 if any of the following are true:
  - BRBINFINJ_EL1.VALID == '00'
  - BRBINFINJ_EL1.VALID == '01'
  - BRBINFINJ_EL1.TYPE[5] == '1'
- Otherwise, access to this field is RW.

Bits [4:2]

Reserved, RES0.

VALID, bits [1:0]

The Branch record is valid.

VALID	Meaning
0b00	This Branch record is not valid. The values of following fields are not valid: • BRBTGTINJ_EL1.ADDRESS. • BRBSRCINJ_EL1.ADDRESS. • BRBINFINJ_EL1.MPRED. • BRBINFINJ_EL1.LASTFAILED. • BRBINFINJ_EL1.T. • BRBINFINJ_EL1.EL. • BRBINFINJ_EL1.TYPE. • BRBINFINJ_EL1.CC. • BRBINFINJ_EL1.CCU.
0b01	This Branch record is valid. The values of following fields are not valid: • BRBSRCINJ_EL1.ADDRESS. • BRBINFINJ_EL1.T. • BRBINFINJ_EL1.MPRED.
0b10	This Branch record is valid. The values of following fields are not valid: • BRBTGTINJ_EL1.ADDRESS. • BRBINFINJ_EL1.EL.
0b11	This Branch record is valid.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing BRBINFINJ_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, BRBINFINJ_EL1
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b10	0b001	0b1001	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.nBRBDATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = BRBINFINJ_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = BRBINFINJ_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = BRBINFINJ_EL1;
```

```
MSR BRBINFINJ_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b1001	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.nBRBDATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBINFINJ_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBINFINJ_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    BRBINFINJ_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.8.6 BRBINF<n>_EL1, Branch Record Buffer Information Register <n>, n = 0 - 31

The BRBINF<n>_EL1 characteristics are:

Purpose

The information for Branch record n + (BRBFCR_EL1.BANK × 32).

Configuration

This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBINF<n>_EL1 are UNDEFINED.

Attributes

BRBINF<n>_EL1 is a 64-bit register.

Field descriptions

Bits [63:47]

Reserved, RES0.

CCU, bit [46]

The number of PE clock cycles since the last Branch record entry is UNKNOWN.

CCU	Meaning
0b0	Indicates that the number of PE clock cycles since the last Branch record is indicated by BRBINF<n>_EL1.CC.
0b1	Indicates that the number of PE clock cycles since the last Branch record is UNKNOWN.

The value in this field is only valid when BRBINF<n>_EL1.VALID != 0b00.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When BRBINF<n>_EL1.VALID == ‘00’, access to this field is RES0.
- Otherwise, access to this field is RO.

CC, bits [45:32]

The number of PE clock cycles since the last Branch record entry.

The format of this field uses a mantissa and exponent to express the cycle count value, as follows:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- CC bits[7:0] indicate the mantissa M.
- CC bits[13:8] indicate the exponent E.

The cycle count is expressed using the following function:

```
if IsZero(E), then UInt(M) else UInt('1':M:Zeros(UInt(E)-1))
```

If required, the cycle count is rounded to a multiple of $2^{(E-1)}$ towards zero before being encoded.

A value of all ones in both the mantissa and exponent indicates the cycle count value exceeded the size of the cycle counter.

The value in this field is only valid when BRBINF<n>_EL1.VALID != `0b00`.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is RES0 if any of the following are true:
  - BRBINF<n>_EL1.CCU == ‘1’
  - BRBINF<n>_EL1.VALID == ‘00’
- Otherwise, access to this field is RO.

Bits [31:14]

Reserved, RES0.

TYPE, bits [13:8]

Branch type.

TYPE	Meaning
`0b000000`	Unconditional direct branch, excluding Branch with link.
`0b000001`	Indirect branch, excluding Branch with link, Return from subroutine, and Exception return.
`0b000010`	Direct Branch with link.
`0b000011`	Indirect Branch with link.
`0b000101`	Return from subroutine.
`0b000111`	Exception return.
`0b001000`	Conditional direct branch.
`0b100001`	Debug halt.
`0b100010`	Call.
`0b100011`	Trap.
`0b100100`	SError.
`0b100110`	Instruction debug.
`0b100111`	Data debug.
`0b101010`	Alignment.
`0b101011`	Inst Fault.
`0b101100`	Data Fault.
`0b101110`	IRQ.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TYPE	Meaning
0b101111	FIQ.
0b110000	IMPLEMENTATION DEFINED exception to EL3.
0b111001	Debug State Exit.

All other values are reserved.

The value in this field is only valid when BRBINF<n>_EL1.VALID != 0b00.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When BRBINF<n>_EL1.VALID == '00', access to this field is RES0.
- Otherwise, access to this field is RO.

EL, bits [7:6]

The Exception level at the target address.

EL	Meaning	Applies when
0b00	EL0.	
0b01	EL1.	
0b10	EL2.	
0b11	EL3.	FEAT_BRBEv1p1 is implemented

The value in this field is only valid when BRBINF<n>_EL1.VALID == 0b11 or BRBINF<n>_EL1.VALID == 0b01.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is RES0 if any of the following are true:
  - BRBINF<n>_EL1.VALID == '00'
  - BRBINF<n>_EL1.VALID == '10'
- Otherwise, access to this field is RO.

MPRED, bit [5]

Branch mispredict.

MPRED	Meaning
0b0	Branch was correctly predicted or the result of the prediction was not captured.
0b1	Branch was incorrectly predicted.

The value in this field is only valid when BRBINF<n>_EL1.VALID == 0b11 or BRBINF<n>_EL1.VALID == 0b10.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is RES0 if any of the following are true:
  - BRBINF<n>_EL1.VALID == '00'
  - BRBINF<n>_EL1.VALID == '01'
  - BRBINF<n>_EL1.TYPE[5] == '1'
- Otherwise, access to this field is RO.

Bits [4:2]

Reserved, RES0.

VALID, bits [1:0]

The Branch record is valid.

VALID	Meaning
0b00	This Branch record is not valid. The values of following fields are not valid: • BRBTGT<n>_EL1.ADDRESS. • BRBSRC<n>_EL1.ADDRESS. • BRBINF<n>_EL1.MPRED. • BRBINF<n>_EL1.LASTFAILED. • BRBINF<n>_EL1.T. • BRBINF<n>_EL1.EL. • BRBINF<n>_EL1.TYPE. • BRBINF<n>_EL1.CC. • BRBINF<n>_EL1.CCU.
0b01	This Branch record is valid. The values of following fields are not valid: • BRBSRC<n>_EL1.ADDRESS. • BRBINF<n>_EL1.T. • BRBINF<n>_EL1.MPRED.
0b10	This Branch record is valid. The values of following fields are not valid: • BRBTGT<n>_EL1.ADDRESS. • BRBINF<n>_EL1.EL.
0b11	This Branch record is valid.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing BRBINF<n>_EL1

BRBINF<n>_EL1 reads-as-zero if n + (BRBFCR_EL1.BANK × 32) >= BRBIDR0_EL1.NUMREC.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, BRBINF<m>_EL1 ; Where m = 0-31
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b10	0b001	0b1000	m[3:0]	m[4]:0b00

```
integer m = UInt(op2<2>:CRm<3:0>);

if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.nBRBDATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif m + (UInt(BRBFCR_EL1.BANK) * 32) >= NUM_BRBE_RECORDS then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = BRBINF_EL1[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif m + (UInt(BRBFCR_EL1.BANK) * 32) >= NUM_BRBE_RECORDS then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = BRBINF_EL1[m];
elsif PSTATE.EL == EL3 then
    if m + (UInt(BRBFCR_EL1.BANK) * 32) >= NUM_BRBE_RECORDS then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = BRBINF_EL1[m];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.8.7 BRBSRCINJ_EL1, Branch Record Buffer Source Address Injection Register

The BRBSRCINJ_EL1 characteristics are:

Purpose

The source address of a Branch record for injection.

Configuration

This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBSRCINJ_EL1 are UNDEFINED.

Attributes

BRBSRCINJ_EL1 is a 64-bit register.

Field descriptions

ADDRESS, bits [63:0]

Source virtual address of the Branch record.

When a direct write occurs with a value with ADDRESS bits [63:P] indicating an invalid address, an UNKNOWN value which indicates an invalid address is written to bits [63:P].

An invalid address is:

- For an address in a translation regime with 2 VA ranges, with bits [63:P] not all zeroes or all ones.
- For an address in a translation regime with a single VA range, with bits [63:P] not all zeroes.

P is defined as:

- 56, when FEAT_LVA3 is implemented.
- 52, when FEAT_LVA is implemented.
- 48, otherwise.

The value in bits [P-1:0] is the value written.

When a direct write occurs with a value with ADDRESS bits [63:P] indicating a valid address, the written value is written to bits [63:0], and a read of the register returns the written value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is RES0 if any of the following are true:
  - BRBINFINJ_EL1.VALID == ‘00’
  - BRBINFINJ_EL1.VALID == ‘01’
- Otherwise, access to this field is RW.

Accessing BRBSRCINJ_EL1

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, BRBSRCINJ_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b1001	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.nBRBDATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = BRBSRCINJ_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = BRBSRCINJ_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = BRBSRCINJ_EL1;
```

```
MSR BRBSRCINJ_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b1001	0b0001	0b001

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.nBRBDATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBSRCINJ_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBSRCINJ_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    BRBSRCINJ_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.8.8 BRBSRC<n>_EL1, Branch Record Buffer Source Address Register <n>, n = 0 - 31

The BRBSRC<n>_EL1 characteristics are:

Purpose

The source address of Branch record n + (BRBFCR_EL1.BANK × 32).

Configuration

This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBSRC<n>_EL1 are UNDEFINED.

Attributes

BRBSRC<n>_EL1 is a 64-bit register.

Field descriptions

ADDRESS, bits [63:0]

Source virtual address of the Branch record.

When an indirect write occurs with a value with ADDRESS bits [63:P] indicating an invalid address, an UNKNOWN value which indicates an invalid address is written to bits [63:P].

An invalid address is:

- For an address in a translation regime with 2 VA ranges, with bits [63:P] not all zeroes or all ones.
- For an address in a translation regime with a single VA range, with bits [63:P] not all zeroes.

P is defined as:

- 56, when FEAT_LVA3 is implemented.
- 52, when FEAT_LVA is implemented.
- 48, otherwise.

The value in bits [P-1:0] is the value written.

When an indirect write occurs with a value with ADDRESS bits [63:P] indicating a valid address, the written value is written to bits [63:0], and a read of the register returns the written value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is RES0 if any of the following are true:
  - BRBINF<n>_EL1.VALID == ‘00’
  - BRBINF<n>_EL1.VALID == ‘01’
- Otherwise, access to this field is RO.

Accessing BRBSRC<n>_EL1

BRBSRC<n>_EL1 is RES0 if n + (BRBFCR_EL1.BANK × 32) >= BRBIDR0_EL1.NUMREC.

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, BRBSRC<m>_EL1 ; Where m = 0-31
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b1000	m[3:0]	m[4]:0b01

```
integer m = UInt(op2<2>:CRm<3:0>);

if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.nBRBDATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif m + (UInt(BRBFCR_EL1.BANK) * 32) >= NUM_BRBE_RECORDS then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = BRBSRC_EL1[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif m + (UInt(BRBFCR_EL1.BANK) * 32) >= NUM_BRBE_RECORDS then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = BRBSRC_EL1[m];
elsif PSTATE.EL == EL3 then
    if m + (UInt(BRBFCR_EL1.BANK) * 32) >= NUM_BRBE_RECORDS then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = BRBSRC_EL1[m];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.8.9 BRBTGTINJ_EL1, Branch Record Buffer Target Address Injection Register

The BRBTGTINJ_EL1 characteristics are:

Purpose

The target address of a Branch record for injection.

Configuration

This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBTGTINJ_EL1 are UNDEFINED.

Attributes

BRBTGTINJ_EL1 is a 64-bit register.

Field descriptions

ADDRESS, bits [63:0]

Target virtual address of the Branch record.

When a direct write occurs with a value with ADDRESS bits [63:P] indicating an invalid address, an UNKNOWN value which indicates an invalid address is written to bits [63:P].

An invalid address is:

- For an address in a translation regime with 2 VA ranges, with bits [63:P] not all zeroes or all ones.
- For an address in a translation regime with a single VA range, with bits [63:P] not all zeroes.

P is defined as:

- 56, when FEAT_LVA3 is implemented.
- 52, when FEAT_LVA is implemented.
- 48, otherwise.

The value in bits [P-1:0] is the value written.

When a direct write occurs with a value with ADDRESS bits [63:P] indicating a valid address, the written value is written to bits [63:0], and a read of the register returns the written value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is RES0 if any of the following are true:
  - BRBINFINJ_EL1.VALID == ‘00’
  - BRBINFINJ_EL1.VALID == ‘10’
- Otherwise, access to this field is RW.

Accessing BRBTGTINJ_EL1

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, BRBTGTINJ_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b1001	0b0001	0b010

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.nBRBDATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = BRBTGTINJ_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = BRBTGTINJ_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = BRBTGTINJ_EL1;
```

```
MSR BRBTGTINJ_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b1001	0b0001	0b010

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.nBRBDATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBTGTINJ_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBTGTINJ_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    BRBTGTINJ_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.8.10 BRBTGT<n>_EL1, Branch Record Buffer Target Address Register <n>, n = 0 - 31

The BRBTGT<n>_EL1 characteristics are:

Purpose

The target address of Branch record n + (BRBFCR_EL1.BANK × 32).

Configuration

This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBTGT<n>_EL1 are UNDEFINED.

Attributes

BRBTGT<n>_EL1 is a 64-bit register.

Field descriptions

ADDRESS, bits [63:0]

Target virtual address of the Branch record.

When an indirect write occurs with a value with ADDRESS bits [63:P] indicating an invalid address, an UNKNOWN value which indicates an invalid address is written to bits [63:P].

An invalid address is:

- For an address in a translation regime with 2 VA ranges, with bits [63:P] not all zeroes or all ones.
- For an address in a translation regime with a single VA range, with bits [63:P] not all zeroes.

P is defined as:

- 56, when FEAT_LVA3 is implemented.
- 52, when FEAT_LVA is implemented.
- 48, otherwise.

The value in bits [P-1:0] is the value written.

When an indirect write occurs with a value with ADDRESS bits [63:P] indicating a valid address, the written value is written to bits [63:0], and a read of the register returns the written value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- Access to this field is RES0 if any of the following are true:
  - BRBINF<n>_EL1.VALID == ‘00’
  - BRBINF<n>_EL1.VALID == ‘10’
- Otherwise, access to this field is RO.

Accessing BRBTGT<n>_EL1

BRBTGT<n>_EL1 is RES0 if n + (BRBFCR_EL1.BANK × 32) >= BRBIDR0_EL1.NUMREC.

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, BRBTGT<m>_EL1 ; Where m = 0-31
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b1000	m[3:0]	m[4]:0b10

```
integer m = UInt(op2<2>:CRm<3:0>);

if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.nBRBDATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif m + (UInt(BRBFCR_EL1.BANK) * 32) >= NUM_BRBE_RECORDS then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = BRBTGT_EL1[m];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif m + (UInt(BRBFCR_EL1.BANK) * 32) >= NUM_BRBE_RECORDS then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = BRBTGT_EL1[m];
elsif PSTATE.EL == EL3 then
    if m + (UInt(BRBFCR_EL1.BANK) * 32) >= NUM_BRBE_RECORDS then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = BRBTGT_EL1[m];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.8.11 BRBTS_EL1, Branch Record Buffer Timestamp Register

The BRBTS_EL1 characteristics are:

Purpose

Captures the Timestamp value on a BRBE freeze event.

Configuration

This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBTS_EL1 are UNDEFINED.

Attributes

BRBTS_EL1 is a 64-bit register.

Field descriptions

TS, bits [63:0]

Timestamp value at the time of a BRBE freeze event.

The reset behavior of this field is:

- On a Warm reset:
  - When FEAT_BRBEv1p1 is not implemented, this field resets to `0x0000000000000000`.

When FEAT_BRBEv1p1 is implemented, Arm recommends that this field is preserved on a Warm reset, but it is IMPLEMENTATION DEFINED whether this field resets to 0 or is preserved.

When FEAT_BRBEv1p1 is implemented, on a Cold reset it is IMPLEMENTATION DEFINED whether this field resets to an architecturally UNKNOWN value.

Accessing BRBTS_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, BRBTS_EL1
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b1001	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGRTR_EL2.nBRBDATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = BRBTS_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = BRBTS_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = BRBTS_EL1;
```

```
MSR BRBTS_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b10	0b001	0b1001	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_BRBE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HDFGWTR_EL2.nBRBDATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBTS_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE IN {'x0'} && SCR_EL3.NS == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBTS_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    BRBTS_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.9 RAS registers

This section lists RAS registers in AArch64.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.9.1 DISR_EL1, Deferred Interrupt Status Register

The DISR_EL1 characteristics are:

Purpose

Records that an SError exception has been consumed by an `ESB` instruction.

Configuration

AArch64 System register DISR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DISR[31:0].

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to DISR_EL1 are UNDEFINED.

Attributes

DISR_EL1 is a 64-bit register.

Field descriptions

When DISR_EL1.IDS == '0':

Bits [63:32]

Reserved, RES0.

A, bit [31]

Set to 1 when an `ESB` instruction defers an asynchronous SError exception. If the implementation does not include any sources of SError exception that can be synchronized by an Error Synchronization Barrier, then this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [30:25]

Reserved, RES0.

IDS, bit [24]

Indicates the deferred SError exception type.

IDS	Meaning
`0b0`	Deferred error uses architecturally-defined format.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [23:18]

Reserved, RES0.

WU, bits [17:16]

When FEAT_RASv2 is implemented:

Write update. See the description of ESR_ELx.WU for an SError exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [15:13]

Reserved, RES0.

AET, bits [12:10]

Asynchronous Error Type. See the description of ESR_ELx.AET for an SError exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EA, bit [9]

External abort Type. See the description of ESR_ELx.EA for an SError exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [8]

Reserved, RES0.

WnRV, bit [7]

When FEAT_RASv2 is implemented:

Write-not-Read Valid. See the description of ESR_ELx.WnRV for an SError exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

WnR, bit [6]

When FEAT_RASv2 is implemented:

Write-not-Read. See the description of ESR_ELx.WnR for an SError exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

DFSC, bits [5:0]

Fault Status Code. See the description of ESR_ELx.DFSC for an SError exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When DISR_EL1.IDS == '1':

Bits [63:32]

Reserved, RES0.

A, bit [31]

Set to 1 when an ESB instruction defers an asynchronous SError exception. If the implementation does not include any sources of SError exception that can be synchronized by an Error Synchronization Barrier, then this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [30:25]

Reserved, RES0.

IDS, bit [24]

Indicates the deferred SError exception type.

IDS	Meaning
0b1	Deferred error uses IMPLEMENTATION DEFINED format.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [23:0]

IMPLEMENTATION DEFINED syndrome. See the description of ESR_ELx[23:0] for an SError exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing DISR_EL1

An indirect write to DISR_EL1 made by an ESB instruction does not require an explicit synchronization operation for the value that is written to be observed by a direct read of DISR_EL1 occurring in program order after the ESB instruction.

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, DISR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && (HCR_EL2.AMO == '1' || (IsFeatureImplemented(FEAT_DoubleFault2) &&
        ↪IsHCRXEL2Enabled() && HCRX_EL2.TMEA == '1')) then
        X[t, 64] = VDISR_EL2;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_E3DSE) && SCR_EL3.EnDSE == '1' then
        X[t, 64] = VDISR_EL3;
    elsif HaveEL(EL3) && !Halted() && SCR_EL3.EA == '1' then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = DISR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_E3DSE) && SCR_EL3.EnDSE == '1' then
        X[t, 64] = VDISR_EL3;
    elsif HaveEL(EL3) && !Halted() && SCR_EL3.EA == '1' then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = DISR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = DISR_EL1;
```

```
MSR DISR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && (HCR_EL2.AMO == '1' || (IsFeatureImplemented(FEAT_DoubleFault2) &&
        ↪IsHCRXEL2Enabled() && HCRX_EL2.TMEA == '1')) then
        VDISR_EL2 = X[t, 64];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_E3DSE) && SCR_EL3.EnDSE == '1' then
        VDISR_EL3 = X[t, 64];
    elsif HaveEL(EL3) && !Halted() && SCR_EL3.EA == '1' then
        return;
    else
        DISR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_E3DSE) && SCR_EL3.EnDSE == '1' then
        VDISR_EL3 = X[t, 64];
    elsif HaveEL(EL3) && !Halted() && SCR_EL3.EA == '1' then
        return;
    else
        DISR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL3 then
    DISR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.9.2 ERRIDR_EL1, Error Record ID Register

The ERRIDR_EL1 characteristics are:

Purpose

Defines the highest numbered index of the error records that can be accessed through the Error Record System registers.

Configuration

AArch64 System register ERRIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register ERRIDR[31:0].

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERRIDR_EL1 are UNDEFINED.

Attributes

ERRIDR_EL1 is a 64-bit register.

Field descriptions

Bits [63:16]

Reserved, RES0.

NUM, bits [15:0]

Highest numbered index of the records that can be accessed through the Error Record System registers plus one. Zero indicates no records can be accessed through the Error Record System registers.

Each implemented record is owned by a node. A node might own multiple records.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Accessing ERRIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ERRIDR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0011	0b000

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.ERRIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERRIDR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERRIDR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ERRIDR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.9.3 ERRSELR_EL1, Error Record Select Register

The ERRSELR_EL1 characteristics are:

Purpose

Selects an error record to be accessed through the Error Record System registers.

Configuration

If ERRIDR_EL1 indicates that zero error records are implemented, then it is IMPLEMENTATION DEFINED whether ERRSELR_EL1 is UNDEFINED or RES0.

AArch64 System register ERRSELR_EL1 bits [31:0] are architecturally mapped to AArch32 System register ERRSELR[31:0].

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERRSELR_EL1 are UNDEFINED.

Attributes

ERRSELR_EL1 is a 64-bit register.

Field descriptions

Bits [63:16]

Reserved, RES0.

SEL, bits [15:0]

Selects the error record accessed through the ERX registers.

For example, if ERRSELR_EL1.SEL is `0x0004`, then direct reads and writes of ERXSTATUS_EL1 access ERR4STATUS.

If ERRSELR_EL1.SEL is greater than or equal to ERRIDR_EL1.NUM, then all of the following apply:

- The value read back from ERRSELR_EL1.SEL is UNKNOWN.
- Unless the access to an ERX*_EL1 register generates a higher priority exception, one of the following occurs:
  - An UNKNOWN error record is selected.
  - The ERX*_EL1 registers are RAZ/WI.
  - ERX*_EL1 register reads and writes are NOPs.
  - ERX*_EL1 register reads and writes are UNDEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ERRSELR_EL1

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, ERRSELR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.ERRSELR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERRSELR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERRSELR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ERRSELR_EL1;
```

```
MSR ERRSELR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.ERRSELR_EL1 == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERRSELR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERRSELR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ERRSELR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.9.4 ERXADDR_EL1, Selected Error Record Address Register

The ERXADDR_EL1 characteristics are:

Purpose

Accesses ERR<n>ADDR for the error record <n> selected by ERRSELR_EL1.SEL.

Configuration

AArch64 System register ERXADDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register ERXADDR[31:0].

AArch64 System register ERXADDR_EL1 bits [63:32] are architecturally mapped to AArch32 System register ERXADDR2[31:0].

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERXADDR_EL1 are UNDEFINED.

Attributes

ERXADDR_EL1 is a 64-bit register.

Field descriptions

ERRnADDR, bits [63:0]

ERXADDR_EL1 accesses ERR<n>ADDR, where <n> is the value in ERRSELR_EL1.SEL.

Accessing ERXADDR_EL1

If ERRIDR_EL1.NUM is `0x0000` or ERRSELR_EL1.SEL is greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXADDR_EL1 is RAZ/WI.
- Direct reads and writes of ERXADDR_EL1 are NOPs.
- Direct reads and writes of ERXADDR_EL1 are UNDEFINED.

ERR<n>ADDR describes additional constraints that also apply when ERR<n>ADDR is accessed through ERXADDR_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ERXADDR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0100	0b011

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.ERXADDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXADDR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXADDR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ERXADDR_EL1;
```

```
MSR ERXADDR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0100	0b011

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.ERXADDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXADDR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXADDR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ERXADDR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.9.5 ERXCTLR_EL1, Selected Error Record Control Register

The ERXCTLR_EL1 characteristics are:

Purpose

Accesses ERR<n>CTLR for the error record <n> selected by ERRSELR_EL1.SEL.

Configuration

AArch64 System register ERXCTLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register ERXCTLR[31:0].

AArch64 System register ERXCTLR_EL1 bits [63:32] are architecturally mapped to AArch32 System register ERXCTLR2[31:0].

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERXCTLR_EL1 are UNDEFINED.

Attributes

ERXCTLR_EL1 is a 64-bit register.

Field descriptions

ERRnCTLR, bits [63:0]

ERXCTLR_EL1 accesses ERR<n>CTLR, where <n> is the value in ERRSELR_EL1.SEL.

Accessing ERXCTLR_EL1

If ERRIDR_EL1.NUM is `0x0000` or ERRSELR_EL1.SEL is greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXCTLR_EL1 is RAZ/WI.
- Direct reads and writes of ERXCTLR_EL1 are NOPs.
- Direct reads and writes of ERXCTLR_EL1 are UNDEFINED.

If ERRSELR_EL1.SEL is not the index of the first error record owned by a node, then ERR<n>CTLR is not present, meaning reads and writes of ERXCTLR_EL1 are RES0.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ERXCTLR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0100	0b001

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.ERXCTLR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXCTLR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXCTLR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ERXCTLR_EL1;
```

```
MSR ERXCTLR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0100	0b001

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.ERXCTLR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXCTLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXCTLR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ERXCTLR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.9.6 ERXFR_EL1, Selected Error Record Feature Register

The ERXFR_EL1 characteristics are:

Purpose

Accesses ERR<n>FR for the error record <n> selected by ERRSELR_EL1.SEL.

Configuration

AArch64 System register ERXFR_EL1 bits [31:0] are architecturally mapped to AArch32 System register ERXFR[31:0].

AArch64 System register ERXFR_EL1 bits [63:32] are architecturally mapped to AArch32 System register ERXFR2[31:0].

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERXFR_EL1 are UNDEFINED.

Attributes

ERXFR_EL1 is a 64-bit register.

Field descriptions

ERRnFR, bits [63:0]

ERXFR_EL1 accesses ERR<n>FR, where <n> is the value in ERRSELR_EL1.SEL.

Accessing ERXFR_EL1

If ERRIDR_EL1.NUM is `0x0000` or ERRSELR_EL1.SEL is greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXFR_EL1 is RAZ.
- Direct reads of ERXFR_EL1 are NOPs.
- Direct reads of ERXFR_EL1 are UNDEFINED.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ERXFR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0100	0b000

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.ERXFR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXFR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXFR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ERXFR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.9.7 ERXGSR_EL1, Selected Error Record Group Status Register

The ERXGSR_EL1 characteristics are:

Purpose

Shows the status for the records in a group of error records.

Accesses ERRGSR for the group of error records <n> selected by ERRSELR_EL1.SEL[15:6].

Configuration

This register is present only when FEAT_RASv2 is implemented. Otherwise, direct accesses to ERXGSR_EL1 are UNDEFINED.

Attributes

ERXGSR_EL1 is a 64-bit register.

Field descriptions

S<q> , bits [q], for q = 63 to 0

When error record m is implemented and error record m supports this type of reporting:

The status for error record <m>, where m = q + (UInt(ERRSELR_EL1.SEL[15:6])×64). A read-only copy of ERR<m>STATUS.V.

S<q>	Meaning
0b0	No error.
0b1	One or more errors.

Otherwise:

Reserved, RES0.

Accessing ERXGSR_EL1

If ERRIDR_EL1.NUM is 0x0000 or ERRSELR_EL1.SEL is greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An UNKNOWN group of error records are selected.
- ERXGSR_EL1 is RAZ.
- Direct reads of ERXGSR_EL1 are NOPs.
- Direct reads of ERXGSR_EL1 are UNDEFINED.

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, ERXGSR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0011	0b010

```
if !IsFeatureImplemented(FEAT_RASv2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT2) && ((HaveEL(EL3) && SCR_EL3.FGTEn2 == '0') ||
         ↪HFGRTR2_EL2.nERXGSR_EL1 == '0') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXGSR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXGSR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ERXGSR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.9.8 ERXMISC0_EL1, Selected Error Record Miscellaneous Register 0

The ERXMISC0_EL1 characteristics are:

Purpose

Accesses ERR<n>MISC0 for the error record <n> selected by ERRSELR_EL1.SEL.

Configuration

AArch64 System register ERXMISC0_EL1 bits [31:0] are architecturally mapped to AArch32 System register ERXMISC0[31:0].

AArch64 System register ERXMISC0_EL1 bits [63:32] are architecturally mapped to AArch32 System register ERXMISC1[31:0].

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERXMISC0_EL1 are UNDEFINED.

Attributes

ERXMISC0_EL1 is a 64-bit register.

Field descriptions

ERRnMISC0, bits [63:0]

ERXMISC0_EL1 accesses ERR<n>MISC0, where <n> is the value in ERRSELR_EL1.SEL.

Accessing ERXMISC0_EL1

If ERRIDR_EL1.NUM is `0x0000` or ERRSELR_EL1.SEL is greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXMISC0_EL1 is RAZ/WI.
- Direct reads and writes of ERXMISC0_EL1 are NOPs.
- Direct reads and writes of ERXMISC0_EL1 are UNDEFINED.

ERR<n>MISC0 describes additional constraints that also apply when ERR<n>MISC0 is accessed through ERXMISC0_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ERXMISC0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0101	0b000

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.ERXMISCn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXMISC0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXMISC0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ERXMISC0_EL1;
```

```
MSR ERXMISC0_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.ERXMISCn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXMISC0_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXMISC0_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ERXMISC0_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.9.9 ERXMISC1_EL1, Selected Error Record Miscellaneous Register 1

The ERXMISC1_EL1 characteristics are:

Purpose

Accesses ERR<n>MISC1 for the error record <n> selected by ERRSELR_EL1.SEL.

Configuration

AArch64 System register ERXMISC1_EL1 bits [31:0] are architecturally mapped to AArch32 System register ERXMISC2[31:0].

AArch64 System register ERXMISC1_EL1 bits [63:32] are architecturally mapped to AArch32 System register ERXMISC3[31:0].

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERXMISC1_EL1 are UNDEFINED.

Attributes

ERXMISC1_EL1 is a 64-bit register.

Field descriptions

ERRnMISC1, bits [63:0]

ERXMISC1_EL1 accesses ERR<n>MISC1, where <n> is the value in ERRSELR_EL1.SEL.

Accessing ERXMISC1_EL1

If ERRIDR_EL1.NUM is `0x0000` or ERRSELR_EL1.SEL is greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXMISC1_EL1 is RAZ/WI.
- Direct reads and writes of ERXMISC1_EL1 are NOPs.
- Direct reads and writes of ERXMISC1_EL1 are UNDEFINED.

ERR<n>MISC1 describes additional constraints that also apply when ERR<n>MISC1 is accessed through ERXMISC1_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ERXMISC1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0101	0b001

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.ERXMISCn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXMISC1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXMISC1_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ERXMISC1_EL1;
```

MSR ERXMISC1_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.ERXMISCn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXMISC1_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXMISC1_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ERXMISC1_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.9.10 ERXMISC2_EL1, Selected Error Record Miscellaneous Register 2

The ERXMISC2_EL1 characteristics are:

Purpose

Accesses ERR<n>MISC2 for the error record <n> selected by ERRSELR_EL1.SEL.

Configuration

AArch64 System register ERXMISC2_EL1 bits [31:0] are architecturally mapped to AArch32 System register ERXMISC4[31:0].

AArch64 System register ERXMISC2_EL1 bits [63:32] are architecturally mapped to AArch32 System register ERXMISC5[31:0].

This register is present only when FEAT_RASv1p1 is implemented. Otherwise, direct accesses to ERXMISC2_EL1 are UNDEFINED.

Attributes

ERXMISC2_EL1 is a 64-bit register.

Field descriptions

ERRnMISC2, bits [63:0]

ERXMISC2_EL1 accesses ERR<n>MISC2, where <n> is the value in ERRSELR_EL1.SEL.

Accessing ERXMISC2_EL1

If ERRIDR_EL1.NUM is `0x0000` or ERRSELR_EL1.SEL is greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXMISC2_EL1 is RAZ/WI.
- Direct reads and writes of ERXMISC2_EL1 are NOPs.
- Direct reads and writes of ERXMISC2_EL1 are UNDEFINED.

ERR<n>MISC2 describes additional constraints that also apply when ERR<n>MISC2 is accessed through ERXMISC2_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ERXMISC2_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0101	0b010

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_RASv1p1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.ERXMISCn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXMISC2_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXMISC2_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ERXMISC2_EL1;
```

MSR ERXMISC2_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0101	0b010

```
if !IsFeatureImplemented(FEAT_RASv1p1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.ERXMISCn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXMISC2_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXMISC2_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ERXMISC2_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.9.11 ERXMISC3_EL1, Selected Error Record Miscellaneous Register 3

The ERXMISC3_EL1 characteristics are:

Purpose

Accesses ERR<n>MISC3 for the error record <n> selected by ERRSELR_EL1.SEL.

Configuration

AArch64 System register ERXMISC3_EL1 bits [31:0] are architecturally mapped to AArch32 System register ERXMISC6[31:0].

AArch64 System register ERXMISC3_EL1 bits [63:32] are architecturally mapped to AArch32 System register ERXMISC7[31:0].

This register is present only when FEAT_RASv1p1 is implemented. Otherwise, direct accesses to ERXMISC3_EL1 are UNDEFINED.

Attributes

ERXMISC3_EL1 is a 64-bit register.

Field descriptions

ERRnMISC3, bits [63:0]

ERXMISC3_EL1 accesses ERR<n>MISC3, where <n> is the value in ERRSELR_EL1.SEL.

Accessing ERXMISC3_EL1

If ERRIDR_EL1.NUM is `0x0000` or ERRSELR_EL1.SEL is greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXMISC3_EL1 is RAZ/WI.
- Direct reads and writes of ERXMISC3_EL1 are NOPs.
- Direct reads and writes of ERXMISC3_EL1 are UNDEFINED.

ERR<n>MISC3 describes additional constraints that also apply when ERR<n>MISC3 is accessed through ERXMISC3_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ERXMISC3_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0101	0b011

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_RASv1p1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.ERXMISCn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXMISC3_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXMISC3_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ERXMISC3_EL1;
```

MSR ERXMISC3_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0101	0b011

```
if !IsFeatureImplemented(FEAT_RASv1p1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.ERXMISCn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXMISC3_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXMISC3_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ERXMISC3_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.9.12 ERXPFGCDN_EL1, Selected Pseudo-fault Generation Countdown Register

The ERXPFGCDN_EL1 characteristics are:

Purpose

Accesses ERR<n>PFGCDN for the error record <n> selected by ERRSELR_EL1.SEL.

Configuration

This register is present only when FEAT_RASv1p1 is implemented. Otherwise, direct accesses to ERXPFGCDN_EL1 are UNDEFINED.

Attributes

ERXPFGCDN_EL1 is a 64-bit register.

Field descriptions

ERRnPFGCDN, bits [63:0]

ERXPFGCDN_EL1 accesses ERR<n>PFGCDN, where <n> is the value in ERRSELR_EL1.SEL.

Accessing ERXPFGCDN_EL1

If ERRIDR_EL1.NUM is `0x0000` or ERRSELR_EL1.SEL is greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXPFGCDN_EL1 is RAZ/WI.
- Direct reads and writes of ERXPFGCDN_EL1 are NOPs.
- Direct reads and writes of ERXPFGCDN_EL1 are UNDEFINED.

If ERRSELR_EL1.SEL selects an error record owned by a node that does not implement the Common Fault Injection Model Extension, then one of the following occurs:

- ERXPFGCDN_EL1 is RAZ/WI.
- Direct reads and writes of ERXPFGCDN_EL1 are NOPs.
- Direct reads and writes of ERXPFGCDN_EL1 are UNDEFINED.

Note

A node does not implement the Common Fault Injection Model Extension if ERR<q>FR.INJ reads as `0b00`. <q> is the index of the first error record owned by the same node as error record <n>, where <n> is the value in ERRSELR_EL1.SEL. If the node owns a single record then q = n.

If ERRSELR_EL1.SEL is not the index of the first error record owned by a node, then ERR<n>PFGCDN is not present, meaning reads and writes of ERXPFGCDN_EL1 are RES0.

ERR<n>PFGCDN describes additional constraints that also apply when ERR<n>PFGCDN is accessed through ERXPFGCDN_EL1.

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, ERXPFGCDN_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0100	0b110

```
if !IsFeatureImplemented(FEAT_RASv1p1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIEN == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.FIEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.ERXPFGCDN_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIEN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXPFGCDN_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIEN == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FIEN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXPFGCDN_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ERXPFGCDN_EL1;
```

```
MSR ERXPFGCDN_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0100	0b110

```
if !IsFeatureImplemented(FEAT_RASv1p1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIEN == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.FIEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.ERXPFGCDN_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIEN == '0' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXPFGCDN_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIEN == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FIEN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXPFGCDN_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ERXPFGCDN_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.9.13 ERXPFGCTL_EL1, Selected Pseudo-fault Generation Control Register

The ERXPFGCTL_EL1 characteristics are:

Purpose

Accesses ERR<n>PFGCTL for the error record <n> selected by ERRSELR_EL1.SEL.

Configuration

This register is present only when FEAT_RASv1p1 is implemented. Otherwise, direct accesses to ERXPFGCTL_EL1 are UNDEFINED.

Attributes

ERXPFGCTL_EL1 is a 64-bit register.

Field descriptions

ERRnPFGCTL, bits [63:0]

ERXPFGCTL_EL1 accesses ERR<n>PFGCTL, where <n> is the value in ERRSELR_EL1.SEL.

Accessing ERXPFGCTL_EL1

If ERRIDR_EL1.NUM is `0x0000` or ERRSELR_EL1.SEL is greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXPFGCTL_EL1 is RAZ/WI.
- Direct reads and writes of ERXPFGCTL_EL1 are NOPs.
- Direct reads and writes of ERXPFGCTL_EL1 are UNDEFINED.

If ERRSELR_EL1.SEL selects an error record owned by a node that does not implement the Common Fault Injection Model Extension, then one of the following occurs:

- ERXPFGCTL_EL1 is RAZ/WI.
- Direct reads and writes of ERXPFGCTL_EL1 are NOPs.
- Direct reads and writes of ERXPFGCTL_EL1 are UNDEFINED.

Note

A node does not implement the Common Fault Injection Model Extension if ERR<q>FR.INJ reads as `0b00`. <q> is the index of the first error record owned by the same node as error record <n>, where <n> is the value in ERRSELR_EL1.SEL. If the node owns a single record then q = n.

If ERRSELR_EL1.SEL is not the index of the first error record owned by a node, then ERR<n>PFGCTL is not present, meaning reads and writes of ERXPFGCTL_EL1 are RES0.

ERR<n>PFGCTL describes additional constraints that also apply when ERR<n>PFGCTL is accessed through ERXPFGCTL_EL1.

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, ERXPFGCTL_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0100	0b101

```
if !IsFeatureImplemented(FEAT_RASv1p1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIEN == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.FIEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.ERXPFGCTL_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIEN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXPFGCTL_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIEN == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FIEN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXPFGCTL_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ERXPFGCTL_EL1;
```

```
MSR ERXPFGCTL_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0100	0b101

```
if !IsFeatureImplemented(FEAT_RASv1p1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIEN == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.FIEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.ERXPFGCTL_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIEN == '0' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXPFGCTL_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIEN == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FIEN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXPFGCTL_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ERXPFGCTL_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.9.14 ERXPFGF_EL1, Selected Pseudo-fault Generation Feature Register

The ERXPFGF_EL1 characteristics are:

Purpose

Accesses ERR<n>PFGF for the error record <n> selected by ERRSELR_EL1.SEL.

Configuration

This register is present only when FEAT_RASv1p1 is implemented. Otherwise, direct accesses to ERXPFGF_EL1 are UNDEFINED.

Attributes

ERXPFGF_EL1 is a 64-bit register.

Field descriptions

ERRnPFGF, bits [63:0]

ERXPFGF_EL1 accesses ERR<n>PFGF, where <n> is the value in ERRSELR_EL1.SEL.

Accessing ERXPFGF_EL1

If ERRIDR_EL1.NUM is `0x0000` or ERRSELR_EL1.SEL is greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXPFGF_EL1 is RAZ.
- Direct reads of ERXPFGF_EL1 are NOPs.
- Direct reads of ERXPFGF_EL1 are UNDEFINED.

If ERRSELR_EL1.SEL selects an error record owned by a node that does not implement the Common Fault Injection Model Extension, then one of the following occurs:

- ERXPFGF_EL1 is RAZ.
- Direct reads of ERXPFGF_EL1 are NOPs.
- Direct reads of ERXPFGF_EL1 are UNDEFINED.

Note

A node does not implement the Common Fault Injection Model Extension if ERR<q>FR.INJ reads as `0b00`. <q> is the index of the first error record owned by the same node as error record <n>, where <n> is the value in ERRSELR_EL1.SEL. If the node owns a single record then q = n.

If ERRSELR_EL1.SEL is not the index of the first error record owned by a node, then ERR<n>PFGF is not present, meaning reads of ERXPFGF_EL1 are RES0.

ERR<n>PFGF describes additional constraints that also apply when ERR<n>PFGF is accessed through ERXPFGF_EL1.

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, ERXPFGF_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0100	0b100

```
if !IsFeatureImplemented(FEAT_RASv1p1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIEN == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.FIEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.ERXPFGF_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIEN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXPFGF_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.FIEN == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FIEN == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXPFGF_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ERXPFGF_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.9.15 ERXSTATUS_EL1, Selected Error Record Primary Status Register

The ERXSTATUS_EL1 characteristics are:

Purpose

Accesses ERR<n>STATUS for the error record <n> selected by ERRSELR_EL1.SEL.

Configuration

AArch64 System register ERXSTATUS_EL1 bits [31:0] are architecturally mapped to AArch32 System register ERXSTATUS[31:0].

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERXSTATUS_EL1 are UNDEFINED.

Attributes

ERXSTATUS_EL1 is a 64-bit register.

Field descriptions

ERRnSTATUS, bits [63:0]

ERXSTATUS_EL1 accesses ERR<n>STATUS, where <n> is the value in ERRSELR_EL1.SEL.

Accessing ERXSTATUS_EL1

If ERRIDR_EL1.NUM is `0x0000` or ERRSELR_EL1.SEL is greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXSTATUS_EL1 is RAZ/WI.
- Direct reads and writes of ERXSTATUS_EL1 are NOPs.
- Direct reads and writes of ERXSTATUS_EL1 are UNDEFINED.

ERR<n>STATUS describes additional constraints that also apply when ERR<n>STATUS is accessed through ERXSTATUS_EL1.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, ERXSTATUS_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0100	0b010

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.ERXSTATUS_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXSTATUS_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = ERXSTATUS_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = ERXSTATUS_EL1;
```

```
MSR ERXSTATUS_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0101	0b0100	0b010

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.ERXSTATUS_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXSTATUS_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.TWERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TWERR == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXSTATUS_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    ERXSTATUS_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.9.16 VDISR_EL2, Virtual Deferred Interrupt Status Register (EL2)

The VDISR_EL2 characteristics are:

Purpose

Records that a virtual SError exception has been consumed by an `ESB` instruction executed at EL1.

An indirect write to VDISR_EL2 made by an `ESB` instruction does not require an explicit synchronization operation for the value written to be observed by a direct read of one of the following registers occurring in program order after the `ESB` instruction:

- DISR_EL1.
- DISR.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

AArch64 System register VDISR_EL2 bits [31:0] are architecturally mapped to AArch32 System register VDISR[31:0].

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to VDISR_EL2 are UNDEFINED.

Attributes

VDISR_EL2 is a 64-bit register.

Field descriptions

When EL1 is using AArch64:

Bits [63:32]

Reserved, RES0.

A, bit [31]

Set to 1 when an `ESB` instruction defers a virtual SError exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [30:25]

Reserved, RES0.

IDS, bit [24]

The value copied from VSESR_EL2.IDS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ISS, bits [23:0]

The value copied from VSESR_EL2.ISS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When EL1 is using AArch32 and VDISR_EL2.LPAE == '0':

Bits [63:32]

Reserved, RES0.

A, bit [31]

Set to 1 when an `ESB` instruction defers a virtual SError exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [30:16]

Reserved, RES0.

AET, bits [15:14]

The value copied from VSESR_EL2.AET.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [13]

Reserved, RES0.

ExT, bit [12]

The value copied from VSESR_EL2.ExT.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [11]

Reserved, RES0.

FS, bits [10, 3:0]

Fault status code. Set to `0b10110` when an `ESB` instruction defers a virtual SError exception.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

FS	Meaning
0b10110	Asynchronous SError exception.

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

LPAE, bit [9]

Format.

Set to TTBCR.EAE when an ESB instruction defers a virtual SError exception.

LPAE	Meaning
0b0	Using the Short-descriptor translation table format.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:4]

Reserved, RES0.

When EL1 is using AArch32 and VDISR_EL2.LPAE == '1':

Bits [63:32]

Reserved, RES0.

A, bit [31]

Set to 1 when an ESB instruction defers a virtual SError exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [30:16]

Reserved, RES0.

AET, bits [15:14]

The value copied from VSESR_EL2.AET.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [13]

Reserved, RES0.

ExT, bit [12]

The value copied from VSESR_EL2.ExT.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:10]

Reserved, RES0.

LPAE, bit [9]

Format.

Set to TTBCR.EAE when an ESB instruction defers a virtual SError exception.

LPAE	Meaning
0b1	Using the Long-descriptor translation table format.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:6]

Reserved, RES0.

STATUS, bits [5:0]

Fault status code. Set to 0b010001 when an ESB instruction defers a virtual SError exception.

STATUS	Meaning
0b010001	Asynchronous SError exception.

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing VDISR_EL2

An indirect write to VDISR_EL2 made by an ESB instruction does not require an explicit synchronization operation for the value that is written to be observed by a direct read of one of the following registers occurring in program order after the ESB instruction:

- DISR_EL1.
- DISR.

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, VDISR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1100	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x500];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = VDISR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = VDISR_EL2;
```

```
MSR VDISR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1100	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x500] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VDISR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    VDISR_EL2 = X[t, 64];
```

```
MRS <Xt>, DISR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b0001	0b001

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && (HCR_EL2.AMO == '1' || (IsFeatureImplemented(FEAT_DoubleFault2) &&
         ↪IsHCRXEL2Enabled() && HCRX_EL2.TMEA == '1')) then
        X[t, 64] = VDISR_EL2;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_E3DSE) && SCR_EL3.EnDSE == '1' then
        X[t, 64] = VDISR_EL3;
    elsif HaveEL(EL3) && !Halted() && SCR_EL3.EA == '1' then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = DISR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_E3DSE) && SCR_EL3.EnDSE == '1' then
        X[t, 64] = VDISR_EL3;
    elsif HaveEL(EL3) && !Halted() && SCR_EL3.EA == '1' then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = DISR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = DISR_EL1;
```

MSR DISR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && (HCR_EL2.AMO == '1' || (IsFeatureImplemented(FEAT_DoubleFault2) &&
         ↪IsHCRXEL2Enabled() && HCRX_EL2.TMEA == '1')) then
        VDISR_EL2 = X[t, 64];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_E3DSE) && SCR_EL3.EnDSE == '1' then
        VDISR_EL3 = X[t, 64];
    elsif HaveEL(EL3) && !Halted() && SCR_EL3.EA == '1' then
        return;
    else
        DISR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_E3DSE) && SCR_EL3.EnDSE == '1' then
        VDISR_EL3 = X[t, 64];
    elsif HaveEL(EL3) && !Halted() && SCR_EL3.EA == '1' then
        return;
    else
        DISR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    DISR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.9.17 VDISR_EL3, Virtual Deferred Interrupt Status Register (EL3)

The VDISR_EL3 characteristics are:

Purpose

Records that a delegated SError exception has been consumed by an `ESB` instruction executed at EL2 or EL1 when the Effective value of SCR_EL3.DSE is 1.

An indirect write to VDISR_EL3 made by an `ESB` instruction does not require an explicit synchronization operation for the value written to be observed by a direct read of DISR_EL1 occurring in program order after the `ESB` instruction.

Configuration

Note

The encoding for this register is subject to change.

This register is present only when FEAT_E3DSE is implemented. Otherwise, direct accesses to VDISR_EL3 are UNDEFINED.

Attributes

VDISR_EL3 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

A, bit [31]

Set to 1 when an `ESB` instruction defers a delegated SError exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [30:25]

Reserved, RES0.

IDS, bit [24]

The value copied from VSESR_EL3.IDS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS, bits [23:0]

The value copied from VSESR_EL3.ISS.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing VDISR_EL3

An indirect write to VDISR_EL3 made by an ESB instruction does not require an explicit synchronization operation for the value that is written to be observed by a direct read of DISR_EL1 occurring in program order after the ESB instruction.

Note

The accessibility pseudocode for DISR_EL1 has not been updated to show the effect of VDISR_EL3.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, VDISR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1100	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_E3DSE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = VDISR_EL3;
```

```
MSR VDISR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1100	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_E3DSE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    VDISR_EL3 = X[t, 64];
```

```
MRS <Xt>, DISR_EL1
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && (HCR_EL2.AMO == '1' || (IsFeatureImplemented(FEAT_DoubleFault2) &&
         ↪IsHCRXEL2Enabled() && HCRX_EL2.TMEA == '1')) then
        X[t, 64] = VDISR_EL2;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_E3DSE) && SCR_EL3.EnDSE == '1' then
        X[t, 64] = VDISR_EL3;
    elsif HaveEL(EL3) && !Halted() && SCR_EL3.EA == '1' then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = DISR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_E3DSE) && SCR_EL3.EnDSE == '1' then
        X[t, 64] = VDISR_EL3;
    elsif HaveEL(EL3) && !Halted() && SCR_EL3.EA == '1' then
        X[t, 64] = Zeros(64);
    else
        X[t, 64] = DISR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = DISR_EL1;
```

```
MSR DISR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1100	0b0001	0b001

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && (HCR_EL2.AMO == '1' || (IsFeatureImplemented(FEAT_DoubleFault2) &&
         ↪IsHCRXEL2Enabled() && HCRX_EL2.TMEA == '1')) then
        VDISR_EL2 = X[t, 64];
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_E3DSE) && SCR_EL3.EnDSE == '1' then
        VDISR_EL3 = X[t, 64];
    elsif HaveEL(EL3) && !Halted() && SCR_EL3.EA == '1' then
        return;
    else
        DISR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && IsFeatureImplemented(FEAT_E3DSE) && SCR_EL3.EnDSE == '1' then
        VDISR_EL3 = X[t, 64];
    elsif HaveEL(EL3) && !Halted() && SCR_EL3.EA == '1' then
        return;
    else
        DISR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    DISR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.9.18 VSESR_EL2, Virtual SError Exception Syndrome Register (EL2)

The VSESR_EL2 characteristics are:

Purpose

Provides the syndrome value reported to software on taking a virtual SError exception to EL1, or on executing an `ESB` instruction at EL1.

When the virtual SError exception injected using HCR_EL2.VSE is taken to EL1 using AArch64, then the syndrome value is reported in ESR_EL1.

When the virtual SError exception injected using HCR_EL2.VSE is taken to EL1 using AArch32, then the syndrome value is reported in DFSR.{AET, ExT} and the remainder of DFSR is set as defined by VMSAv8-32. For more information, see The AArch32 Virtual Memory System Architecture.

When the virtual SError exception injected using HCR_EL2.VSE is deferred by an `ESB` instruction, then the syndrome value is written to VDISR_EL2.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

AArch64 System register VSESR_EL2 bits [31:0] are architecturally mapped to AArch32 System register VDFSR[31:0].

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to VSESR_EL2 are UNDEFINED.

Attributes

VSESR_EL2 is a 64-bit register.

Field descriptions

When EL1 is using AArch32:

Bits [63:16]

Reserved, RES0.

AET, bits [15:14]

When a virtual SError exception is taken to EL1 using AArch32, DFSR[15:14] is set to VSESR_EL2.AET.

When a virtual SError exception is deferred by an `ESB` instruction, VDISR_EL2[15:14] is set to VSESR_EL2.AET.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [13]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ExT, bit [12]

When a virtual SError exception is taken to EL1 using AArch32, DFSR[12] is set to VSESR_EL2.ExT.

When a virtual SError exception is deferred by an ESB instruction, VDISR_EL2[12] is set to VSESR_EL2.ExT.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:0]

Reserved, RES0.

When EL1 is using AArch64:

Bits [63:25]

Reserved, RES0.

IDS, bit [24]

When a virtual SError exception is taken to EL1 using AArch64, ESR_EL1[24] is set to VSESR_EL2.IDS.

When a virtual SError exception is deferred by an ESB instruction, VDISR_EL2[24] is set to VSESR_EL2.IDS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS, bits [23:0]

When a virtual SError exception is taken to EL1 using AArch64, ESR_EL1[23:0] is set to VSESR_EL2.ISS.

When a virtual SError exception is deferred by an ESB instruction, VDISR_EL2[23:0] is set to VSESR_EL2.ISS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing VSESR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, VSESR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0101	0b0010	0b011

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x508];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = VSESR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = VSESR_EL2;
```

MSR VSESR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0101	0b0010	0b011

```
if !IsFeatureImplemented(FEAT_RAS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x508] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VSESR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    VSESR_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.9.19 VSESR_EL3, Virtual SError Exception Syndrome Register (EL3)

The VSESR_EL3 characteristics are:

Purpose

Provides the syndrome value reported to software when the Effective value of SCR_EL3.DSE is 1 on taking a delegated SError exception to EL2 or EL1, or on executing an `ESB` instruction at EL2 or EL1.

When the delegated SError exception injected using SCR_EL3.DSE is taken to EL2 using AArch64, then the syndrome value is reported in ESR_EL2.

When the delegated SError exception injected using SCR_EL3.DSE is taken to EL1 using AArch64, then the syndrome value is reported in ESR_EL1.

When the delegated SError exception injected using SCR_EL3.DSE is deferred by an `ESB` instruction, then the syndrome value is written to VDISR_EL3.

Configuration

This register is present only when FEAT_E3DSE is implemented. Otherwise, direct accesses to VSESR_EL3 are UNDEFINED.

Attributes

VSESR_EL3 is a 64-bit register.

Field descriptions

Bits [63:25]

Reserved, RES0.

IDS, bit [24]

When a delegated SError exception triggered by SCR_EL3.DSE is taken to EL2 or EL1 using AArch64, ESR_ELx[24] is set to VSESR_EL3.IDS.

When a delegated SError exception triggered by SCR_EL3.DSE is deferred by an `ESB` instruction, VDISR_EL3[24] is set to VSESR_EL3.IDS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS, bits [23:0]

When a delegated SError exception triggered by SCR_EL3.DSE is taken to EL2 or EL1 using AArch64, ESR_ELx[23:0] is set to VSESR_EL3.ISS.

When a delegated SError exception triggered by SCR_EL3.DSE is deferred by an `ESB` instruction, VDISR_EL3[23:0] is set to VSESR_EL3.ISS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing VSESR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, VSESR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0101	0b0010	0b011

```
if !IsFeatureImplemented(FEAT_E3DSE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = VSESR_EL3;
```

```
MSR VSESR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0101	0b0010	0b011

```
if !IsFeatureImplemented(FEAT_E3DSE) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    VSESR_EL3 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.10 Generic Timer registers

This section lists the Generic Timer registers in AArch64.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.10.1 CNTFRQ_EL0, Counter-timer Frequency Register

The CNTFRQ_EL0 characteristics are:

Purpose

This register is provided so that software can discover the effective frequency of the system counter. It must be programmed with this value as part of system initialization. The value of the register is not interpreted by hardware.

Configuration

AArch64 System register CNTFRQ_EL0 bits [31:0] are architecturally mapped to AArch32 System register CNTFRQ[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to CNTFRQ_EL0 are UNDEFINED.

Attributes

CNTFRQ_EL0 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

ClockFreq, bits [31:0]

Clock frequency. Indicates the effective frequency of the system counter, in Hz.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTFRQ_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTFRQ_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.<EL0PCTEN,EL0VCTEN> == '00' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.<EL0PCTEN,EL0VCTEN> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = CNTFRQ_EL0;
elsif PSTATE.EL == EL1 then
    X[t, 64] = CNTFRQ_EL0;
elsif PSTATE.EL == EL2 then
    X[t, 64] = CNTFRQ_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTFRQ_EL0;
```

```
MSR CNTFRQ_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0000	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif IsHighestEL(PSTATE.EL) then
    CNTFRQ_EL0 = X[t, 64];
else
    UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.10.2 CNTHCTL_EL2, Counter-timer Hypervisor Control Register

The CNTHCTL_EL2 characteristics are:

Purpose

Controls the generation of an event stream from the physical counter, and access from EL1 to the physical counter and the EL1 physical timer.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

AArch64 System register CNTHCTL_EL2 bits [31:0] are architecturally mapped to AArch32 System register CNTHCTL[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to CNTHCTL_EL2 are UNDEFINED.

Attributes

CNTHCTL_EL2 is a 64-bit register.

Field descriptions

When ELIsInHost(EL2):

Bits [63:20]

Reserved, RES0.

CNTPMASK, bit [19]

When FEAT_RME is implemented:

CNTPMASK	Meaning
0b0	This control has no effect on CNTP_CTL_EL0.IMASK.
0b1	CNTP_CTL_EL0.IMASK behaves as if set to 1 for all purposes other than a direct read of the field.

This bit is RES0 in Non-secure and Secure state.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CNTVMASK, bit [18]

When FEAT_RME is implemented:

CNTVMASK	Meaning
0b0	This control has no effect on CNTV_CTL_EL0.IMASK.
0b1	CNTV_CTL_EL0.IMASK behaves as if set to 1 for all purposes other than a direct read of the field.

This bit is RES0 in Non-secure and Secure state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EVNTIS, bit [17]

When FEAT_ECV is implemented:

Controls the scale of the generation of the event stream.

EVNTIS	Meaning
0b0	The CNTHCTL_EL2.EVNTI field applies to CNTPCT_EL0[15:0].
0b1	The CNTHCTL_EL2.EVNTI field applies to CNTPCT_EL0[23:8].

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1NVVCT, bit [16]

When FEAT_ECV is implemented:

When HCR_EL2.TGE is 0 and the Effective value of HCR_EL2.{NV2, NV1, NV} is {1, 0, 1}, traps EL1 accesses to the specified EL1 virtual timer registers using the EL02 descriptors to EL2 as follows:

Accesses to CNTV_CTL_EL02 and CNTV_CVAL_EL02 are trapped to EL2 and reported with EC syndrome value 0x18.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EL1NVVCT	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL1 accesses to the specified registers are trapped to EL2.

If HCR_EL2.TGE is 1 or the Effective value of HCR_EL2.{NV2, NV1, NV} is not {1, 0, 1}, this control does not cause any instructions to be trapped.

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1NVPCT, bit [15]

When FEAT_ECV is implemented:

When HCR_EL2.TGE is 0 and the Effective value of HCR_EL2.{NV2, NV1, NV} is {1, 0, 1}, traps EL1 accesses to the specified EL1 physical timer registers using the EL02 descriptors to EL2 as follows:

Accesses to CNTP_CTL_EL02 and CNTP_CVAL_EL02 are trapped to EL2 and reported with EC syndrome value 0x18.

EL1NVPCT	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL1 accesses to the specified registers are trapped to EL2.

If HCR_EL2.TGE is 1 or the Effective value of HCR_EL2.{NV2, NV1, NV} is not {1, 0, 1}, this control does not cause any instructions to be trapped.

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1TVCT, bit [14]

When FEAT_ECV is implemented:

Traps EL0 and EL1 accesses to the EL1 virtual counter registers to EL2 when EL2 is enabled in the current Security state, as follows:

- In AArch64 state, accesses to CNTVCT_EL0 and CNTVCTSS_EL0 are trapped to EL2 and reported using EC syndrome value 0x18, unless they are trapped by CNTKCTL_EL1.EL0VCTEN.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- In AArch32 state, accesses to CNTVCT are trapped to EL2 and reported with EC syndrome value `0x04`, unless they are trapped by CNTKCTL_EL1.EL0VCTEN or CNTKCTL.PL0VCTEN.

EL1TVCT	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	EL0 and EL1 accesses to the specified registers are trapped to EL2.

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

If HCR_EL2.TGE is 1, this control does not cause any instructions to be trapped.

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1TVT, bit [13]

When FEAT_ECV is implemented:

When the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, traps EL0 and EL1 accesses to the EL1 virtual timer registers to EL2, when EL2 is enabled for the current Security state, as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2 and reported with EC syndrome value `0x18`, unless they are trapped by CNTKCTL_EL1.EL0VTEN:
  - CNTV_CTL_EL0.
  - CNTV_CVAL_EL0.
  - CNTV_TVAL_EL0.
- In AArch32 state, MRC and MCR accesses to the following registers are trapped to EL2 and reported using EC syndrome value `0x03`, and MCRR and MRRC accesses are trapped to EL2 and reported using EC syndrome value `0x04`, unless they are trapped by CNTKCTL_EL1.EL0VTEN or CNTKCTL.PL0VTEN:
  - CNTV_CTL.
  - CNTV_CVAL.
  - CNTV_TVAL.

EL1TVT	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	EL0 and EL1 accesses to the specified registers are trapped to EL2.

If HCR_EL2.TGE is 1, this control does not cause any instructions to be trapped.

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Otherwise:

Reserved, RES0.

ECV, bit [12]

When FEAT_ECV_POFF is implemented:

Enables the Enhanced Counter Virtualization functionality registers.

When the Enhanced Counter Virtualization is enabled, the behavior is as follows:

- An MRS to CNTPCT_EL0 from either EL0 or EL1 that is not trapped will return the value (`PCount`<63:0> - CNTPOFF_EL2<63:0>).
- The EL1 physical timer interrupt is triggered when ((`PCount`<63:0> - CNTPOFF_EL2<63:0>) - `PCVal`<63:0>) is greater than or equal to 0.

`PCount`<63:0> is the physical count returned when CNTPCT_EL0 is read from EL2 or EL3.

`PCVal`<63:0> is the EL1 physical timer compare value for this timer.

ECV	Meaning
`0b0`	Enhanced Counter Virtualization functionality is disabled.
`0b1`	Enhanced Counter Virtualization functionality is enabled.

When HCR_EL2.TGE is 1 or SCR_EL3.{NS, EEL2} is {0, 0}, the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1PTEN, bit [11]

When HCR_EL2.TGE is 0, traps EL0 and EL1 accesses to the EL1 physical timer registers to EL2 when EL2 is enabled in the current Security state, as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2 and reported with EC syndrome value `0x18`, unless they are trapped by CNTKCTL_EL1.EL0PTEN:
  - CNTP_CTL_EL0.
  - CNTP_CVAL_EL0.
  - CNTP_TVAL_EL0.
- In AArch32 state, MRC and MCR accesses to the following registers are trapped and reported using EC syndrome value `0x03` and MRRC and MCRR accesses are trapped and reported using EC syndrome value `0x04`, unless they are trapped by CNTKCTL_EL1.EL0PTEN or CNTKCTL.PL0PTEN:
  - CNTP_CTL.
  - CNTP_CVAL.
  - CNTP_TVAL.

EL1PTEN	Meaning
`0b0`	EL0 and EL1 accesses to the specified registers are trapped to EL2.
`0b1`	This control does not cause any instructions to be trapped.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When HCR_EL2.TGE is 1, this control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EL1PCTEN, bit [10]

When HCR_EL2.TGE is 0, traps EL0 and EL1 accesses to the EL1 physical counter registers to EL2 when EL2 is enabled in the current Security state, as follows:

- In AArch64 state, accesses to CNTPCT_EL0 and CNTPCTSS_EL0 are trapped to EL2 and reported with EC syndrome value `0x18`, unless they are trapped by CNTKCTL_EL1.EL0PCTEN.
- In AArch32 state, MRRC or MCRR accesses to CNTPCT are trapped to EL2 and reported with EC syndrome value `0x04`, unless they are trapped by CNTKCTL_EL1.EL0PCTEN or CNTKCTL.PL0PCTEN.

EL1PCTEN	Meaning
`0b0`	EL0 and EL1 accesses to the specified registers are trapped to EL2.
`0b1`	This control does not cause any instructions to be trapped.

When HCR_EL2.TGE is 1, this control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EL0PTEN, bit [9]

When HCR_EL2.TGE is 1, traps EL0 accesses to the physical timer registers to EL2, as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2 and reported with EC syndrome value `0x18`:
  - CNTP_CTL_EL0.
  - CNTP_CVAL_EL0.
  - CNTP_TVAL_EL0.
- In AArch32 state, MRC and MCR accesses to the following registers are trapped and reported using EC syndrome value `0x03` and MRRC and MCRR accesses are trapped and reported using EC syndrome value `0x04`:
  - CNTP_CTL.
  - CNTP_CVAL.
  - CNTP_TVAL.

EL0PTEN	Meaning
`0b0`	EL0 accesses to the specified registers are trapped to EL2.
`0b1`	This control does not cause any instructions to be trapped.

When HCR_EL2.TGE is 0, this control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EL0VTEN, bit [8]

When HCR_EL2.TGE is 1, traps EL0 accesses to the virtual timer registers to EL2 as follows:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- In AArch64 state, accesses to the following registers are trapped to EL2 and reported with EC syndrome value `0x18`:
  - CNTV_CTL_EL0.
  - CNTV_CVAL_EL0.
  - CNTV_TVAL_EL0.
- In AArch32 state, MRC and MCR accesses to the following registers are trapped and reported using EC syndrome value `0x03` and MRRC and MCRR accesses are trapped and reported using EC syndrome value `0x04`:
  - CNTV_CTL.
  - CNTV_CVAL.
  - CNTV_TVAL.

EL0VTEN	Meaning
`0b0`	EL0 accesses to the specified registers are trapped to EL2.
`0b1`	This control does not cause any instructions to be trapped.

When HCR_EL2.TGE is 0, this control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTI, bits [7:4]

Selects which bit of CNTPCT_EL0, as seen from EL2, is the trigger for the event stream generated from that counter when that stream is enabled.

If FEAT_ECV is implemented, and CNTHCTL_EL2.EVNTIS is 1, this field selects a trigger bit in the range 8 to 23 of CNTPCT_EL0.

Otherwise, this field selects a trigger bit in the range 0 to 15 of CNTPCT_EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTDIR, bit [3]

Controls which transition of the CNTPCT_EL0 trigger bit, as seen from EL2 and defined by EVNTI, generates an event when the event stream is enabled.

EVNTDIR	Meaning
`0b0`	A 0 to 1 transition of the trigger bit triggers an event.
`0b1`	A 1 to 0 transition of the trigger bit triggers an event.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTEN, bit [2]

Enables the generation of an event stream from CNTPCT_EL0 as seen from EL2.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EVNTEN	Meaning
0b0	Disables the event stream.
0b1	Enables the event stream.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EL0VCTEN, bit [1]

When HCR_EL2.TGE is 1, traps EL0 accesses to the frequency register and virtual counter registers to EL2, as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2 and reported with EC syndrome value 0x18:
  - CNTVCT_EL0.
  - CNTVCTSS_EL0.
  - CNTFRQ_EL0 if CNTHCTL_EL2.EL0PCTEN is 0.
- In AArch32 state, MRC and MCR accesses to the following registers are trapped and reported using EC syndrome value 0x03 and MRRC and MCRR accesses are trapped and reported using EC syndrome value 0x04:
  - CNTVCT and if CNTHCTL.EL0PCTEN is 0, CNTFRQ.

EL0VCTEN	Meaning
0b0	EL0 accesses to the specified registers are trapped to EL2.
0b1	This control does not cause any instructions to be trapped.

If HCR_EL2.TGE is 0, the field is ignored for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EL0PCTEN, bit [0]

Traps EL0 accesses to the frequency register and physical counter registers to EL2, as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2 and reported with EC syndrome value 0x18:
  - CNTPCT_EL0.
  - CNTPCTSS_EL0.
  - CNTFRQ_EL0 if CNTHCTL_EL2.EL0VCTEN is 0.
- In AArch32 state, MRC and MCR accesses to the following registers are trapped and reported using EC syndrome value 0x03 and MRRC and MCRR accesses are trapped and reported using EC syndrome value 0x04:
  - CNTPCT and if CNTHCTL_EL2.EL0VCTEN is 0, CNTFRQ.

EL0PCTEN	Meaning
0b0	From AArch64 state: EL0 accesses to the specified registers are trapped to EL2.
0b1	This control does not cause any instructions to be trapped.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EL0PCTEN	Meaning

If HCR_EL2.TGE is 0, the control does not cause any instructions to be trapped for all purposes other than direct reads and writes of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

The following field descriptions apply in all Armv8.0 implementations.

The descriptions also explain the behavior when EL3 is implemented and EL2 is not implemented.

Bits [63:20]

Reserved, RES0.

CNTPMASK, bit [19]

When FEAT_RME is implemented:

CNTPMASK	Meaning
0b0	This control has no effect on CNTP_CTL_EL0.IMASK.
0b1	CNTP_CTL_EL0.IMASK behaves as if set to 1 for all purposes other than a direct read of the field.

This bit is RES0 in Non-secure and Secure state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CNTVMASK, bit [18]

When FEAT_RME is implemented:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CNTVMASK	Meaning
0b0	This control has no effect on CNTV_CTL_EL0.IMASK.
0b1	CNTV_CTL_EL0.IMASK behaves as if set to 1 for all purposes other than a direct read of the field.

This bit is RES0 in Non-secure and Secure state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EVNTIS, bit [17]

When FEAT_ECV is implemented:

Controls the scale of the generation of the event stream.

EVNTIS	Meaning
0b0	The CNTHCTL_EL2.EVNTI field applies to CNTPCT_EL0[15:0].
0b1	The CNTHCTL_EL2.EVNTI field applies to CNTPCT_EL0[23:8].

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1NVVCT, bit [16]

When FEAT_ECV is implemented:

When the Effective value of HCR_EL2.{NV2, NV1, NV} is {1, 0, 1}, traps EL1 accesses to the specified EL1 virtual timer registers using the EL02 descriptors to EL2 as follows:

Accesses to CNTV_CTL_EL02 and CNTV_CVAL_EL02 are trapped to EL2 and reported with EC syndrome value 0x18.

EL1NVVCT	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	EL1 accesses to the specified registers are trapped to EL2.

If the Effective value of HCR_EL2.{NV2, NV1, NV} is not {1, 0, 1}, this control does not cause any instructions to be trapped.

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1NVPCT, bit [15]

When FEAT_ECV is implemented:

When the Effective value of HCR_EL2.{NV2, NV1, NV} is {1, 0, 1}, traps EL1 accesses to the specified EL1 physical timer registers using the EL02 descriptors to EL2 as follows:

Accesses to CNTP_CTL_EL02 and CNTP_CVAL_EL02 are trapped to EL2 and reported with EC syndrome value `0x18`.

EL1NVPCT	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	EL1 accesses to the specified registers are trapped to EL2.

If the Effective value of HCR_EL2.{NV2, NV1, NV} is not {1, 0, 1}, this control does not cause any instructions to be trapped.

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1TVCT, bit [14]

When FEAT_ECV is implemented:

Traps EL0 and EL1 accesses to the EL1 virtual counter registers to EL2, when EL2 is enabled in the current Security state, as follows:

- In AArch64 state, accesses to CNTVCT_EL0 and CNTVCTSS_EL0 are trapped to EL2 and reported using EC syndrome value `0x18`, unless they are trapped by CNTKCTL_EL1.EL0VCTEN.
- In AArch32 state, accesses to CNTVCT are trapped to EL2 and reported using EC syndrome value `0x04`, unless they are trapped by CNTKCTL_EL1.EL0VCTEN or CNTKCTL.PL0VCTEN.

EL1TVCT	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	EL0 and EL1 accesses to the specified registers are trapped to EL2.

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1TVT, bit [13]

When FEAT_ECV is implemented:

If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, then traps EL0 and EL1 accesses to the EL1 virtual timer registers to EL2, when EL2 is enabled for the current Security state, as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2 and reported with EC syndrome value `0x18`, unless they are trapped by CNTKCTL_EL1.EL0VTEN:
  - CNTV_CTL_EL0.
  - CNTV_CVAL_EL0.
  - CNTV_TVAL_EL0.
- In AArch32 state, MRC and MCR accesses to the following registers are trapped and reported using EC syndrome value `0x03` and MRRC and MCRR accesses are trapped and reported using EC syndrome value `0x04`, unless they are trapped by CNTKCTL_EL1.EL0VTEN or CNTKCTL.PL0VTEN:
  - CNTV_CTL.
  - CNTV_CVAL.
  - CNTV_TVAL.

EL1TVT	Meaning
`0b0`	This control does not cause any instructions to be trapped.
`0b1`	EL0 and EL1 accesses to the specified registers are trapped to EL2.

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ECV, bit [12]

When FEAT_ECV_POFF is implemented:

Enables the Enhanced Counter Virtualization functionality registers.

When the Enhanced Counter Virtualization is enabled, the behavior is as follows:

- An MRS to CNTPCT_EL0 from either EL0 or EL1 that is not trapped will return the value (`PCount`<63:0> - CNTPOFF_EL2<63:0>).
- The EL1 physical timer interrupt is triggered when ((`PCount`<63:0> - CNTPOFF_EL2<63:0>) - `PCVal`<63:0>) is greater than or equal to 0.

`PCount` is the physical count returned when CNTPCT_EL0 is read from EL2 or EL3.

`PCVal`<63:0> is the EL1 physical timer compare value for this timer.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ECV	Meaning
0b0	Enhanced Counter Virtualization functionality is disabled.
0b1	Enhanced Counter Virtualization functionality is enabled.

When SCR_EL3.{NS, EEL2} is {0, 0} or if FEAT_ECV_POFF is not implemented, the Effective value of this field is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [11:8]

Reserved, RES0.

EVNTI, bits [7:4]

Selects which bit of CNTPCT_EL0, as seen from EL2,is the trigger for the event stream generated from that counter when that stream is enabled.

If FEAT_ECV is implemented, and CNTHCTL_EL2.EVNTIS is 1, this field selects a trigger bit in the range 8 to 23 of CNTPCT_EL0.

Otherwise, this field selects a trigger bit in the range 0 to 15 of CNTPCT_EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTDIR, bit [3]

Controls which transition of the CNTPCT_EL0 trigger bit, as seen from EL2 and defined by EVNTI, generates an event when the event stream is enabled.

EVNTDIR	Meaning
0b0	A 0 to 1 transition of the trigger bit triggers an event.
0b1	A 1 to 0 transition of the trigger bit triggers an event.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTEN, bit [2]

Enables the generation of an event stream from CNTPCT_EL0 as seen from EL2.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EVNTEN	Meaning
0b0	Disables the event stream.
0b1	Enables the event stream.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EL1PCEN, bit [1]

Traps EL0 and EL1 accesses to the EL1 physical timer registers to EL2 when EL2 is enabled in the current Security state, as follows:

- In AArch64 state, accesses to CNTP_CTL_EL0, CNTP_CVAL_EL0, CNTP_TVAL_EL0 are trapped to EL2, reported using EC syndrome value 0x18, unless they are trapped by CNTKCTL_EL1.EL0PTEN.
- In AArch32 state, MRC or MCR accesses to the following registers are trapped to EL2 and reported using EC syndrome value 0x03, and MRRC and MCRR accesses are trapped to EL2, reported using EC syndrome value 0x04, unless they are trapped by CNTKCTL_EL1.EL0PTEN or CNTKCTL.PL0PTEN:
  - CNTP_CTL, CNTP_CVAL, CNTP_TVAL.

EL1PCEN	Meaning
0b0	EL0 and EL1 accesses to the specified registers are trapped to EL2.
0b1	This control does not cause any instructions to be trapped.

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 1 other than for the purpose of a direct read.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EL1PCTEN, bit [0]

Traps EL0 and EL1 accesses to the EL1 physical counter registers to EL2 when EL2 is enabled in the current Security state, as follows:

- In AArch64 state, accesses to CNTPCT_EL0 and CNTPCTSS_EL0 are trapped to EL2 and reported using EC syndrome value 0x18, unless they are trapped by CNTKCTL_EL1.EL0PCTEN.
- In AArch32 state, MRRC or MCRR accesses to CNTPCT are trapped to EL2 and reported using EC syndrome value 0x04, unless they are trapped by CNTKCTL_EL1.EL0PCTEN or CNTKCTL.PL0PCTEN.

EL1PCTEN	Meaning
0b0	EL0 and EL1 accesses to the specified registers are trapped to EL2.
0b1	This control does not cause any instructions to be trapped.

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 1 other than for the purpose of a direct read.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing CNTHCTL_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name `CNTHCTL_EL2` or `CNTKCTL_EL1` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTHCTL_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = CNTHCTL_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTHCTL_EL2;
```

```
MSR CNTHCTL_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHCTL_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTHCTL_EL2 = X[t, 64];
```

```
When FEAT_VHE is implemented MRS <Xt>, CNTKCTL_EL1
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b000	0b1110	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    X[t, 64] = CNTKCTL_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = CNTHCTL_EL2_VHE(CNTHCTL_EL2);
    else
        X[t, 64] = CNTKCTL_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTKCTL_EL1;
```

When FEAT_VHE is implemented MSR CNTKCTL_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1110	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    CNTKCTL_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        CNTHCTL_EL2 = CNTHCTL_EL2_VHE(X[t, 64]);
    else
        CNTKCTL_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTKCTL_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.10.3 CNTHP_CTL_EL2, Counter-timer Hypervisor Physical Timer Control Register

The CNTHP_CTL_EL2 characteristics are:

Purpose

Control register for the EL2 physical timer.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

AArch64 System register CNTHP_CTL_EL2 bits [31:0] are architecturally mapped to AArch32 System register CNTHP_CTL[31:0].

This register is present only when (HaveEL(EL3) || (((!HaveEL(EL3)) && HaveEL(EL2)) && (!IsFeatureImplemented(FEAT_SEL2)))) && IsFeatureImplemented(FEAT_AA64). Otherwise, direct accesses to CNTHP_CTL_EL2 are UNDEFINED.

Attributes

CNTHP_CTL_EL2 is a 64-bit register.

Field descriptions

Bits [63:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

ISTATUS	Meaning
0b0	Timer condition is not met.
0b1	Timer condition is met.

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

IMASK	Meaning
0b0	Timer interrupt is not masked by the IMASK bit.
0b1	Timer interrupt is masked by the IMASK bit.

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

ENABLE	Meaning
0b0	Timer disabled.
0b1	Timer enabled.

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTHP_TVAL_EL2 continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHP_CTL_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name `CNTHP_CTL_EL2` or `CNTP_CTL_EL0` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTHP_CTL_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0010	0b001

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !((HaveEL(EL3) || (!HaveEL(EL3) && HaveEL(EL2) && !IsFeatureImplemented(FEAT_SEL2))) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = CNTHP_CTL_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTHP_CTL_EL2;
```

MSR CNTHP_CTL_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0010	0b001

```
if !((HaveEL(EL3) || (!HaveEL(EL3) && HaveEL(EL2) && !IsFeatureImplemented(FEAT_SEL2))) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHP_CTL_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTHP_CTL_EL2 = X[t, 64];
```

MRS <Xt>, CNTP_CTL_EL0

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHPS_CTL_EL2;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHP_CTL_EL2;
    else
        X[t, 64] = CNTP_CTL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x180];
    else
        X[t, 64] = CNTP_CTL_EL0;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHPS_CTL_EL2;
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHP_CTL_EL2;
    else
        X[t, 64] = CNTP_CTL_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTP_CTL_EL0;
```

```
MSR CNTP_CTL_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CTL_EL2 = X[t, 64];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CTL_EL2 = X[t, 64];
    else
        CNTP_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x180] = X[t, 64];
    else
        CNTP_CTL_EL0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CTL_EL2 = X[t, 64];
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CTL_EL2 = X[t, 64];
    else
        CNTP_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTP_CTL_EL0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.10.4 CNTHP_CVAL_EL2, Counter-timer Physical Timer CompareValue Register (EL2)

The CNTHP_CVAL_EL2 characteristics are:

Purpose

Holds the compare value for the EL2 physical timer.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

AArch64 System register CNTHP_CVAL_EL2 bits [63:0] are architecturally mapped to AArch32 System register CNTHP_CVAL[63:0].

This register is present only when (HaveEL(EL3) || (((!HaveEL(EL3)) && HaveEL(EL2)) && (!IsFeatureImplemented(FEAT_SEL2)))) && IsFeatureImplemented(FEAT_AA64). Otherwise, direct accesses to CNTHP_CVAL_EL2 are UNDEFINED.

Attributes

CNTHP_CVAL_EL2 is a 64-bit register.

Field descriptions

CompareValue, bits [63:0]

Holds the EL2 physical timer CompareValue.

When CNTHP_CTL_EL2.ENABLE is 1, and TimerConditionMet is TRUE for the EL2 physical timer, the timer condition is met and all of the following are true:

- CNTHP_CTL_EL2.ISTATUS is set to 1.
- If CNTHP_CTL_EL2.IMASK is 0, an interrupt is generated.

TimerConditionMet is defined by 'Operation of the CompareValue views of the timers'.

The CompareValue view of the timer acts like a 64-bit upcounter timer.

When CNTHP_CTL_EL2.ENABLE is 0, the timer condition is not met, but CNTPCT_EL0 continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHP_CVAL_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name `CNTHP_CVAL_EL2` or `CNTP_CVAL_EL0` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

MRS <Xt>, CNTHP_CVAL_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0010	0b010

```
if !((HaveEL(EL3) || (!HaveEL(EL3) && HaveEL(EL2) && !IsFeatureImplemented(FEAT_SEL2))) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = CNTHP_CVAL_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTHP_CVAL_EL2;
```

MSR CNTHP_CVAL_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0010	0b010

```
if !((HaveEL(EL3) || (!HaveEL(EL3) && HaveEL(EL2) && !IsFeatureImplemented(FEAT_SEL2))) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHP_CVAL_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTHP_CVAL_EL2 = X[t, 64];
```

MRS <Xt>, CNTP_CVAL_EL0

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b010

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHPS_CVAL_EL2;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHP_CVAL_EL2;
    else
        X[t, 64] = CNTP_CVAL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x178];
    else
        X[t, 64] = CNTP_CVAL_EL0;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHPS_CVAL_EL2;
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHP_CVAL_EL2;
    else
        X[t, 64] = CNTP_CVAL_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTP_CVAL_EL0;
```

MSR CNTP_CVAL_EL0, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        CNTHPS_CVAL_EL2 = X[t, 64];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CVAL_EL2 = X[t, 64];
    else
        CNTP_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x178] = X[t, 64];
    else
        CNTP_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CVAL_EL2 = X[t, 64];
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CVAL_EL2 = X[t, 64];
    else
        CNTP_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTP_CVAL_EL0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.10.5 CNTHP_TVAL_EL2, Counter-timer Physical Timer TimerValue Register (EL2)

The CNTHP_TVAL_EL2 characteristics are:

Purpose

Holds the timer value for the EL2 physical timer.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

AArch64 System register CNTHP_TVAL_EL2 bits [31:0] are architecturally mapped to AArch32 System register CNTHP_TVAL[31:0].

This register is present only when (HaveEL(EL3) || (((!HaveEL(EL3)) && HaveEL(EL2)) && (!IsFeatureImplemented(FEAT_SEL2)))) && IsFeatureImplemented(FEAT_AA64). Otherwise, direct accesses to CNTHP_TVAL_EL2 are UNDEFINED.

Attributes

CNTHP_TVAL_EL2 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

TimerValue, bits [31:0]

The TimerValue view of the EL2 physical timer.

On a read of this register:

- If CNTHP_CTL_EL2.ENABLE is 0, the value returned is UNKNOWN.
- If CNTHP_CTL_EL2.ENABLE is 1, the value returned is (CNTHP_CVAL_EL2 - CNTPCT_EL0).

On a write of this register, CNTHP_CVAL_EL2 is set to (CNTPCT_EL0 + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTHP_CTL_EL2.ENABLE is 1, the timer condition is met when (CNTPCT_EL0 - CNTHP_CVAL_EL2) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTHP_CTL_EL2.ISTATUS is set to 1.
- If CNTHP_CTL_EL2.IMASK is 0, an interrupt is generated.

When CNTHP_CTL_EL2.ENABLE is 0, the TimerValue cannot be read but continues to decrement. When the timer is enabled, the TimerValue represents the elapsed time whether that time was spent enabled or disabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing CNTHP_TVAL_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name `CNTHP_TVAL_EL2` or `CNTP_TVAL_EL0` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTHP_TVAL_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0010	0b000

```
if !((HaveEL(EL3) || (!HaveEL(EL3) && HaveEL(EL2) && !IsFeatureImplemented(FEAT_SEL2))) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if CNTHP_CTL_EL2.ENABLE == '0' then
        X[t, 64] = bits(64) UNKNOWN;
    else
        X[t, 64] = ZeroExtend((CNTHP_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL3 then
    if CNTHP_CTL_EL2.ENABLE == '0' then
        X[t, 64] = bits(64) UNKNOWN;
    else
        X[t, 64] = ZeroExtend((CNTHP_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
```

```
MSR CNTHP_TVAL_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0010	0b000

```
if !((HaveEL(EL3) || (!HaveEL(EL3) && HaveEL(EL2) && !IsFeatureImplemented(FEAT_SEL2))) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHP_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    CNTHP_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, CNTP_TVAL_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        if CNTHPS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHPS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        if CNTHP_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHP_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.ECVEn == '1')
     ↪&& CNTHCTL_EL2.ECV == '1' && !ELIsInHost(EL0) then
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - (PhysicalCountInt() - CNTPOFF_EL2))<31:0>, 64);
    else
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.ECVEn == '1')
     ↪&& CNTHCTL_EL2.ECV == '1' then
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - (PhysicalCountInt() - CNTPOFF_EL2))<31:0>, 64);
    else
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        if CNTHPS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            X[t, 64] = ZeroExtend((CNTHPS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
        elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
            if CNTHP_CTL_EL2.ENABLE == '0' then
                X[t, 64] = bits(64) UNKNOWN;
            else
                X[t, 64] = ZeroExtend((CNTHP_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
        else
            if CNTP_CTL_EL0.ENABLE == '0' then
                X[t, 64] = bits(64) UNKNOWN;
            else
                X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
    elsif PSTATE.EL == EL3 then
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
```

MSR CNTP_TVAL_EL0, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.ECVEn == '1')
         ↪&& CNTHCTL_EL2.ECV == '1' && !ELIsInHost(EL0) then
        CNTP_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTPOFF_EL2;
    else
        CNTP_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.ECVEn == '1')
         ↪&& CNTHCTL_EL2.ECV == '1' then
        CNTP_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTPOFF_EL2;
    else
        CNTP_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    else
        CNTP_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    CNTP_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.10.6 CNTHPS_CTL_EL2, Counter-timer Secure Physical Timer Control Register (EL2)

The CNTHPS_CTL_EL2 characteristics are:

Purpose

Control register for the Secure EL2 physical timer.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

AArch64 System register CNTHPS_CTL_EL2 bits [31:0] are architecturally mapped to AArch32 System register CNTHPS_CTL[31:0].

This register is present only when FEAT_SEL2 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to CNTHPS_CTL_EL2 are UNDEFINED.

Attributes

CNTHPS_CTL_EL2 is a 64-bit register.

Field descriptions

Bits [63:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

ISTATUS	Meaning
0b0	Timer condition is not met.
0b1	Timer condition is met.

When the value of the CNTHPS_CTL_EL2.ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the CNTHPS_CTL_EL2.ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

IMASK	Meaning
0b0	Timer interrupt is not masked by the IMASK bit.
0b1	Timer interrupt is masked by the IMASK bit.

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

ENABLE	Meaning
0b0	Timer disabled.
0b1	Timer enabled.

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTHPS_TVAL_EL2 continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHPS_CTL_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTHPS_CTL_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0101	0b001

```
if !(IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    else
        X[t, 64] = CNTHPS_CTL_EL2;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        X[t, 64] = CNTHPS_CTL_EL2;
```

MSR CNTHPS_CTL_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0101	0b001

```
if !(IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    else
        CNTHPS_CTL_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        CNTHPS_CTL_EL2 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, CNTP_CTL_EL0

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
             ↪then
            X[t, 64] = CNTHPS_CTL_EL2;
        elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
            X[t, 64] = CNTHP_CTL_EL2;
        else
            X[t, 64] = CNTP_CTL_EL0;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EffectiveHCR_EL2_NVx() IN {'111'} then
            X[t, 64] = NVMem[0x180];
        else
            X[t, 64] = CNTP_CTL_EL0;
    elsif PSTATE.EL == EL2 then
        if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
            X[t, 64] = CNTHPS_CTL_EL2;
        elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
            X[t, 64] = CNTHP_CTL_EL2;
        else
            X[t, 64] = CNTP_CTL_EL0;
    elsif PSTATE.EL == EL3 then
        X[t, 64] = CNTP_CTL_EL0;
```

When FEAT_VHE is implemented MSR CNTP_CTL_EL0, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b001

```
    if !IsFeatureImplemented(FEAT_AA64) then
        UNDEFINED;
    elsif PSTATE.EL == EL0 then
        if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
            if EL2Enabled() && HCR_EL2.TGE == '1' then
                AArch64.SystemAccessTrap(EL2, 0x18);
            else
                AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
             ↪then
            CNTHPS_CTL_EL2 = X[t, 64];
        elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
            CNTHP_CTL_EL2 = X[t, 64];
        else
            CNTP_CTL_EL0 = X[t, 64];
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x180] = X[t, 64];
    else
        CNTP_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CTL_EL2 = X[t, 64];
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CTL_EL2 = X[t, 64];
    else
        CNTP_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTP_CTL_EL0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.10.7 CNTHPS_CVAL_EL2, Counter-timer Secure Physical Timer CompareValue Register (EL2)

The CNTHPS_CVAL_EL2 characteristics are:

Purpose

Holds the compare value for the Secure EL2 physical timer.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

AArch64 System register CNTHPS_CVAL_EL2 bits [31:0] are architecturally mapped to AArch32 System register CNTHPS_CVAL[31:0].

This register is present only when EL2 is implemented, FEAT_SEL2 is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to CNTHPS_CVAL_EL2 are UNDEFINED.

Attributes

CNTHPS_CVAL_EL2 is a 64-bit register.

Field descriptions

CompareValue, bits [63:0]

Holds the EL2 physical timer CompareValue.

When CNTHPS_CTL_EL2.ENABLE is 1, and TimerConditionMet is TRUE for the Secure EL2 physical timer, the timer condition is met and all of the following are true:

- CNTHPS_CTL_EL2.ISTATUS is set to 1.
- If CNTHPS_CTL_EL2.IMASK is 0, an interrupt is generated.

TimerConditionMet is defined by ‘Operation of the CompareValue views of the timers’.

The CompareValue view of the timer acts like a 64-bit upcounter timer.

When CNTHPS_CTL_EL2.ENABLE is 0, the timer condition is not met, but CNTPCT_EL0 continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHPS_CVAL_EL2

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, CNTHPS_CVAL_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0101	0b010

```
if !(HaveEL(EL2) && IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    else
        X[t, 64] = CNTHPS_CVAL_EL2;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        X[t, 64] = CNTHPS_CVAL_EL2;
```

```
MSR CNTHPS_CVAL_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0101	0b010

```
if !(HaveEL(EL2) && IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    else
        CNTHPS_CVAL_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        CNTHPS_CVAL_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_VHE is implemented MRS <Xt>, CNTP_CVAL_EL0

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
     ↪then
        X[t, 64] = CNTHPS_CVAL_EL2;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHP_CVAL_EL2;
    else
        X[t, 64] = CNTP_CVAL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x178];
    else
        X[t, 64] = CNTP_CVAL_EL0;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHPS_CVAL_EL2;
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHP_CVAL_EL2;
    else
        X[t, 64] = CNTP_CVAL_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTP_CVAL_EL0;
```

When FEAT_VHE is implemented MSR CNTP_CVAL_EL0, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
     ↪then
        CNTHPS_CVAL_EL2 = X[t, 64];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CVAL_EL2 = X[t, 64];
    else
        CNTP_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x178] = X[t, 64];
    else
        CNTP_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CVAL_EL2 = X[t, 64];
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CVAL_EL2 = X[t, 64];
    else
        CNTP_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTP_CVAL_EL0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.10.8 CNTHPS_TVAL_EL2, Counter-timer Secure Physical Timer TimerValue Register (EL2)

The CNTHPS_TVAL_EL2 characteristics are:

Purpose

Holds the timer value for the Secure EL2 physical timer.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

AArch64 System register CNTHPS_TVAL_EL2 bits [31:0] are architecturally mapped to AArch32 System register CNTHPS_TVAL[31:0].

This register is present only when EL2 is implemented, FEAT_SEL2 is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to CNTHPS_TVAL_EL2 are UNDEFINED.

Attributes

CNTHPS_TVAL_EL2 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

TimerValue, bits [31:0]

The TimerValue view of the EL2 physical timer.

On a read of this register:

- If CNTHPS_CTL_EL2.ENABLE is 0, the value returned is UNKNOWN.
- If CNTHPS_CTL_EL2.ENABLE is 1, the value returned is (CNTHPS_CVAL_EL2 - CNTPCT_EL0).

On a write of this register, CNTHPS_CVAL_EL2 is set to (CNTPCT_EL0 + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTHPS_CTL_EL2.ENABLE is 1, the timer condition is met when (CNTPCT_EL0 - CNTHPS_CVAL_EL2) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTHPS_CTL_EL2.ISTATUS is set to 1.
- If CNTHPS_CTL_EL2.IMASK is 0, an interrupt is generated.

When CNTHPS_CTL_EL2.ENABLE is 0, the TimerValue cannot be read but continues to decrement. When the timer is enabled, the TimerValue represents the elapsed time whether that time was spent enabled or disabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHPS_TVAL_EL2

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, CNTHPS_TVAL_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0101	0b000

```
if !(HaveEL(EL2) && IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    else
        if CNTHPS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHPS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        if CNTHPS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHPS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
```

```
MSR CNTHPS_TVAL_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0101	0b000

```
if !(HaveEL(EL2) && IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    else
        CNTHPS_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL3 then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        CNTHPS_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
```

When FEAT_VHE is implemented MRS <Xt>, CNTP_TVAL_EL0

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
     ↪then
        if CNTHPS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHPS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        if CNTHP_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHP_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.ECVEn ==
     ↪'1') && CNTHCTL_EL2.ECV == '1' && !ELIsInHost(EL0) then
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - (PhysicalCountInt() - CNTPOFF_EL2))<31:0>, 64);
    else
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.ECVEn ==
     ↪'1') && CNTHCTL_EL2.ECV == '1' then
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - (PhysicalCountInt() - CNTPOFF_EL2))<31:0>, 64);
    else
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        if CNTHPS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHPS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        if CNTHP_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHP_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    else
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL3 then
    if CNTP_CTL_EL0.ENABLE == '0' then
        X[t, 64] = bits(64) UNKNOWN;
    else
        X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
```

When FEAT_VHE is implemented MSR CNTP_TVAL_EL0, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2)
         ↪then
        CNTHPS_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.ECVEn ==
         ↪'1') && CNTHCTL_EL2.ECV == '1' && !ELIsInHost(EL0) then
        CNTP_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTPOFF_EL2;
    else
        CNTP_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.ECVEn ==
         ↪'1') && CNTHCTL_EL2.ECV == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        CNTP_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTPOFF_EL2;
    else
        CNTP_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    else
        CNTP_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    CNTP_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.10.9 CNTHV_CTL_EL2, Counter-timer Virtual Timer Control Register (EL2)

The CNTHV_CTL_EL2 characteristics are:

Purpose

Control register for the EL2 virtual timer.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

AArch64 System register CNTHV_CTL_EL2 bits [31:0] are architecturally mapped to AArch32 System register CNTHV_CTL[31:0].

This register is present only when (IsFeatureImplemented(FEAT_VHE) && (HaveEL(EL3) || ((!HaveEL(EL3)) && (!IsFeatureImplemented(FEAT_SEL2))))) && IsFeatureImplemented(FEAT_AA64). Otherwise, direct accesses to CNTHV_CTL_EL2 are UNDEFINED.

Attributes

CNTHV_CTL_EL2 is a 64-bit register.

Field descriptions

Bits [63:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

ISTATUS	Meaning
0b0	Timer condition is not met.
0b1	Timer condition is met.

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

IMASK	Meaning
0b0	Timer interrupt is not masked by the IMASK bit.
0b1	Timer interrupt is masked by the IMASK bit.

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

ENABLE	Meaning
0b0	Timer disabled.
0b1	Timer enabled.

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTHV_TVAL_EL2 continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHV_CTL_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name `CNTHV_CTL_EL2` or `CNTV_CTL_EL0` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTHV_CTL_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0011	0b001

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_VHE) && (HaveEL(EL3) || (!HaveEL(EL3) &&
     ↪!IsFeatureImplemented(FEAT_SEL2))) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = CNTHV_CTL_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTHV_CTL_EL2;
```

MSR CNTHV_CTL_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0011	0b001

```
if !(IsFeatureImplemented(FEAT_VHE) && (HaveEL(EL3) || (!HaveEL(EL3) &&
     ↪!IsFeatureImplemented(FEAT_SEL2))) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHV_CTL_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTHV_CTL_EL2 = X[t, 64];
```

MRS <Xt>, CNTV_CTL_EL0

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT ==
         ↪'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHVS_CTL_EL2;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHV_CTL_EL2;
    else
        X[t, 64] = CNTV_CTL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x170];
    else
        X[t, 64] = CNTV_CTL_EL0;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHVS_CTL_EL2;
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHV_CTL_EL2;
    else
        X[t, 64] = CNTV_CTL_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTV_CTL_EL0;
```

```
MSR CNTV_CTL_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT ==
     ↪'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CTL_EL2 = X[t, 64];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CTL_EL2 = X[t, 64];
    else
        CNTV_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x170] = X[t, 64];
    else
        CNTV_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CTL_EL2 = X[t, 64];
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CTL_EL2 = X[t, 64];
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        CNTV_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTV_CTL_EL0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.10.10 CNTHV_CVAL_EL2, Counter-timer Virtual Timer CompareValue Register (EL2)

The CNTHV_CVAL_EL2 characteristics are:

Purpose

Holds the compare value for the EL2 virtual timer.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

AArch64 System register CNTHV_CVAL_EL2 bits [63:0] are architecturally mapped to AArch32 System register CNTHV_CVAL[63:0].

This register is present only when (IsFeatureImplemented(FEAT_VHE) && (HaveEL(EL3) || ((!HaveEL(EL3)) && (!IsFeatureImplemented(FEAT_SEL2))))) && IsFeatureImplemented(FEAT_AA64). Otherwise, direct accesses to CNTHV_CVAL_EL2 are UNDEFINED.

Attributes

CNTHV_CVAL_EL2 is a 64-bit register.

Field descriptions

CompareValue, bits [63:0]

Holds the EL2 virtual timer CompareValue.

When CNTHV_CTL_EL2.ENABLE is 1, and TimerConditionMet is TRUE for the EL2 virtual timer, the timer condition is met and all of the following are true:

- CNTHV_CTL_EL2.ISTATUS is set to 1.
- If CNTHV_CTL_EL2.IMASK is 0, an interrupt is generated.

TimerConditionMet is defined by 'Operation of the CompareValue views of the timers'.

The CompareValue view of the timer acts like a 64-bit upcounter timer.

When CNTHV_CTL_EL2.ENABLE is 0, the timer condition is not met, but CNTVCT_EL0 continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHV_CVAL_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name `CNTHV_CVAL_EL2` or `CNTV_CVAL_EL0` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

MRS <Xt>, CNTHV_CVAL_EL2

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0011	0b010

```
if !(IsFeatureImplemented(FEAT_VHE) && (HaveEL(EL3) || (!HaveEL(EL3) &&
     ↪!IsFeatureImplemented(FEAT_SEL2))) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = CNTHV_CVAL_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTHV_CVAL_EL2;
```

MSR CNTHV_CVAL_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0011	0b010

```
if !(IsFeatureImplemented(FEAT_VHE) && (HaveEL(EL3) || (!HaveEL(EL3) &&
     ↪!IsFeatureImplemented(FEAT_SEL2))) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHV_CVAL_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTHV_CVAL_EL2 = X[t, 64];
```

MRS <Xt>, CNTV_CVAL_EL0

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b010

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT ==
         ↪'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHVS_CVAL_EL2;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHV_CVAL_EL2;
    else
        X[t, 64] = CNTV_CVAL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x168];
    else
        X[t, 64] = CNTV_CVAL_EL0;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHVS_CVAL_EL2;
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHV_CVAL_EL2;
    else
        X[t, 64] = CNTV_CVAL_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTV_CVAL_EL0;
```

MSR CNTV_CVAL_EL0, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT ==
         ↪'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = X[t, 64];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CVAL_EL2 = X[t, 64];
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        CNTV_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x168] = X[t, 64];
    else
        CNTV_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = X[t, 64];
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CVAL_EL2 = X[t, 64];
    else
        CNTV_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTV_CVAL_EL0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.10.11 CNTHV_TVAL_EL2, Counter-timer Virtual Timer TimerValue Register (EL2)

The CNTHV_TVAL_EL2 characteristics are:

Purpose

Holds the timer value for the EL2 virtual timer.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

AArch64 System register CNTHV_TVAL_EL2 bits [31:0] are architecturally mapped to AArch32 System register CNTHV_TVAL[31:0].

This register is present only when (IsFeatureImplemented(FEAT_VHE) && (HaveEL(EL3) || ((!HaveEL(EL3)) && (!IsFeatureImplemented(FEAT_SEL2))))) && IsFeatureImplemented(FEAT_AA64). Otherwise, direct accesses to CNTHV_TVAL_EL2 are UNDEFINED.

Attributes

CNTHV_TVAL_EL2 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

TimerValue, bits [31:0]

The TimerValue view of the EL2 virtual timer.

On a read of this register:

- If CNTHV_CTL_EL2.ENABLE is 0, the value returned is UNKNOWN.
- If CNTHV_CTL_EL2.ENABLE is 1, the value returned is (CNTHV_CVAL_EL2 - CNTVCT_EL0).

On a write of this register, CNTHV_CVAL_EL2 is set to (CNTVCT_EL0 + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTHV_CTL_EL2.ENABLE is 1, the timer condition is met when (CNTVCT_EL0 - CNTHV_CVAL_EL2) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTHV_CTL_EL2.ISTATUS is set to 1.
- If CNTHV_CTL_EL2.IMASK is 0, an interrupt is generated.

When CNTHV_CTL_EL2.ENABLE is 0, the TimerValue cannot be read but continues to decrement. When the timer is enabled, the TimerValue represents the elapsed time whether that time was spent enabled or disabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing CNTHV_TVAL_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name `CNTHV_TVAL_EL2` or `CNTV_TVAL_EL0` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTHV_TVAL_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0011	0b000

```
if !(IsFeatureImplemented(FEAT_VHE) && (HaveEL(EL3) || (!HaveEL(EL3) &&
     ↪!IsFeatureImplemented(FEAT_SEL2))) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if CNTHV_CTL_EL2.ENABLE == '0' then
        X[t, 64] = bits(64) UNKNOWN;
    else
        X[t, 64] = ZeroExtend((CNTHV_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL3 then
    if CNTHV_CTL_EL2.ENABLE == '0' then
        X[t, 64] = bits(64) UNKNOWN;
    else
        X[t, 64] = ZeroExtend((CNTHV_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
```

```
MSR CNTHV_TVAL_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0011	0b000

```
if !(IsFeatureImplemented(FEAT_VHE) && (HaveEL(EL3) || (!HaveEL(EL3) &&
     ↪!IsFeatureImplemented(FEAT_SEL2))) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHV_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    CNTHV_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, CNTV_TVAL_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT ==
     ↪'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        if CNTHVS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHVS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        if CNTHV_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHV_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif HaveEL(EL2) && !ELIsInHost(EL0) then
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>, 64);
    else
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL2) then
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>, 64);
    else
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        if CNTHVS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHVS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        if CNTHV_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            X[t, 64] = ZeroExtend((CNTHV_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
        elsif !ELIsInHost(EL2) then
            if CNTV_CTL_EL0.ENABLE == '0' then
                X[t, 64] = bits(64) UNKNOWN;
            else
                X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>, 64);
        else
            if CNTV_CTL_EL0.ENABLE == '0' then
                X[t, 64] = bits(64) UNKNOWN;
            else
                X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
    elsif PSTATE.EL == EL3 then
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>, 64);
        elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) then
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF))<31:0>, 64);
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
```

```
MSR CNTV_TVAL_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT ==
     ↪'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif HaveEL(EL2) && !ELIsInHost(EL0) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF_EL2;
    else
        CNTV_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL2) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF_EL2;
    else
        CNTV_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif !ELIsInHost(EL2) then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF_EL2;
    else
        CNTV_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    if HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF_EL2;
    elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF;
    else
        CNTV_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.10.12 CNTHVS_CTL_EL2, Counter-timer Secure Virtual Timer Control Register (EL2)

The CNTHVS_CTL_EL2 characteristics are:

Purpose

Control register for the Secure EL2 virtual timer.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

AArch64 System register CNTHVS_CTL_EL2 bits [31:0] are architecturally mapped to AArch32 System register CNTHVS_CTL[31:0].

This register is present only when FEAT_SEL2 is implemented, FEAT_VHE is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to CNTHVS_CTL_EL2 are UNDEFINED.

Attributes

CNTHVS_CTL_EL2 is a 64-bit register.

Field descriptions

Bits [63:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

ISTATUS	Meaning
0b0	Timer condition is not met.
0b1	Timer condition is met.

When the value of the CNTHVS_CTL_EL2.ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

IMASK	Meaning
0b0	Timer interrupt is not masked by the IMASK bit.
0b1	Timer interrupt is masked by the IMASK bit.

For more information, see the description of the CNTHVS_CTL_EL2.ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

ENABLE	Meaning
0b0	Timer disabled.
0b1	Timer enabled.

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTHVS_TVAL_EL2 continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHVS_CTL_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTHVS_CTL_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0100	0b001

```
if !(IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_VHE) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    else
        X[t, 64] = CNTHVS_CTL_EL2;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        X[t, 64] = CNTHVS_CTL_EL2;
```

MSR CNTHVS_CTL_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0100	0b001

```
if !(IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_VHE) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    else
        CNTHVS_CTL_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        CNTHVS_CTL_EL2 = X[t, 64];
```

MRS <Xt>, CNTV_CTL_EL0

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT ==
     ↪'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHVS_CTL_EL2;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHV_CTL_EL2;
    else
        X[t, 64] = CNTV_CTL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x170];
    else
        X[t, 64] = CNTV_CTL_EL0;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHVS_CTL_EL2;
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHV_CTL_EL2;
    else
        X[t, 64] = CNTV_CTL_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTV_CTL_EL0;
```

```
MSR CNTV_CTL_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT ==
     ↪'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CTL_EL2 = X[t, 64];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CTL_EL2 = X[t, 64];
    else
        CNTV_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x170] = X[t, 64];
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        CNTV_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CTL_EL2 = X[t, 64];
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CTL_EL2 = X[t, 64];
    else
        CNTV_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTV_CTL_EL0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.10.13 CNTHVS_CVAL_EL2, Counter-timer Secure Virtual Timer CompareValue Register (EL2)

The CNTHVS_CVAL_EL2 characteristics are:

Purpose

Holds the compare value for the Secure EL2 virtual timer.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

AArch64 System register CNTHVS_CVAL_EL2 bits [63:0] are architecturally mapped to AArch32 System register CNTHVS_CVAL[63:0].

This register is present only when FEAT_SEL2 is implemented, FEAT_VHE is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to CNTHVS_CVAL_EL2 are UNDEFINED.

Attributes

CNTHVS_CVAL_EL2 is a 64-bit register.

Field descriptions

CompareValue, bits [63:0]

Holds the Secure EL2 virtual timer CompareValue.

When CNTHVS_CTL_EL2.ENABLE is 1, and TimerConditionMet is TRUE for the Secure EL2 virtual timer, the timer condition is met and all of the following are true:

- CNTHVS_CTL_EL2.ISTATUS is set to 1.
- If CNTHVS_CTL_EL2.IMASK is 0, an interrupt is generated.

TimerConditionMet is defined by ‘Operation of the CompareValue views of the timers’.

The CompareValue view of the timer acts like a 64-bit upcounter timer.

When CNTHVS_CTL_EL2.ENABLE is 0, the timer condition is not met, but CNTVCT_EL0 continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHVS_CVAL_EL2

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, CNTHVS_CVAL_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0100	0b010

```
if !(IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_VHE) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    else
        X[t, 64] = CNTHVS_CVAL_EL2;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        X[t, 64] = CNTHVS_CVAL_EL2;
```

```
MSR CNTHVS_CVAL_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0100	0b010

```
if !(IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_VHE) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    else
        CNTHVS_CVAL_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        CNTHVS_CVAL_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, CNTV_CVAL_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT ==
     ↪'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHVS_CVAL_EL2;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHV_CVAL_EL2;
    else
        X[t, 64] = CNTV_CVAL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x168];
    else
        X[t, 64] = CNTV_CVAL_EL0;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHVS_CVAL_EL2;
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHV_CVAL_EL2;
    else
        X[t, 64] = CNTV_CVAL_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTV_CVAL_EL0;
```

```
MSR CNTV_CVAL_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT ==
        ↪'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = X[t, 64];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CVAL_EL2 = X[t, 64];
    else
        CNTV_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x168] = X[t, 64];
    else
        CNTV_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = X[t, 64];
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CVAL_EL2 = X[t, 64];
    else
        CNTV_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTV_CVAL_EL0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.10.14 CNTHVS_TVAL_EL2, Counter-timer Secure Virtual Timer TimerValue Register (EL2)

The CNTHVS_TVAL_EL2 characteristics are:

Purpose

Holds the timer value for the Secure EL2 virtual timer.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

AArch64 System register CNTHVS_TVAL_EL2 bits [31:0] are architecturally mapped to AArch32 System register CNTHVS_TVAL[31:0].

This register is present only when FEAT_SEL2 is implemented, FEAT_VHE is implemented, and FEAT_AA64 is implemented. Otherwise, direct accesses to CNTHVS_TVAL_EL2 are UNDEFINED.

Attributes

CNTHVS_TVAL_EL2 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

TimerValue, bits [31:0]

The TimerValue view of the EL2 virtual timer.

On a read of this register:

- If CNTHVS_CTL_EL2.ENABLE is 0, the value returned is UNKNOWN.
- If CNTHVS_CTL_EL2.ENABLE is 1, the value returned is (CNTHVS_CVAL_EL2 - CNTVCT_EL0).

On a write of this register, CNTHVS_CVAL_EL2 is set to (CNTVCT_EL0 + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTHVS_CTL_EL2.ENABLE is 1, the timer condition is met when ((CNTVCT_EL0 - CNTHVS_CVAL_EL2) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTHVS_CTL_EL2.ISTATUS is set to 1.
- If CNTHVS_CTL_EL2.IMASK is 0, an interrupt is generated.

When CNTHVS_CTL_EL2.ENABLE is 0, the TimerValue cannot be read but continues to decrement. When the timer is enabled, the TimerValue represents the elapsed time whether that time was spent enabled or disabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHVS_TVAL_EL2

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, CNTHVS_TVAL_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_VHE) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    else
        if CNTHVS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHVS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        if CNTHVS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHVS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
```

```
MSR CNTHVS_TVAL_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_SEL2) && IsFeatureImplemented(FEAT_VHE) &&
     ↪IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsCurrentSecurityState(SS_Secure) then
        UNDEFINED;
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        CNTHVS_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        CNTHVS_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
```

MRS <Xt>, CNTV_TVAL_EL0

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT ==
     ↪'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        if CNTHVS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHVS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        if CNTHV_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHV_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif HaveEL(EL2) && !ELIsInHost(EL0) then
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>, 64);
    else
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL2) then
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>, 64);
    else
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        if CNTHVS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHVS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        if CNTHV_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHV_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif !ELIsInHost(EL2) then
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>, 64);
    else
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL3 then
    if CNTV_CTL_EL0.ENABLE == '0' then
        X[t, 64] = bits(64) UNKNOWN;
    elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
        X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>, 64);
    elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) then
        X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF))<31:0>, 64);
    else
        X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
```

```
MSR CNTV_TVAL_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT ==
     ↪'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif HaveEL(EL2) && !ELIsInHost(EL0) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF_EL2;
    else
        CNTV_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL2) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF_EL2;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    else
        CNTV_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif !ELIsInHost(EL2) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF_EL2;
    else
        CNTV_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    if HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF_EL2;
    elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF;
    else
        CNTV_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.10.15 CNTKCTL_EL1, Counter-timer Kernel Control Register

The CNTKCTL_EL1 characteristics are:

Purpose

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this register does not cause any event stream from the virtual counter to be generated, and does not control access to the counters and timers. The access to counters and timers at EL0 is controlled by CNTHCTL_EL2.

When FEAT_VHE is not implemented, or when the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, this register controls the generation of an event stream from the virtual counter, and access from EL0 to the physical counter, virtual counter, EL1 physical timers, and the virtual timer.

Configuration

AArch64 System register CNTKCTL_EL1 bits [31:0] are architecturally mapped to AArch32 System register CNTKCTL[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to CNTKCTL_EL1 are UNDEFINED.

Attributes

CNTKCTL_EL1 is a 64-bit register.

Field descriptions

Bits [63:20]

Reserved, RES0.

CNTPMASK, bit [19]

When FEAT_RME is implemented and FEAT_NV2p1 is implemented:

Reserved for software use in nested virtualization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CNTVMASK, bit [18]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When FEAT_RME is implemented and FEAT_NV2p1 is implemented:

Reserved for software use in nested virtualization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EVNTIS, bit [17]

When FEAT_ECV is implemented:

Controls the scale of the generation of the event stream.

EVNTIS	Meaning
0b0	The CNTKCTL_EL1.EVNTI field applies to CNTVCT_EL0[15:0].
0b1	The CNTKCTL_EL1.EVNTI field applies to CNTVCT_EL0[23:8].

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1NVVCT, bit [16]

When FEAT_ECV is implemented and FEAT_NV2p1 is implemented:

Reserved for software use in nested virtualization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1NVPCT, bit [15]

When FEAT_ECV is implemented and FEAT_NV2p1 is implemented:

Reserved for software use in nested virtualization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1TVCT, bit [14]

When FEAT_ECV is implemented and FEAT_NV2p1 is implemented:

Reserved for software use in nested virtualization.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1TVT, bit [13]

When FEAT_ECV is implemented and FEAT_NV2p1 is implemented:

Reserved for software use in nested virtualization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ECV, bit [12]

When FEAT_ECV is implemented and FEAT_NV2p1 is implemented:

Reserved for software use in nested virtualization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1PTEN, bit [11]

When FEAT_NV2p1 is implemented:

Reserved for software use in nested virtualization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1PCTEN, bit [10]

When FEAT_NV2p1 is implemented:

Reserved for software use in nested virtualization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL0PTEN, bit [9]

Traps EL0 accesses to the physical timer registers to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, as follows:

- In AArch64 state, the following registers are trapped and reported using EC syndrome value `0x18`:
  - CNTP_CTL_EL0, CNTP_CVAL_EL0, and CNTP_TVAL_EL0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- In AArch32 state, MRC and MCR accesses to the following registers are trapped and reported using EC syndrome value `0x03`, MRRC and MCRR accesses are trapped and reported using EC syndrome value `0x04`:
  - CNTP_CTL, CNTP_CVAL, CNTP_TVAL.

EL0PTEN	Meaning
`0b0`	EL0 accesses to the physical timer registers are trapped to EL1.
`0b1`	This control does not cause any instructions to be trapped.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EL0VTEN, bit [8]

Traps EL0 accesses to the virtual timer registers to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, as follows:

- In AArch64 state, accesses to the following registers are trapped and reported using EC syndrome value `0x18`:
  - CNTV_CTL_EL0, CNTV_CVAL_EL0, and CNTV_TVAL_EL0.
- In AArch32 state, MRC and MCR accesses to the following registers are trapped and reported using EC syndrome value `0x03`, MRRC and MCRR accesses are trapped using EC syndrome value `0x04`:
  - CNTV_CTL, CNTV_CVAL, and CNTV_TVAL.

EL0VTEN	Meaning
`0b0`	EL0 accesses to the virtual timer registers are trapped.
`0b1`	This control does not cause any instructions to be trapped.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTI, bits [7:4]

Selects which bit of CNTVCT_EL0, as seen from EL1, is the trigger for the event stream generated from that counter when that stream is enabled.

If FEAT_ECV is implemented, and CNTKCTL_EL1.EVNTIS is 1, this field selects a trigger bit in the range 8 to 23 of CNTVCT_EL0.

Otherwise, this field selects a trigger bit in the range 0 to 15 of CNTVCT_EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTDIR, bit [3]

Controls which transition of the CNTVCT_EL0 trigger bit, as seen from EL1 and defined by EVNTI, generates an event when the event stream is enabled.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EVNTDIR	Meaning
0b0	A 0 to 1 transition of the trigger bit triggers an event.
0b1	A 1 to 0 transition of the trigger bit triggers an event.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTEN, bit [2]

When FEAT_VHE is not implemented, or the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}, enables the generation of an event stream from CNTVCT_EL0 as seen from EL1.

EVNTEN	Meaning
0b0	Disables the event stream.
0b1	Enables the event stream.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this control does not enable the event stream.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EL0VCTEN, bit [1]

Traps EL0 accesses to the frequency register and virtual counter registers to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, as follows:

- In AArch64 state, accesses to the following registers are trapped and reported using EC syndrome value 0x18:
  - CNTVCT_EL0.
  - CNTVCTSS_EL0.
  - If CNTKCTL_EL1.EL0PCTEN is 0, CNTFRQ_EL0.
- In AArch32 state, MRC and MCR accesses to the following registers are trapped and reported using EC syndrome value 0x03, MRRC and MCRR accesses are trapped and reported using EC syndrome value 0x04:
  - CNTVCT and if CNTKCTL_EL1.EL0PCTEN is 0, CNTFRQ.

EL0VCTEN	Meaning
0b0	EL0 accesses to the frequency register and virtual counter registers are trapped.
0b1	This control does not cause any instructions to be trapped.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EL0PCTEN, bit [0]

Traps EL0 accesses to the frequency register and physical counter register to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, as follows:

- In AArch64 state, the following registers are trapped and reported using EC syndrome value `0x18`:
  - CNTPCT_EL0
  - CNTPCTSS_EL0
  - If CNTKCTL_EL1.EL0VCTEN is 0, CNTFRQ_EL0.
- In AArch32 state, MCR or MRC accesses the following registers are trapped and reported using EC syndrome value `0x03`, MCRR or MRRC accesses are trapped and reported using EC syndrome value `0x04`:
  - CNTPCT and if CNTKCTL_EL1.EL0VCTEN is 0, CNTFRQ.

EL0PCTEN	Meaning
`0b0`	EL0 accesses to the frequency register and physical counter register are trapped.
`0b1`	This control does not cause any instructions to be trapped.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, this control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTKCTL_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name `CNTKCTL_EL1` or `CNTKCTL_EL12` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTKCTL_EL1
```

op0	op1	CRn	CRm	op2
`0b11`	`0b000`	`0b1110`	`0b0001`	`0b000`

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    X[t, 64] = CNTKCTL_EL1;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = CNTHCTL_EL2_VHE(CNTHCTL_EL2);
    else
        X[t, 64] = CNTKCTL_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTKCTL_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

MSR CNTKCTL_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1110	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    CNTKCTL_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        CNTHCTL_EL2 = CNTHCTL_EL2_VHE(X[t, 64]);
    else
        CNTKCTL_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTKCTL_EL1 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, CNTKCTL_EL12

op0	op1	CRn	CRm	op2
0b11	0b101	0b1110	0b0001	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = CNTKCTL_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = CNTKCTL_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented MSR CNTKCTL_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b1110	0b0001	0b000

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        CNTKCTL_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        CNTKCTL_EL1 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.10.16 CNTP_CTL_EL0, Counter-timer Physical Timer Control Register

The CNTP_CTL_EL0 characteristics are:

Purpose

Control register for the EL1 physical timer.

Configuration

AArch64 System register CNTP_CTL_EL0 bits [31:0] are architecturally mapped to AArch32 System register CNTP_CTL[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to CNTP_CTL_EL0 are UNDEFINED.

Attributes

CNTP_CTL_EL0 is a 64-bit register.

Field descriptions

Bits [63:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

ISTATUS	Meaning
0b0	Timer condition is not met.
0b1	Timer condition is met.

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

IMASK	Meaning
0b0	Timer interrupt is not masked by the IMASK bit.
0b1	Timer interrupt is masked by the IMASK bit.

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

ENABLE	Meaning
0b0	Timer disabled.
0b1	Timer enabled.

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTP_TVAL_EL0 continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTP_CTL_EL0

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name `CNTP_CTL_EL0` or `CNTP_CTL_EL02` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTP_CTL_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b001

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHPS_CTL_EL2;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHP_CTL_EL2;
    else
        X[t, 64] = CNTP_CTL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x180];
    else
        X[t, 64] = CNTP_CTL_EL0;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHPS_CTL_EL2;
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHP_CTL_EL2;
    else
        X[t, 64] = CNTP_CTL_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTP_CTL_EL0;
```

MSR CNTP_CTL_EL0, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        CNTHPS_CTL_EL2 = X[t, 64];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CTL_EL2 = X[t, 64];
    else
        CNTP_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x180] = X[t, 64];
    else
        CNTP_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CTL_EL2 = X[t, 64];
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CTL_EL2 = X[t, 64];
    else
        CNTP_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTP_CTL_EL0 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, CNTP_CTL_EL02

op0	op1	CRn	CRm	op2
0b11	0b101	0b1110	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        if EL2Enabled() && !ELIsInHost(EL0) && CNTHCTL_EL2.EL1NVPCT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            X[t, 64] = NVMem[0x180];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = CNTP_CTL_EL0;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = CNTP_CTL_EL0;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented MSR CNTP_CTL_EL02, <Xt>

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b101	0b1110	0b0010	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        if EL2Enabled() && !ELIsInHost(EL0) && CNTHCTL_EL2.EL1NVPCT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            NVMem[0x180] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        CNTP_CTL_EL0 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        CNTP_CTL_EL0 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.10.17 CNTP_CVAL_EL0, Counter-timer Physical Timer CompareValue Register

The CNTP_CVAL_EL0 characteristics are:

Purpose

Holds the compare value for the EL1 physical timer.

Configuration

AArch64 System register CNTP_CVAL_EL0 bits [63:0] are architecturally mapped to AArch32 System register CNTP_CVAL[63:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to CNTP_CVAL_EL0 are UNDEFINED.

Attributes

CNTP_CVAL_EL0 is a 64-bit register.

Field descriptions

CompareValue, bits [63:0]

Holds the EL1 physical timer CompareValue.

When CNTP_CTL_EL0.ENABLE is 1, and TimerConditionMet is TRUE for the EL1 physical timer, the timer condition is met and all of the following are true:

- CNTP_CTL_EL0.ISTATUS is set to 1.
- If CNTP_CTL_EL0.IMASK is 0, an interrupt is generated.

TimerConditionMet is defined by 'Operation of the CompareValue views of the timers'.

The CompareValue view of the timer acts like a 64-bit upcounter timer.

When CNTP_CTL_EL0.ENABLE is 0, the timer condition is not met, but CNTPCT_EL0 continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTP_CVAL_EL0

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name `CNTP_CVAL_EL0` or `CNTP_CVAL_EL02` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, CNTP_CVAL_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHPS_CVAL_EL2;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHP_CVAL_EL2;
    else
        X[t, 64] = CNTP_CVAL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x178];
    else
        X[t, 64] = CNTP_CVAL_EL0;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHPS_CVAL_EL2;
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHP_CVAL_EL2;
    else
        X[t, 64] = CNTP_CVAL_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTP_CVAL_EL0;
```

```
MSR CNTP_CVAL_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            else
                AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
            CNTHPS_CVAL_EL2 = X[t, 64];
        elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
            CNTHP_CVAL_EL2 = X[t, 64];
        else
            CNTP_CVAL_EL0 = X[t, 64];
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EffectiveHCR_EL2_NVx() IN {'111'} then
            NVMem[0x178] = X[t, 64];
        else
            CNTP_CVAL_EL0 = X[t, 64];
    elsif PSTATE.EL == EL2 then
        if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
            CNTHPS_CVAL_EL2 = X[t, 64];
        elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
            CNTHP_CVAL_EL2 = X[t, 64];
        else
            CNTP_CVAL_EL0 = X[t, 64];
    elsif PSTATE.EL == EL3 then
        CNTP_CVAL_EL0 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, CNTP_CVAL_EL02

op0	op1	CRn	CRm	op2
0b11	0b101	0b1110	0b0010	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        if EL2Enabled() && !ELIsInHost(EL0) && CNTHCTL_EL2.EL1NVPCT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            X[t, 64] = NVMem[0x178];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = CNTP_CVAL_EL0;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = CNTP_CVAL_EL0;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    else
        UNDEFINED;
```

When FEAT_VHE is implemented MSR CNTP_CVAL_EL02, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b1110	0b0010	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        if EL2Enabled() && !ELIsInHost(EL0) && CNTHCTL_EL2.EL1NVPCT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            NVMem[0x178] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        CNTP_CVAL_EL0 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        CNTP_CVAL_EL0 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.10.18 CNTP_TVAL_EL0, Counter-timer Physical Timer TimerValue Register

The CNTP_TVAL_EL0 characteristics are:

Purpose

Holds the timer value for the EL1 physical timer.

Configuration

AArch64 System register CNTP_TVAL_EL0 bits [31:0] are architecturally mapped to AArch32 System register CNTP_TVAL[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to CNTP_TVAL_EL0 are UNDEFINED.

Attributes

CNTP_TVAL_EL0 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

TimerValue, bits [31:0]

The TimerValue view of the EL1 physical timer.

On a read of this register:

- If CNTP_CTL_EL0.ENABLE is 0, the value returned is UNKNOWN.
- If CNTP_CTL_EL0.ENABLE is 1, the value returned is (CNTP_CVAL_EL0 - CNTPCT_EL0).

On a write of this register, CNTP_CVAL_EL0 is set to (CNTPCT_EL0 + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTP_CTL_EL0.ENABLE is 1, the timer condition is met when (CNTPCT_EL0 - CNTP_CVAL_EL0) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTP_CTL_EL0.ISTATUS is set to 1.
- If CNTP_CTL_EL0.IMASK is 0, an interrupt is generated.

When CNTP_CTL_EL0.ENABLE is 0, the TimerValue cannot be read but continues to decrement. When the timer is enabled, the TimerValue represents the elapsed time whether that time was spent enabled or disabled.

Note

The value of CNTPCT_EL0 used in these calculations is the value seen at the Exception level that the CNTPCT_EL0 register is being read or written from.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing CNTP_TVAL_EL0

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name `CNTP_TVAL_EL0` or `CNTP_TVAL_EL02` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTP_TVAL_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        if CNTHPS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHPS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        if CNTHP_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHP_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.ECVEn == '1')
         ↪&& CNTHCTL_EL2.ECV == '1' && !ELIsInHost(EL0) then
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - (PhysicalCountInt() - CNTPOFF_EL2))<31:0>, 64);
    else
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.ECVEn == '1')
         ↪&& CNTHCTL_EL2.ECV == '1' then
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - (PhysicalCountInt() - CNTPOFF_EL2))<31:0>, 64);
    else
        if CNTP_CTL_EL0.ENABLE == '0' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        if CNTHPS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHPS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        if CNTHP_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHP_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    else
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL3 then
    if CNTP_CTL_EL0.ENABLE == '0' then
        X[t, 64] = bits(64) UNKNOWN;
    else
        X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
```

MSR CNTP_TVAL_EL0, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.ECVEn == '1')
         ↪&& CNTHCTL_EL2.ECV == '1' && !ELIsInHost(EL0) then
        CNTP_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTPOFF_EL2;
    else
        CNTP_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.ECVEn == '1')
         ↪&& CNTHCTL_EL2.ECV == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        CNTP_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTPOFF_EL2;
    else
        CNTP_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHP_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    else
        CNTP_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    CNTP_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
```

When FEAT_VHE is implemented MRS <Xt>, CNTP_TVAL_EL02

op0	op1	CRn	CRm	op2
0b11	0b101	0b1110	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        if CNTP_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTP_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
    else
        UNDEFINED;
```

When FEAT_VHE is implemented MSR CNTP_TVAL_EL02, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b1110	0b0010	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        CNTP_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        CNTP_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.10.19 CNTPCT_EL0, Counter-timer Physical Count Register

The CNTPCT_EL0 characteristics are:

Purpose

Reads of CNTPCT_EL0 return the 64-bit physical count value minus a physical offset.

Configuration

All reads to the CNTPCT_EL0 occur in program order relative to reads to CNTPCTSS_EL0 or CNTPCT_EL0.

AArch64 System register CNTPCT_EL0 bits [63:0] are architecturally mapped to AArch32 System register CNTPCT[63:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to CNTPCT_EL0 are UNDEFINED.

Attributes

CNTPCT_EL0 is a 64-bit register.

Field descriptions

PhysicalCount, bits [63:0]

Physical count value.

If the access is not trapped and all of the following are true, then reads of CNTPCT_EL0 from EL0 or EL1 return (PhysicalCountInt<63:0> - CNTPOFF_EL2<63:0>):

- EL2 is implemented and enabled in the current Security state.
- CNTHCTL_EL2.ECV is 1.
- SCR_EL3.ECVEn is 1.
- The Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.

Otherwise, reads of CNTPCT_EL0 return PhysicalCountInt<63:0>.

PhysicalCountInt is defined by 'The Physical Counter'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTPCT_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTPCT_EL0
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0000	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PCTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.ECVEn == '1')
            ↪&& CNTHCTL_EL2.ECV == '1' && !ELIsInHost(EL0) then
            X[t, 64] = PhysicalCountInt() - CNTPOFF_EL2;
        else
            X[t, 64] = PhysicalCountInt();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && CNTHCTL_EL2.EL1PCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.ECVEn == '1')
            ↪&& CNTHCTL_EL2.ECV == '1' then
            X[t, 64] = PhysicalCountInt() - CNTPOFF_EL2;
        else
            X[t, 64] = PhysicalCountInt();
elsif PSTATE.EL == EL2 then
    X[t, 64] = PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    X[t, 64] = PhysicalCountInt();
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.10.20 CNTPCTSS_EL0, Counter-timer Self-Synchronized Physical Count Register

The CNTPCTSS_EL0 characteristics are:

Purpose

Holds the self-synchronized view of the 64-bit physical count value.

Configuration

All reads to the CNTPCTSS_EL0 occur in program order relative to reads to CNTPCT_EL0 or CNTPCTSS_EL0.

This register is a view of the CNTPCT_EL0 register for which reads appear to occur in program order relative to other instructions, without the need for any explicit synchronization. Reads of this register return a value consistent with the counter not being read until the read instruction is known to be non-speculative.

AArch64 System register CNTPCTSS_EL0 bits [63:0] are architecturally mapped to AArch32 System register CNTPCTSS[63:0].

This register is present only when FEAT_ECV is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to CNTPCTSS_EL0 are UNDEFINED.

Attributes

CNTPCTSS_EL0 is a 64-bit register.

Field descriptions

Physical count value.

If the access is not trapped and all of the following are true, then reads of CNTPCTSS_EL0 from EL0 or EL1 return (PhysicalCountInt<63:0> - CNTPOFF_EL2<63:0>):

- EL2 is implemented and enabled in the current Security state.
- CNTHCTL_EL2.ECV is 1.
- SCR_EL3.ECVEn is 1.
- The Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.

Otherwise, reads of CNTPCTSS_EL0 return PhysicalCountInt<63:0>.

PhysicalCountInt is defined by 'The Physical Counter'.

SSPhysicalCount, bits [63:0]

Self-synchronized physical count value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTPCTSS_EL0

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, CNTPCTSS_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0000	0b101

```
if !(IsFeatureImplemented(FEAT_ECV) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0PCTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL2) && CNTHCTL_EL2.EL1PCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL2) && HCR_EL2.TGE == '0' && CNTHCTL_EL2.EL1PCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0PCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.ECVEn == '1')
            ↪&& CNTHCTL_EL2.ECV == '1' && !ELIsInHost(EL0) then
            X[t, 64] = PhysicalCountInt() - CNTPOFF_EL2;
        else
            X[t, 64] = PhysicalCountInt();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && CNTHCTL_EL2.EL1PCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_ECV_POFF) && EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.ECVEn == '1')
            ↪&& CNTHCTL_EL2.ECV == '1' then
            X[t, 64] = PhysicalCountInt() - CNTPOFF_EL2;
        else
            X[t, 64] = PhysicalCountInt();
elsif PSTATE.EL == EL2 then
    X[t, 64] = PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    X[t, 64] = PhysicalCountInt();
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.10.21 CNTPOFF_EL2, Counter-timer Physical Offset Register

The CNTPOFF_EL2 characteristics are:

Purpose

Holds the 64-bit physical offset. This is the offset for the AArch64 EL1 physical timer and counter when Enhanced Counter Virtualization is enabled.

Configuration

The physical offset applies to:

- Direct reads of CNTPCT_EL0 from EL0 or EL1.
- Indirect reads of the physical counter by the EL1 physical timer. See 'The Generic Timer in AArch64 state'.
- Indirect reads of the physical counter for timestamps generated by profiling logic. See 'The Statistical Profiling Extension' and 'AArch64 Self-hosted Trace'.

The physical offset only applies under conditions described by the relevant sections.

If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_ECV_POFF is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to CNTPOFF_EL2 are UNDEFINED.

Attributes

CNTPOFF_EL2 is a 64-bit register.

Field descriptions

PO, bits [63:0]

Physical offset.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTPOFF_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTPOFF_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0000	0b110

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !(IsFeatureImplemented(FEAT_ECV_POFF) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x1A8];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ECVEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.ECVEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = CNTPOFF_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTPOFF_EL2;
```

MSR CNTPOFF_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0000	0b110

```
if !(IsFeatureImplemented(FEAT_ECV_POFF) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x1A8] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ECVEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.ECVEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        CNTPOFF_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTPOFF_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.10.22 CNTPS_CTL_EL1, Counter-timer Physical Secure Timer Control Register

The CNTPS_CTL_EL1 characteristics are:

Purpose

Control register for the secure physical timer, usually accessible at EL3 but configurably accessible at EL1 in Secure state.

Configuration

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to CNTPS_CTL_EL1 are UNDEFINED.

Attributes

CNTPS_CTL_EL1 is a 64-bit register.

Field descriptions

Bits [63:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

ISTATUS	Meaning
0b0	Timer condition is not met.
0b1	Timer condition is met.

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is RO.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

IMASK	Meaning
0b0	Timer interrupt is not masked by the IMASK bit.
0b1	Timer interrupt is masked by the IMASK bit.

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

ENABLE	Meaning
0b0	Timer disabled.
0b1	Timer enabled.

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTPS_TVAL_EL1 continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTPS_CTL_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTPS_CTL_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b111	0b1110	0b0010	0b001

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ST == '0' then
            UNDEFINED;
        elsif SCR_EL3.EEL2 == '1' then
            UNDEFINED;
        elsif SCR_EL3.ST == '0' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = CNTPS_CTL_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTPS_CTL_EL1;
```

MSR CNTPS_CTL_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b111	0b1110	0b0010	0b001

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ST == '0' then
            UNDEFINED;
        elsif SCR_EL3.EEL2 == '1' then
            UNDEFINED;
        elsif SCR_EL3.ST == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            CNTPS_CTL_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    CNTPS_CTL_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.10.23 CNTPS_CVAL_EL1, Counter-timer Physical Secure Timer CompareValue Register

The CNTPS_CVAL_EL1 characteristics are:

Purpose

Holds the compare value for the secure physical timer, usually accessible at EL3 but configurably accessible at EL1 in Secure state.

Configuration

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to CNTPS_CVAL_EL1 are UNDEFINED.

Attributes

CNTPS_CVAL_EL1 is a 64-bit register.

Field descriptions

CompareValue, bits [63:0]

Holds the secure physical timer CompareValue.

When CNTPS_CTL_EL1.ENABLE is 1, and TimerConditionMet is TRUE for the EL1 secure physical timer, the timer condition is met and all of the following are true:

- CNTPS_CTL_EL1.ISTATUS is set to 1.
- If CNTPS_CTL_EL1.IMASK is 0, an interrupt is generated.

TimerConditionMet is defined by 'Operation of the CompareValue views of the timers'.

The CompareValue view of the timer acts like a 64-bit upcounter timer.

When CNTPS_CTL_EL1.ENABLE is 0, the timer condition is not met, but CNTPCT_EL0 continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTPS_CVAL_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTPS_CVAL_EL1
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b111	0b1110	0b0010	0b010

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ST == '0' then
            UNDEFINED;
        elsif SCR_EL3.EEL2 == '1' then
            UNDEFINED;
        elsif SCR_EL3.ST == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = CNTPS_CVAL_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTPS_CVAL_EL1;
```

```
MSR CNTPS_CVAL_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b111	0b1110	0b0010	0b010

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ST == '0' then
            UNDEFINED;
        elsif SCR_EL3.EEL2 == '1' then
            UNDEFINED;
        elsif SCR_EL3.ST == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            CNTPS_CVAL_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    CNTPS_CVAL_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.10.24 CNTPS_TVAL_EL1, Counter-timer Physical Secure Timer TimerValue Register

The CNTPS_TVAL_EL1 characteristics are:

Purpose

Holds the timer value for the secure physical timer, usually accessible at EL3 but configurably accessible at EL1 in Secure state.

Configuration

This register is present only when EL3 is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to CNTPS_TVAL_EL1 are UNDEFINED.

Attributes

CNTPS_TVAL_EL1 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

TimerValue, bits [31:0]

The TimerValue view of the secure physical timer.

On a read of this register:

- If CNTPS_CTL_EL1.ENABLE is 0, the value returned is UNKNOWN.
- If CNTPS_CTL_EL1.ENABLE is 1, the value returned is (CNTPS_CVAL_EL1 - CNTPCT_EL0).

On a write of this register, CNTPS_CVAL_EL1 is set to (CNTPCT_EL0 + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTPS_CTL_EL1.ENABLE is 1, the timer condition is met when (CNTPCT_EL0 - CNTPS_CVAL_EL1) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTPS_CTL_EL1.ISTATUS is set to 1.
- If CNTPS_CTL_EL1.IMASK is 0, an interrupt is generated.

When CNTPS_CTL_EL1.ENABLE is 0, the TimerValue cannot be read but continues to decrement. When the timer is enabled, the TimerValue represents the elapsed time whether that time was spent enabled or disabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTPS_TVAL_EL1

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, CNTPS_TVAL_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b111	0b1110	0b0010	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ST == '0' then
            UNDEFINED;
        elsif SCR_EL3.EEL2 == '1' then
            UNDEFINED;
        elsif SCR_EL3.ST == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            if CNTPS_CTL_EL1.ENABLE == '0' then
                X[t, 64] = bits(64) UNKNOWN;
            else
                X[t, 64] = ZeroExtend((CNTPS_CVAL_EL1 - PhysicalCountInt())<31:0>, 64);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if CNTPS_CTL_EL1.ENABLE == '0' then
        X[t, 64] = bits(64) UNKNOWN;
    else
        X[t, 64] = ZeroExtend((CNTPS_CVAL_EL1 - PhysicalCountInt())<31:0>, 64);
```

```
MSR CNTPS_TVAL_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b111	0b1110	0b0010	0b000

```
if !(HaveEL(EL3) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.ST == '0' then
            UNDEFINED;
        elsif SCR_EL3.EEL2 == '1' then
            UNDEFINED;
        elsif SCR_EL3.ST == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        else
            CNTPS_CVAL_EL1 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    CNTPS_CVAL_EL1 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.10.25 CNTV_CTL_EL0, Counter-timer Virtual Timer Control Register

The CNTV_CTL_EL0 characteristics are:

Purpose

Control register for the virtual timer.

Configuration

AArch64 System register CNTV_CTL_EL0 bits [31:0] are architecturally mapped to AArch32 System register CNTV_CTL[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to CNTV_CTL_EL0 are UNDEFINED.

Attributes

CNTV_CTL_EL0 is a 64-bit register.

Field descriptions

Bits [63:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

ISTATUS	Meaning
0b0	Timer condition is not met.
0b1	Timer condition is met.

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is RO.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

IMASK	Meaning
0b0	Timer interrupt is not masked by the IMASK bit.
0b1	Timer interrupt is masked by the IMASK bit.

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

ENABLE	Meaning
0b0	Timer disabled.
0b1	Timer enabled.

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTV_TVAL_EL0 continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTV_CTL_EL0

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name `CNTV_CTL_EL0` or `CNTV_CTL_EL02` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTV_CTL_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b001

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT ==
         ↪'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHVS_CTL_EL2;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHV_CTL_EL2;
    else
        X[t, 64] = CNTV_CTL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x170];
    else
        X[t, 64] = CNTV_CTL_EL0;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHVS_CTL_EL2;
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHV_CTL_EL2;
    else
        X[t, 64] = CNTV_CTL_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTV_CTL_EL0;
```

MSR CNTV_CTL_EL0, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT ==
         ↪'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CTL_EL2 = X[t, 64];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CTL_EL2 = X[t, 64];
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        CNTV_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x170] = X[t, 64];
    else
        CNTV_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CTL_EL2 = X[t, 64];
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CTL_EL2 = X[t, 64];
    else
        CNTV_CTL_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTV_CTL_EL0 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, CNTV_CTL_EL02

op0	op1	CRn	CRm	op2
0b11	0b101	0b1110	0b0011	0b001

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        if EL2Enabled() && !ELIsInHost(EL0) && CNTHCTL_EL2.EL1NVVCT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            X[t, 64] = NVMem[0x170];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = CNTV_CTL_EL0;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = CNTV_CTL_EL0;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented MSR CNTV_CTL_EL02, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b1110	0b0011	0b001

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        if EL2Enabled() && !ELIsInHost(EL0) && CNTHCTL_EL2.EL1NVVCT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            NVMem[0x170] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        CNTV_CTL_EL0 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        CNTV_CTL_EL0 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.10.26 CNTV_CVAL_EL0, Counter-timer Virtual Timer CompareValue Register

The CNTV_CVAL_EL0 characteristics are:

Purpose

Holds the compare value for the virtual timer.

Configuration

AArch64 System register CNTV_CVAL_EL0 bits [63:0] are architecturally mapped to AArch32 System register CNTV_CVAL[63:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to CNTV_CVAL_EL0 are UNDEFINED.

Attributes

CNTV_CVAL_EL0 is a 64-bit register.

Field descriptions

CompareValue, bits [63:0]

Holds the EL1 virtual timer CompareValue.

When CNTV_CTL_EL0.ENABLE is 1, and TimerConditionMet is TRUE for the EL1 virtual timer, the timer condition is met and all of the following are true:

- CNTV_CTL_EL0.ISTATUS is set to 1.
- If CNTV_CTL_EL0.IMASK is 0, an interrupt is generated.

TimerConditionMet is defined by 'Operation of the CompareValue views of the timers'.

The CompareValue view of the timer acts like a 64-bit upcounter timer.

When CNTV_CTL_EL0.ENABLE is 0, the timer condition is not met, but CNTVCT_EL0 continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTV_CVAL_EL0

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name `CNTV_CVAL_EL0` or `CNTV_CVAL_EL02` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, CNTV_CVAL_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT ==
     ↪'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHVS_CVAL_EL2;
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHV_CVAL_EL2;
    else
        X[t, 64] = CNTV_CVAL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x168];
    else
        X[t, 64] = CNTV_CVAL_EL0;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        X[t, 64] = CNTHVS_CVAL_EL2;
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        X[t, 64] = CNTHV_CVAL_EL2;
    else
        X[t, 64] = CNTV_CVAL_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTV_CVAL_EL0;
```

```
MSR CNTV_CVAL_EL0, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT ==
        ↪'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = X[t, 64];
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CVAL_EL2 = X[t, 64];
    else
        CNTV_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x168] = X[t, 64];
    else
        CNTV_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = X[t, 64];
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CVAL_EL2 = X[t, 64];
    else
        CNTV_CVAL_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTV_CVAL_EL0 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, CNTV_CVAL_EL02

op0	op1	CRn	CRm	op2
0b11	0b101	0b1110	0b0011	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        if EL2Enabled() && !ELIsInHost(EL0) && CNTHCTL_EL2.EL1NVVCT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            X[t, 64] = NVMem[0x168];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        X[t, 64] = CNTV_CVAL_EL0;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = CNTV_CVAL_EL0;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented MSR CNTV_CVAL_EL02, <Xt>

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b101	0b1110	0b0011	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        if EL2Enabled() && !ELIsInHost(EL0) && CNTHCTL_EL2.EL1NVVCT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            NVMem[0x168] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        CNTV_CVAL_EL0 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        CNTV_CVAL_EL0 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.10.27 CNTV_TVAL_EL0, Counter-timer Virtual Timer TimerValue Register

The CNTV_TVAL_EL0 characteristics are:

Purpose

Holds the timer value for the EL1 virtual timer.

Configuration

AArch64 System register CNTV_TVAL_EL0 bits [31:0] are architecturally mapped to AArch32 System register CNTV_TVAL[31:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to CNTV_TVAL_EL0 are UNDEFINED.

Attributes

CNTV_TVAL_EL0 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

TimerValue, bits [31:0]

The TimerValue view of the EL1 virtual timer.

On a read of this register:

- If CNTV_CTL_EL0.ENABLE is 0, the value returned is UNKNOWN.
- If CNTV_CTL_EL0.ENABLE is 1, the value returned is (CNTV_CVAL_EL0 - CNTVCT_EL0).

On a write of this register, CNTV_CVAL_EL0 is set to (CNTVCT_EL0 + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTV_CTL_EL0.ENABLE is 1, the timer condition is met when (CNTVCT_EL0 - CNTV_CVAL_EL0) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTV_CTL_EL0.ISTATUS is set to 1.
- If CNTV_CTL_EL0.IMASK is 0, an interrupt is generated.

When CNTV_CTL_EL0.ENABLE is 0, the TimerValue cannot be read but continues to decrement. When the timer is enabled, the TimerValue represents the elapsed time whether that time was spent enabled or disabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing CNTV_TVAL_EL0

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name `CNTV_TVAL_EL0` or `CNTV_TVAL_EL02` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTV_TVAL_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT ==
     ↪'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        if CNTHVS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHVS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        if CNTHV_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHV_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif HaveEL(EL2) && !ELIsInHost(EL0) then
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>, 64);
    else
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL2) then
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>, 64);
    else
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        if CNTHVS_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHVS_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        if CNTHV_CTL_EL2.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTHV_CVAL_EL2 - PhysicalCountInt())<31:0>, 64);
    elsif !ELIsInHost(EL2) then
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>, 64);
    else
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
elsif PSTATE.EL == EL3 then
    if CNTV_CTL_EL0.ENABLE == '0' then
        X[t, 64] = bits(64) UNKNOWN;
    elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
        X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>, 64);
    elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) then
        X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF))<31:0>, 64);
    else
        X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - PhysicalCountInt())<31:0>, 64);
```

MSR CNTV_TVAL_EL0, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT ==
     ↪'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif ELIsInHost(EL0) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif ELIsInHost(EL0) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif HaveEL(EL2) && !ELIsInHost(EL0) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF_EL2;
    else
        CNTV_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsFeatureImplemented(FEAT_ECV) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL2) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF_EL2;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    else
        CNTV_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) && IsCurrentSecurityState(SS_Secure) && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif ELIsInHost(EL2) && !IsCurrentSecurityState(SS_Secure) then
        CNTHV_CVAL_EL2 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
    elsif !ELIsInHost(EL2) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF_EL2;
    else
        CNTV_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    if HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF_EL2;
    elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF;
    else
        CNTV_CVAL_EL0 = SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt();
```

When FEAT_VHE is implemented MRS <Xt>, CNTV_TVAL_EL02

op0	op1	CRn	CRm	op2
0b11	0b101	0b1110	0b0011	0b000

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>, 64);
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        if CNTV_CTL_EL0.ENABLE == '0' then
            X[t, 64] = bits(64) UNKNOWN;
        else
            X[t, 64] = ZeroExtend((CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>, 64);
    else
        UNDEFINED;
```

When FEAT_VHE is implemented MSR CNTV_TVAL_EL02, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b1110	0b0011	0b000

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF_EL2;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        CNTV_CVAL_EL0 = (SignExtend(X[t, 64]<31:0>, 64) + PhysicalCountInt()) - CNTVOFF_EL2;
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.10.28 CNTVCT_EL0, Counter-timer Virtual Count Register

The CNTVCT_EL0 characteristics are:

Purpose

Reads of CNTVCT_EL0 return the 64-bit physical count value minus a virtual offset.

Configuration

All reads to the CNTVCT_EL0 occur in program order relative to reads to CNTVCTSS_EL0 or CNTVCT_EL0.

AArch64 System register CNTVCT_EL0 bits [63:0] are architecturally mapped to AArch32 System register CNTVCT[63:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to CNTVCT_EL0 are UNDEFINED.

Attributes

CNTVCT_EL0 is a 64-bit register.

Field descriptions

VirtualCount, bits [63:0]

Virtual count value.

When EL2 is implemented, if the access is not trapped, and any of the following are true, then reads of CNTVCT_EL0 from EL0 and EL1 return (PhysicalCountInt<63:0> - CNTVOFF_EL2<63:0>):

- All of the following are true:
  - The Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.
  - CNTVCT_EL0 is read from EL0.
- All of the following are true:
  - The Effective value of HCR_EL2.E2H is 0.
  - CNTVCT_EL0 is read from EL2.
- CNTVCT_EL0 is read from EL1 or EL3.

Otherwise reads of CNTVCT_EL0 return PhysicalCountInt<63:0>.

PhysicalCountInt is defined by 'The Physical Counter'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTVCT_EL0

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, CNTVCT_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0000	0b010

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VCTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && CNTHCTL_EL2.EL1TVCT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if HaveEL(EL2) && (!EL2Enabled() || !ELIsInHost(EL0)) then
            X[t, 64] = PhysicalCountInt() - CNTVOFF_EL2;
        elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) && !ELUsingAArch32(EL2) &&
             ↪(!EL2Enabled() || !ELIsInHost(EL0)) then
            X[t, 64] = PhysicalCountInt() - CNTVOFF;
        else
            X[t, 64] = PhysicalCountInt();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && CNTHCTL_EL2.EL1TVCT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if HaveEL(EL2) then
            X[t, 64] = PhysicalCountInt() - CNTVOFF_EL2;
        else
            X[t, 64] = PhysicalCountInt();
elsif PSTATE.EL == EL2 then
    if HaveEL(EL2) && !ELIsInHost(EL2) then
        X[t, 64] = PhysicalCountInt() - CNTVOFF_EL2;
    else
        X[t, 64] = PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    if HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
        X[t, 64] = PhysicalCountInt() - CNTVOFF_EL2;
    elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) then
        X[t, 64] = PhysicalCountInt() - CNTVOFF;
    else
        X[t, 64] = PhysicalCountInt();
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.10.29 CNTVCTSS_EL0, Counter-timer Self-Synchronized Virtual Count Register

The CNTVCTSS_EL0 characteristics are:

Purpose

Reads of CNTVCTSS_EL0 return the 64-bit physical count value minus a virtual offset.

Configuration

All reads to the CNTVCTSS_EL0 occur in program order relative to reads to CNTVCT_EL0 or CNTVCTSS_EL0.

This register is a view of the CNTVCT_EL0 register for which reads appear to occur in program order relative to other instructions, without the need for any explicit synchronization. Reads of this register return a value consistent with the counter not being read until the read instruction is known to be non-speculative.

AArch64 System register CNTVCTSS_EL0 bits [63:0] are architecturally mapped to AArch32 System register CNTVCTSS[63:0].

This register is present only when FEAT_ECV is implemented and FEAT_AA64 is implemented. Otherwise, direct accesses to CNTVCTSS_EL0 are UNDEFINED.

Attributes

CNTVCTSS_EL0 is a 64-bit register.

Field descriptions

SSVirtualCount, bits [63:0]

Self-synchronized virtual count value.

When EL2 is implemented, if the access is not trapped, and any of the following are true, then reads of CNTVCTSS_EL0 from EL0 and EL1 return (PhysicalCountInt<63:0> - CNTVOFF_EL2<63:0>):

- All of the following are true:
  - The Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.
  - CNTVCTSS_EL0 is read from EL0.
- All of the following are true:
  - The Effective value of HCR_EL2.E2H is 0.
  - CNTVCTSS_EL0 is read from EL2.
- CNTVCTSS_EL0 is read from EL1 or EL3.

Otherwise reads of CNTVCTSS_EL0 return PhysicalCountInt<63:0>.

PhysicalCountInt is defined by 'The Physical Counter'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTVCTSS_EL0

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, CNTVCTSS_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b1110	0b0000	0b110

```
if !(IsFeatureImplemented(FEAT_ECV) && IsFeatureImplemented(FEAT_AA64)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if !ELIsInHost(EL0) && CNTKCTL_EL1.EL0VCTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ELIsInHost(EL0) && CNTHCTL_EL2.EL0VCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELIsInHost(EL0) && CNTHCTL_EL2.EL1TVCT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if HaveEL(EL2) && (!EL2Enabled() || !ELIsInHost(EL0)) then
            X[t, 64] = PhysicalCountInt() - CNTVOFF_EL2;
        elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) && !ELUsingAArch32(EL2) &&
             ↪(!EL2Enabled() || !ELIsInHost(EL0)) then
            X[t, 64] = PhysicalCountInt() - CNTVOFF;
        else
            X[t, 64] = PhysicalCountInt();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && CNTHCTL_EL2.EL1TVCT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if HaveEL(EL2) then
            X[t, 64] = PhysicalCountInt() - CNTVOFF_EL2;
        else
            X[t, 64] = PhysicalCountInt();
elsif PSTATE.EL == EL2 then
    if HaveEL(EL2) && !ELIsInHost(EL2) then
        X[t, 64] = PhysicalCountInt() - CNTVOFF_EL2;
    else
        X[t, 64] = PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    if HaveEL(EL2) && IsFeatureImplemented(FEAT_AA64EL2) && !ELUsingAArch32(EL2) then
        X[t, 64] = PhysicalCountInt() - CNTVOFF_EL2;
    elsif HaveEL(EL2) && IsFeatureImplemented(FEAT_AA32EL2) then
        X[t, 64] = PhysicalCountInt() - CNTVOFF;
    else
        X[t, 64] = PhysicalCountInt();
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.10.30 CNTVOFF_EL2, Counter-timer Virtual Offset Register

The CNTVOFF_EL2 characteristics are:

Purpose

Holds the 64-bit virtual offset. This is the offset for the AArch64 virtual timers and counters.

Configuration

The virtual offset applies to:

- Direct reads of CNTVCT_EL0.
- Indirect reads of the virtual counter by the EL1 virtual timer. See 'The Generic Timer in AArch64 state'.
- Indirect reads of the virtual counter for timestamps generated by profiling logic. See 'The Statistical Profiling Extension' and 'AArch64 Self-hosted Trace'.

The virtual offset only applies under conditions described by the relevant sections.

AArch64 System register CNTVOFF_EL2 bits [63:0] are architecturally mapped to AArch32 System register CNTVOFF[63:0].

This register is present only when FEAT_AA64 is implemented. Otherwise, direct accesses to CNTVOFF_EL2 are UNDEFINED.

Attributes

CNTVOFF_EL2 is a 64-bit register.

Field descriptions

VOffset, bits [63:0]

Virtual offset.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTVOFF_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, CNTVOFF_EL2
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0000	0b011

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x060];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    X[t, 64] = CNTVOFF_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = CNTVOFF_EL2;
```

MSR CNTVOFF_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1110	0b0000	0b011

```
if !IsFeatureImplemented(FEAT_AA64) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x060] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTVOFF_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    CNTVOFF_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.11 Guarded Control Stack registers

This section lists the Guarded Control Stack registers in AArch64.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.11.1 GCSCR_EL1, Guarded Control Stack Control Register (EL1)

The GCSCR_EL1 characteristics are:

Purpose

Controls the Guarded Control Stack at EL1.

Configuration

This register is present only when FEAT_GCS is implemented. Otherwise, direct accesses to GCSCR_EL1 are UNDEFINED.

Attributes

GCSCR_EL1 is a 64-bit register.

Field descriptions

Bits [63:10]

Reserved, RES0.

STREn, bit [9]

Execution of the following instructions are trapped:

- GCSSTR.
- GCSSTTR if any of the following are true.
  - PSTATE.UAO is 1.
  - If EL2 is implemented and enabled in the current Security state and the Effective value of HCR_EL2.{NV,NV1} is {1,1}.

STREn	Meaning
0b0	Execution of any of the specified instructions at EL1 causes a GCS exception.
0b1	This control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

PUSHMEn, bit [8]

Trap GCSPUSHM instruction.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

PUSHMEn	Meaning
0b0	Execution of a GCSPUSHM instruction at EL1 causes a Trap exception.
0b1	This control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Bit [7]

Reserved, RES0.

EXLOCKEN, bit [6]

Exception state lock.

Prevents MSR instructions from writing to ELR_EL1 or SPSR_EL1.

EXLOCKEN	Meaning
0b0	EL1 exception state locking disabled.
0b1	EL1 exception state locking enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

RVCHKEN, bit [5]

Return value check enable.

RVCHKEN	Meaning
0b0	Return value checking disabled at EL1.
0b1	Return value checking enabled at EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [4:1]

Reserved, RES0.

PCRSEL, bit [0]

Guarded Control Stack procedure call return enable selection.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

PCRSEL	Meaning
0b0	Guarded Control Stack at EL1 is not PCR Selected.
0b1	Guarded Control Stack at EL1 is PCR Selected.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing GCSCR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name GCSCR_EL1 or GCSCR_EL12 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, GCSCR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.nGCS_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x8D0];
    else
        X[t, 64] = GCSCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = GCSCR_EL2;
    else
        X[t, 64] = GCSCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = GCSCR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MSR GCSCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.nGCS_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x8D0] = X[t, 64];
    else
        GCSCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        GCSCR_EL2 = X[t, 64];
    else
        GCSCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    GCSCR_EL1 = X[t, 64];
```

```
When FEAT_VHE is implemented MRS <Xt>, GCSCR_EL12
```

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x8D0];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = GCSCR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = GCSCR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented MSR GCSCR_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x8D0] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            GCSCR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        GCSCR_EL1 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.11.2 GCSCR_EL2, Guarded Control Stack Control Register (EL2)

The GCSCR_EL2 characteristics are:

Purpose

Controls the Guarded Control Stack at EL2.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_GCS is implemented. Otherwise, direct accesses to GCSCR_EL2 are UNDEFINED.

Attributes

GCSCR_EL2 is a 64-bit register.

Field descriptions

Bits [63:10]

Reserved, RES0.

STREn, bit [9]

Execution of the following instructions are trapped:

- GCSSTR.
- GCSSTTR if any of the following are true.
  - The Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.
  - The Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, and PSTATE.UAO is 1.

STREn	Meaning
0b0	Execution of any of the specified instructions at EL2 cause a GCS exception.
0b1	This control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

PUSHMEn, bit [8]

Trap GCSPUSHM instruction.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

PUSHMEn	Meaning
0b0	Execution of a GCSPUSHM instruction at EL2 causes a Trap exception.
0b1	This control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Bit [7]

Reserved, RES0.

EXLOCKEN, bit [6]

Exception state lock.

Prevents MSR instructions from writing to ELR_EL2 or SPSR_EL2.

EXLOCKEN	Meaning
0b0	EL2 exception state locking disabled.
0b1	EL2 exception state locking enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

RVCHKEN, bit [5]

Return value check enable.

RVCHKEN	Meaning
0b0	Return value checking disabled at EL2.
0b1	Return value checking enabled at EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [4:1]

Reserved, RES0.

PCRSEL, bit [0]

Guarded Control Stack procedure call return enable selection.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

PCRSEL	Meaning
0b0	Guarded Control Stack at EL2 is not PCR Selected.
0b1	Guarded Control Stack at EL2 is PCR Selected.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing GCSCR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name GCSCR_EL2 or GCSCR_EL1 are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, GCSCR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = GCSCR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = GCSCR_EL2;
```

```
MSR GCSCR_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0101	0b000

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        GCSCR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    GCSCR_EL2 = X[t, 64];
```

MRS <Xt>, GCSCR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.nGCS_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x8D0];
    else
        X[t, 64] = GCSCR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = GCSCR_EL2;
    else
        X[t, 64] = GCSCR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = GCSCR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MSR GCSCR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.nGCS_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x8D0] = X[t, 64];
    else
        GCSCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        GCSCR_EL2 = X[t, 64];
    else
        GCSCR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    GCSCR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.11.3 GCSCR_EL3, Guarded Control Stack Control Register (EL3)

The GCSCR_EL3 characteristics are:

Purpose

Controls the Guarded Control Stack at EL3.

Configuration

This register is present only when FEAT_GCS is implemented and EL3 is implemented. Otherwise, direct accesses to GCSCR_EL3 are UNDEFINED.

Attributes

GCSCR_EL3 is a 64-bit register.

Field descriptions

Bits [63:10]

Reserved, RES0.

STREn, bit [9]

Execution of the following instructions are trapped:

- GCSSTR.
- GCSSTTR.

STREn	Meaning
0b0	Execution of any of the specified instructions at EL3 cause a GCS exception.
0b1	This control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

PUSHMEn, bit [8]

Trap GCSPUSHM instruction.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

PUSHMEn	Meaning
0b0	Execution of a GCSPUSHM instruction at EL3 causes a Trap exception.
0b1	This control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Bit [7]

Reserved, RES0.

EXLOCKEN, bit [6]

Exception state lock.

Prevents MSR instructions from writing to ELR_EL3 or SPSR_EL3.

EXLOCKEN	Meaning
0b0	EL3 exception state locking disabled.
0b1	EL3 exception state locking enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

RVCHKEN, bit [5]

Return value check enable.

RVCHKEN	Meaning
0b0	Return value checking disabled at EL3.
0b1	Return value checking enabled at EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [4:1]

Reserved, RES0.

PCRSEL, bit [0]

Guarded Control Stack procedure call return enable selection.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

PCRSEL	Meaning
0b0	Guarded Control Stack at EL3 is not PCR Selected.
0b1	Guarded Control Stack at EL3 is PCR Selected.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing GCSCR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, GCSCR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0010	0b0101	0b000

```
if !(IsFeatureImplemented(FEAT_GCS) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = GCSCR_EL3;
```

```
MSR GCSCR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0010	0b0101	0b000

```
if !(IsFeatureImplemented(FEAT_GCS) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.GCSCR_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        GCSCR_EL3 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.11.4 GCSCRE0_EL1, Guarded Control Stack Control Register (EL0)

The GCSCRE0_EL1 characteristics are:

Purpose

Controls the Guarded Control Stack at EL0.

Configuration

This register is present only when FEAT_GCS is implemented. Otherwise, direct accesses to GCSCRE0_EL1 are UNDEFINED.

Attributes

GCSCRE0_EL1 is a 64-bit register.

Field descriptions

Bits [63:11]

Reserved, RES0.

nTR, bit [10]

Trap GCS register accesses from EL0.

nTR	Meaning
0b0	Read accesses to GCSPR_EL0 at EL0 cause a Trap exception.
0b1	This control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

STREn, bit [9]

Execution of the following instructions are trapped:

- GCSSTR.
- GCSSTTR.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

STREn	Meaning
0b0	Execution of any of the specified instructions at EL0 cause a GCS exception.
0b1	This control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

PUSHMEn, bit [8]

Trap GCSPUSHM instruction.

PUSHMEn	Meaning
0b0	Execution of a GCSPUSHM instruction at EL0 causes a Trap exception.
0b1	This control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Bits [7:6]

Reserved, RES0.

RVCHKEN, bit [5]

Return value check enable.

RVCHKEN	Meaning
0b0	Return value checking disabled at EL0.
0b1	Return value checking enabled at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [4:1]

Reserved, RES0.

PCRSEL, bit [0]

Guarded Control Stack procedure call return enable selection.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

PCRSEL	Meaning
0b0	Guarded Control Stack at EL0 is not PCR Selected.
0b1	Guarded Control Stack at EL0 is PCR Selected.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Accessing GCSCRE0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, GCSCRE0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0101	0b010

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.nGCS_EL0 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = GCSCRE0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = GCSCRE0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = GCSCRE0_EL1;
```

```
MSR GCSCRE0_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0101	0b010

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.nGCS_EL0 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        GCSCRE0_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        GCSCRE0_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    GCSCRE0_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.11.5 GCSPR_EL0, Guarded Control Stack Pointer Register (EL0)

The GCSPR_EL0 characteristics are:

Purpose

Contains the Guarded Control Stack Pointer at EL0.

Configuration

This register is present only when FEAT_GCS is implemented. Otherwise, direct accesses to GCSPR_EL0 are UNDEFINED.

Attributes

GCSPR_EL0 is a 64-bit register.

Field descriptions

PTR[63:3], bits [63:3]

EL0 Guarded Control Stack Pointer bits [63:3].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [2:0]

Reserved, RES0.

Accessing GCSPR_EL0

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, GCSPR_EL0
```

op0	op1	CRn	CRm	op2
0b11	0b011	0b0010	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif (!EL2Enabled() || HCR_EL2.TGE != '1') && GCSCRE0_EL1.nTR == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && HCR_EL2.TGE == '1' && GCSCRE0_EL1.nTR == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif EL2Enabled() && !ELIsInHost(EL0) && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
         ↪SCR_EL3.FGTEn == '1') && HFGRTR_EL2.nGCS_EL0 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = GCSPR_EL0;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.nGCS_EL0 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = GCSPR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = GCSPR_EL0;
elsif PSTATE.EL == EL3 then
    X[t, 64] = GCSPR_EL0;
```

MSR GCSPR_EL0, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b011	0b0010	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.nGCS_EL0 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        GCSPR_EL0 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        GCSPR_EL0 = X[t, 64];
elsif PSTATE.EL == EL3 then
    GCSPR_EL0 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.11.6 GCSPR_EL1, Guarded Control Stack Pointer Register (EL1)

The GCSPR_EL1 characteristics are:

Purpose

Contains the Guarded Control Stack Pointer at EL1.

Configuration

This register is present only when FEAT_GCS is implemented. Otherwise, direct accesses to GCSPR_EL1 are UNDEFINED.

Attributes

GCSPR_EL1 is a 64-bit register.

Field descriptions

PTR[63:3], bits [63:3]

EL1 Guarded Control Stack Pointer bits [63:3].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [2:0]

Reserved, RES0.

Accessing GCSPR_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name `GCSPR_EL1` or `GCSPR_EL12` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, GCSPR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.nGCS_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x8C0];
    else
        X[t, 64] = GCSPR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = GCSPR_EL2;
    else
        X[t, 64] = GCSPR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = GCSPR_EL1;
```

```
MSR GCSPR_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.nGCS_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x8C0] = X[t, 64];
    else
        GCSPR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif ELIsInHost(EL2) then
        GCSPR_EL2 = X[t, 64];
    else
        GCSPR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    GCSPR_EL1 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, GCSPR_EL12

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x8C0];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = GCSPR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = GCSPR_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented MSR GCSPR_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b0010	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x8C0] = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            GCSPR_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        GCSPR_EL1 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.11.7 GCSPR_EL2, Guarded Control Stack Pointer Register (EL2)

The GCSPR_EL2 characteristics are:

Purpose

Contains the Guarded Control Stack Pointer at EL2.

Configuration

If EL2 is not implemented, this register is RES0 from EL3.

This register is present only when FEAT_GCS is implemented. Otherwise, direct accesses to GCSPR_EL2 are UNDEFINED.

Attributes

GCSPR_EL2 is a 64-bit register.

Field descriptions

PTR[63:3], bits [63:3]

EL2 Guarded Control Stack Pointer bits [63:3].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [2:0]

Reserved, RES0.

Accessing GCSPR_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name `GCSPR_EL2` or `GCSPR_EL1` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, GCSPR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0101	0b001

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = GCSPR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = GCSPR_EL2;
```

MSR GCSPR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b0010	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        GCSPR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    GCSPR_EL2 = X[t, 64];
```

MRS <Xt>, GCSPR_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0101	0b001

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGRTR_EL2.nGCS_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x8C0];
    else
        X[t, 64] = GCSPR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = GCSPR_EL2;
    else
        X[t, 64] = GCSPR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = GCSPR_EL1;
```

MSR GCSPR_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_GCS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
         ↪HFGWTR_EL2.nGCS_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x8C0] = X[t, 64];
    else
        GCSPR_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    if HaveEL(EL3) && EL3SDDUndefPriority() && SCR_EL3.GCSEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.GCSEn == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        GCSPR_EL2 = X[t, 64];
    else
        GCSPR_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    GCSPR_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.11.8 GCSPR_EL3, Guarded Control Stack Pointer Register (EL3)

The GCSPR_EL3 characteristics are:

Purpose

Contains the Guarded Control Stack Pointer at EL3.

Configuration

This register is present only when FEAT_GCS is implemented and EL3 is implemented. Otherwise, direct accesses to GCSPR_EL3 are UNDEFINED.

Attributes

GCSPR_EL3 is a 64-bit register.

Field descriptions

PTR[63:3], bits [63:3]

EL3 Guarded Control Stack Pointer bits [63:3].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [2:0]

Reserved, RES0.

Accessing GCSPR_EL3

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, GCSPR_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0010	0b0101	0b001

```
if !(IsFeatureImplemented(FEAT_GCS) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = GCSPR_EL3;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MSR GCSPR_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b0010	0b0101	0b001

```
if !(IsFeatureImplemented(FEAT_GCS) && HaveEL(EL3)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.GCSPR_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        GCSPR_EL3 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.12 MPAM registers

This section lists the MPAM registers in AArch64.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.12.1 MPAM0_EL1, MPAM0 Register (EL1)

The MPAM0_EL1 characteristics are:

Purpose

Holds information to generate MPAM labels for memory requests when executing at EL0.

Configuration

If FEAT_MPAM is implemented, EL2 is implemented and enabled in the current Security state, and the MPAM virtualization option is present, then:

- If MPAMHCR_EL2.GSTAPP_PLK == 1 and HCR_EL2.TGE == 0, MPAM1_EL1 is used instead of MPAM0_EL1 to generate MPAM information to label memory requests.
- If the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1} and MPAMHCR_EL2.EL0_VPMEN == 1, then MPAM PARTIDs in MPAM0_EL1 are virtual and mapped into physical PARTIDs for the current Security state.

This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAM0_EL1 are UNDEFINED.

Attributes

MPAM0_EL1 is a 64-bit register.

Field descriptions

Bits [63:48]

Reserved, RES0.

PMG_D, bits [47:40]

Performance monitoring group property for PARTID_D.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PMG_I, bits [39:32]

Performance monitoring group property for PARTID_I.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PARTID_D, bits [31:16]

Partition ID for data accesses, including load and store accesses, made from EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

PARTID_I, bits [15:0]

Partition ID for instruction accesses made from EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAM0_EL1

None of the fields in this register are permitted to be cached in a TLB.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAM0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b001

```
if !IsFeatureImplemented(FEAT_MPAM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
        ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_MPAM) && MPAM2_EL2.TRAPMPAM0EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = MPAM0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
        ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAM0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAM0_EL1;
```

```
MSR MPAM0_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b001

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_MPAM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
        ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_MPAM) && MPAM2_EL2.TRAPMPAM0EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        MPAM0_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
        ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAM0_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAM0_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.12.2 MPAM1_EL1, MPAM1 Register (EL1)

The MPAM1_EL1 characteristics are:

Purpose

Holds information to generate MPAM labels for memory requests when executing at EL1.

Configuration

If FEAT_MPAMv1p0 or FEAT_MPAMv0p1 is implemented, EL2 is implemented and enabled in the current Security state, and the MPAM virtualization option is present, then:

- If MPAMHCR_EL2.GSTAPP_PLK == 1 and HCR_EL2.TGE == 0, MPAM1_EL1 is used instead of MPAM0_EL1 to generate MPAM labels for memory requests when executing at EL0.
- If MPAMHCR_EL2.EL1_VPMEN == 1, MPAM PARTIDs in MPAM1_EL1 are virtual and mapped into physical PARTIDs for the current Security state. This mapping of MPAM1_EL1 virtual PARTIDs to physical PARTIDs when EL1_VPMEN is 1 also applies when MPAM1_EL1 is used at EL0 due to MPAMHCR_EL2.GSTAPP_PLK.

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p0 is implemented) and EL3 is implemented, AArch64 System register MPAM1_EL1 bit [63] is architecturally mapped to AArch64 System register MPAM3_EL3[63].

When ((IsFeatureImplemented(FEAT_MPAMv0p1) || IsFeatureImplemented(FEAT_MPAMv1p0)) && (!HaveEL(EL3))) && HaveEL(EL2), AArch64 System register MPAM1_EL1 bit [63] is architecturally mapped to AArch64 System register MPAM2_EL2[63].

This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAM1_EL1 are UNDEFINED.

Attributes

MPAM1_EL1 is a 64-bit register.

Field descriptions

MPAMEN, bit [63]

MPAM Enable. MPAM is enabled when MPAMEN == 1. When disabled, all PARTIDs and PMGs are output as their default value in the corresponding ID space.

MPAMEN	Meaning
0b0	The default PARTID and default PMG are output in MPAM information.
0b1	MPAM information is output based on the MPAMn_ELx register for ELn according the MPAM configuration.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to `'0'`.

Accessing this field has the following behavior:

- Access to this field is RAZ/WI if all of the following are true:
  - EL3 is implemented
  - MPAM3_EL3.MPAMEN == ‘0’
- Access to this field is RAO/WI if all of the following are true:
  - EL3 is implemented
  - MPAM3_EL3.MPAMEN == ‘1’
- Access to this field is RAZ/WI if all of the following are true:
  - EL3 is not implemented
  - EL2 is implemented
  - MPAM2_EL2.MPAMEN == ‘0’
- Access to this field is RAO/WI if all of the following are true:
  - EL3 is not implemented
  - EL2 is implemented
  - MPAM2_EL2.MPAMEN == ‘1’

Bits [62:61]

Reserved, RES0.

FORCED_NS, bit [60]

When FEAT_MPAMv0p1 is implemented:

In the Secure state, FORCED_NS indicates the state of MPAM3_EL3.FORCE_NS.

FORCED_NS	Meaning
`0b0`	In the Non-secure state, always reads as 0. In the Secure state, indicates that MPAM3_EL3.FORCE_NS == 0.
`0b1`	In the Secure state, indicates that MPAM3_EL3.FORCE_NS == 1.

Always reads as 0 in the Non-secure state.

Writes are ignored.

Access to this field is RO.

Otherwise:

Reserved, RES0.

Bits [59:55]

Reserved, RES0.

ALTSP_FRCD, bit [54]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == '1':

Alternative PARTID forced for PARTIDs in this register.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ALTSP_FRCD	Meaning
0b0	The PARTIDs in MPAM1_EL1 and MPAM0_EL1 are using the primary PARTID space.
0b1	The PARTIDs in MPAM1_EL1 and MPAM0_EL1 are using the alternative PARTID space.

This bit indicates that a higher Exception level has forced the PARTIDs in this register to use the alternative PARTID space defined for the current Security state.

In MPAM1_EL1, it also indicates that MPAM0_EL1 is forced to use alternative PARTID space.

For more information, see In a PE with FEAT_RME, selection of primary or alternative PARTID space.

Accessing this field has the following behavior:

- When !UsePrimarySpaceEL10(), access to this field is RAO/WI.
- Otherwise, access to this field is RAZ/WI.

Otherwise:

Reserved, RES0.

Bits [53:48]

Reserved, RES0.

PMG_D, bits [47:40]

Performance monitoring group property for PARTID_D.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PMG_I, bits [39:32]

Performance monitoring group property for PARTID_I.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PARTID_D, bits [31:16]

Partition ID for data accesses, including load and store accesses, made from EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PARTID_I, bits [15:0]

Partition ID for instruction accesses made from EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAM1_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name `MPAM1_EL1` or `MPAM1_EL12` are not guaranteed to be ordered with respect to accesses using the other accessor name.

None of the fields in this register are permitted to be cached in a TLB.

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, MPAM1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_MPAM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
        ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_MPAM) && MPAM2_EL2.TRAPMPAM1EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x900];
    else
        X[t, 64] = MPAM1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
        ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = MPAM2_EL2;
    else
        X[t, 64] = MPAM1_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAM1_EL1;
```

```
MSR MPAM1_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_MPAM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
        ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_MPAM) && MPAM2_EL2.TRAPMPAM1EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x900] = X[t, 64];
    else
        MPAM1_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
     ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        MPAM2_EL2 = X[t, 64];
    else
        MPAM1_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAM1_EL1 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, MPAM1_EL12

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_MPAM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x900];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) &&
         ↪MPAM3_EL3.TRAPLOWER == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = MPAM1_EL1;
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        X[t, 64] = MPAM1_EL1;
    else
        UNDEFINED;
```

When FEAT_VHE is implemented MSR MPAM1_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_MPAM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x900] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) &&
            ↪MPAM3_EL3.TRAPLOWER == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            MPAM1_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        MPAM1_EL1 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.12.3 MPAM2_EL2, MPAM2 Register (EL2)

The MPAM2_EL2 characteristics are:

Purpose

Holds information to generate MPAM labels for memory requests when executing at EL2.

Configuration

This register has no effect if EL2 is not enabled in the current Security state.

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p0 is implemented) and EL3 is implemented, AArch64 System register MPAM2_EL2 bit [63] is architecturally mapped to AArch64 System register MPAM3_EL3[63].

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p0 is implemented, AArch64 System register MPAM2_EL2 bit [63] is architecturally mapped to AArch64 System register MPAM1_EL1[63].

This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAM2_EL2 are UNDEFINED.

Attributes

MPAM2_EL2 is a 64-bit register.

Field descriptions

MPAMEN, bit [63]

MPAM Enable. MPAM is enabled when MPAMEN == 1. When disabled, all PARTIDs and PMGs are output as their default value in the corresponding ID space.

MPAMEN	Meaning
0b0	The default PARTID and default PMG are output in MPAM information from all Exception levels.
0b1	MPAM information is output based on the MPAMn_ELx register for ELn according to the MPAM configuration.

If EL3 is implemented, this field is read-only and reads the current value of the read/write MPAM3_EL3.MPAMEN bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing this field has the following behavior:

- Access to this field is RAZ/WI if all of the following are true:
  - EL3 is implemented
  - MPAM3_EL3.MPAMEN == ‘0’
- Access to this field is RAO/WI if all of the following are true:
  - EL3 is implemented
  - MPAM3_EL3.MPAMEN == ‘1’
- When EL3 is not implemented, access to this field is RW.

Bits [62:59]

Reserved, RES0.

TIDR, bit [58]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMIDR_EL1.HAS_TIDR == '1':

TIDR traps accesses to MPAMIDR_EL1 from EL1 to EL2.

TIDR	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Trap accesses to MPAMIDR_EL1 from EL1 to EL2.

MPAMHCR_EL2.TRAP_MPAMIDR_EL1 == 1 also traps MPAMIDR_EL1 accesses from EL1 to EL2. If either TIDR or TRAP_MPAMIDR_EL1 are 1, accesses are trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [57]

Reserved, RES0.

ALTSP_HFC, bit [56]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == '1':

Hierarchical force of alternative PARTID space controls. When MPAM3_EL3.ALTSP_HEN is 0, ALTSP controls in MPAM2_EL2 have no effect. When MPAM3_EL3.ALTSP_HEN is 1, this bit selects whether the PARTIDs in MPAM1_EL1 and MPAM0_EL1 are in the primary (0) or alternative (1) PARTID space for the security state.

ALTSP_HFC	Meaning
0b0	When MPAM3_EL3.ALTSP_HEN is 1, the PARTID space of MPAM1_EL1.PARTID_I, MPAM1_EL1.PARTID_D, MPAM0_EL1.PARTID_I, and MPAM0_EL1.PARTID_D are in the primary PARTID space for the Security state.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ALTSP_HFC	Meaning
0b1	When MPAM3_EL3.ALTSP_HEN is 1, the PARTID space of MPAM1_EL1.PARTID_I, MPAM1_EL1.PARTID_D, MPAM0_EL1.PARTID_I, and MPAM0_EL1.PARTID_D are in the alternative PARTID space for the Security state.

This control has no effect when MPAM3_EL3.ALTSP_HEN is 0.

For more information, see In a PE with FEAT_RME, selection of primary or alternative PARTID space.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ALTSP_EL2, bit [55]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == '1':

Select alternative PARTID space for PARTIDs in MPAM2_EL2 when MPAM3_EL3.ALTSP_HEN is 1.

ALTSP_EL2	Meaning
0b0	When MPAM3_EL3.ALTSP_HEN is 1, selects the primary PARTID space for MPAM2_EL2.PARTID_I and MPAM2_EL2.PARTID_D.
0b1	When MPAM3_EL3.ALTSP_HEN is 1, selects the alternative PARTID space for MPAM2_EL2.PARTID_I and MPAM2_EL2.PARTID_D.

For more information, see In a PE with FEAT_RME, selection of primary or alternative PARTID space.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ALTSP_FRCD, bit [54]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == '1':

Alternative PARTID forced for PARTIDs in this register.

ALTSP_FRCD	Meaning
0b0	The PARTIDs in this register are using the primary PARTID space.
0b1	The PARTIDs in this register are using the alternative PARTID space.

This bit indicates that a higher Exception level has forced the PARTIDs in this register to use the alternative PARTID space defined for the current Security state. In EL2, it is also 1 when MPAM2_EL2.ALTSP_EL2 is 1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

For more information, see In a PE with FEAT_RME, selection of primary or alternative PARTID space.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When !UsePrimarySpaceEL2(), access to this field is RAO/WI.
- Otherwise, access to this field is RAZ/WI.

Otherwise:

Reserved, RES0.

Bits [53:51]

Reserved, RES0.

EnMPAMSM, bit [50]

When FEAT_SME is implemented:

Traps execution at EL1 of instructions that directly access the MPAMSM_EL1 register to EL2. The exception is reported using ESR_ELx.EC syndrome value `0x18`.

EnMPAMSM	Meaning
`0b0`	This control causes execution of these instructions at EL1 to be trapped.
`0b1`	This control does not cause execution of any instructions to be trapped.

This field has no effect on accesses to MPAMSM_EL1 from EL2 or EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRAPMPAM0EL1, bit [49]

Trap accesses from EL1 to the MPAM0_EL1 register trap to EL2.

TRAPMPAM0EL1	Meaning
`0b0`	Accesses to MPAM0_EL1 from EL1 are not trapped.
`0b1`	Accesses to MPAM0_EL1 from EL1 are trapped to EL2.

The reset behavior of this field is:

- On a Warm reset:
  - When EL3 is not implemented, this field resets to `'1'`.
  - When EL3 is implemented, this field resets to an architecturally UNKNOWN value.

TRAPMPAM1EL1, bit [48]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Trap accesses from EL1 to the MPAM1_EL1 register trap to EL2.

TRAPMPAM1EL1	Meaning
`0b0`	Accesses to MPAM1_EL1 from EL1 are not trapped.
`0b1`	Accesses to MPAM1_EL1 from EL1 are trapped to EL2.

The reset behavior of this field is:

- On a Warm reset:
  - When EL3 is not implemented, this field resets to `'1'`.
  - When EL3 is implemented, this field resets to an architecturally UNKNOWN value.

PMG_D, bits [47:40]

Performance monitoring group for data accesses.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PMG_I, bits [39:32]

Performance monitoring group for instruction accesses.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PARTID_D, bits [31:16]

Partition ID for data accesses, including load and store accesses, made from EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PARTID_I, bits [15:0]

Partition ID for instruction accesses made from EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAM2_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name `MPAM2_EL2` or `MPAM1_EL1` are not guaranteed to be ordered with respect to accesses using the other accessor name.

None of the fields in this register are permitted to be cached in a TLB.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAM2_EL2
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_MPAM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
     ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAM2_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAM2_EL2;
```

MSR MPAM2_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_MPAM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
     ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAM2_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAM2_EL2 = X[t, 64];
```

MRS <Xt>, MPAM1_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_MPAM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
        ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_MPAM) && MPAM2_EL2.TRAPMPAM1EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        X[t, 64] = NVMem[0x900];
    else
        X[t, 64] = MPAM1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
        ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        X[t, 64] = MPAM2_EL2;
    else
        X[t, 64] = MPAM1_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAM1_EL1;
```

MSR MPAM1_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b000

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !IsFeatureImplemented(FEAT_MPAM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
        ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_MPAM) && MPAM2_EL2.TRAPMPAM1EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x900] = X[t, 64];
    else
        MPAM1_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
        ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        MPAM2_EL2 = X[t, 64];
    else
        MPAM1_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAM1_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.12.4 MPAM3_EL3, MPAM3 Register (EL3)

The MPAM3_EL3 characteristics are:

Purpose

Holds information to generate MPAM labels for memory requests when executing at EL3.

Configuration

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p0 is implemented) and EL2 is implemented, AArch64 System register MPAM3_EL3 bit [63] is architecturally mapped to AArch64 System register MPAM2_EL2[63].

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p0 is implemented, AArch64 System register MPAM3_EL3 bit [63] is architecturally mapped to AArch64 System register MPAM1_EL1[63].

This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAM3_EL3 are UNDEFINED.

Attributes

MPAM3_EL3 is a 64-bit register.

Field descriptions

MPAMEN, bit [63]

MPAM Enable. MPAM is enabled when MPAMEN == 1. When disabled, all PARTIDs and PMGs are output as their default value in the corresponding ID space.

Values of this field are:

MPAMEN	Meaning
0b0	The default PARTID and default PMG are output in MPAM information when executing at any ELn.
0b1	MPAM information is output based on the MPAMn_ELx register for ELn according the MPAM configuration.

The reset behavior of this field is:

- On a Warm reset, this field resets to '0'.

Access to this field is RW.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

TRAPLOWER, bit [62]

Trap direct accesses to MPAM System registers that are not UNDEFINED from all ELn lower than EL3.

TRAPLOWER	Meaning
`0b0`	Do not force trapping of direct accesses of MPAM System registers to EL3.
`0b1`	Force direct accesses of MPAM System registers to trap to EL3.

Note

This trap is higher priority than any of the traps controlled by MPAM _EL2 registers.

The reset behavior of this field is:

- On a Warm reset, this field resets to `'1'`.

SDEFLT, bit [61]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMIDR_EL1.HAS_SDEFLT == '1':

SDEFLT overrides the PARTID and PMG with the default PARTID and default PMG when executing in the Secure state.

SDEFLT	Meaning
`0b0`	The PARTID and PMG are determined normally in the Secure state.
`0b1`	When executing in the Secure state, the PARTID is always PARTID 0, and the PMG is always PMG 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FORCE_NS, bit [60]

When FEAT_MPAMv0p1 is implemented and MPAMIDR_EL1.HAS_FORCE_NS == '1':

FORCE_NS forces MPAM_NS to always be 1 in the Secure state.

FORCE_NS	Meaning
`0b0`	MPAM_NS is 0 when executing in the Secure state.
`0b1`	MPAM_NS is 1 when executing in the Secure state.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

An implementation is permitted to have this field as RAO if the implementation does not support generating MPAM_NS as 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [59:58]

Reserved, RES0.

ALTSP_HEN, bit [57]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == '1':

Hierarchical enable for alternative PARTID space controls. Alternative PARTID space controls in MPAM2_EL2 have no effect when this field is zero.

ALTSP_HEN	Meaning
0b0	Disable alternative PARTID space controls in MPAM2_EL2. The PARTID space for PARTIDs in MPAM2_EL2, MPAM1_EL1, and MPAM0_EL1 is selected by MPAM3_EL3.ALTSP_HFC.
0b1	Enable alternative PARTID space controls in MPAM2_EL2 to control the PARTID space used for PARTIDs in MPAM2_EL2, MPAM1_EL1, and MPAM0_EL1.

For more information, see In a PE with FEAT_RME, selection of primary or alternative PARTID space.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ALTSP_HFC, bit [56]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == '1':

Hierarchical force of alternative PARTID space controls. When MPAM3_EL3.ALTSP_HEN is 0, the PARTID space for PARTIDs in MPAM2_EL2, MPAM1_EL1, and MPAM0_EL1 is selected by the value of this bit.

ALTSP_HFC	Meaning
0b0	When MPAM3_EL3.ALTSP_HEN is 0, the PARTID space of MPAM2_EL2.PARTID, MPAM1_EL1.PARTID and MPAM0_EL1.PARTID are the primary PARTID space for the security state.
0b1	When MPAM3_EL3.ALTSP_HEN is 0, the PARTID space of MPAM2_EL2.PARTID and MPAM1_EL1.PARTID and MPAM0_EL1.PARTID are the alternative PARTID space for the security state.

For more information, see In a PE with FEAT_RME, selection of primary or alternative PARTID space.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ALTSP_EL3, bit [55]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == '1':

Select alternative PARTID space for PARTIDs in MPAM3_EL3.

ALTSP_EL3	Meaning
0b0	Selects the primary PARTID space of MPAM3_EL3.PARTID_I and MPAM3_EL3.PARTID_D.
0b1	Selects the alternative PARTID space of MPAM3_EL3.PARTID_I and MPAM3_EL3.PARTID_D.

For more information, see In a PE with FEAT_RME, selection of primary or alternative PARTID space.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RES0.

Bits [54:53]

Reserved, RES0.

RT_ALTSP_NS, bit [52]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == '1':

Selects whether the alternative PARTID space for the Root security state is the Secure PARTID space or the Non-secure PARTID space. MPAM3_EL3.RT_ALTSP_NS selects the alternative PARTID space for the Root Security state when MPAM3_EL3.ALTSP_EL3 == 1.

RT_ALTSP_NS	Meaning
0b0	The alternative PARTID space in the Root security state is the Secure PARTID space.
0b1	The alternative PARTID space in the Root security state is the Non-secure PARTID space.

This field has no effect except in the Root security state (EL3).

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RES0.

Bits [51:48]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

PMG_D, bits [47:40]

Performance monitoring group for data accesses.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PMG_I, bits [39:32]

Performance monitoring group for instruction accesses.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PARTID_D, bits [31:16]

Partition ID for data accesses, including load and store accesses, made from EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PARTID_I, bits [15:0]

Partition ID for instruction accesses made from EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAM3_EL3

None of the fields in this register are permitted to be cached in a TLB.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAM3_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_MPAM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAM3_EL3;
```

```
MSR MPAM3_EL3, <Xt>
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b0101	0b000

```
if !IsFeatureImplemented(FEAT_MPAM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_FGWTE3) && FGWTE3_EL3.MPAM3_EL3 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAM3_EL3 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.12.5 MPAMBW0_EL1, MPAM PE-side Maximum-bandwidth Control Register (EL0)

The MPAMBW0_EL1 characteristics are:

Purpose

Enables software to configure a maximum fraction of memory bandwidth that the PE is permitted to use when executing at EL0 with its current PARTID.

Configuration

This register is present only when FEAT_MPAM_PE_BW_CTRL is implemented. Otherwise, direct accesses to MPAMBW0_EL1 are UNDEFINED.

Attributes

MPAMBW0_EL1 is a 64-bit register.

Field descriptions

HW_SCALE_ENABLE, bit [63]

When MPAMBWIDR_EL1.HAS_HW_SCALE == '1':

Enables hardware bandwidth scaling of the MPAMBW0_EL1.MAX value.

HW_SCALE_ENABLE	Meaning
0b0	PE-side memory bandwidth control hardware scaling in EL0 is disabled.
0b1	PE-side memory bandwidth control hardware scaling in EL0 is enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ENABLED, bit [62]

Enables the PE-side memory bandwidth control when in EL0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ENABLED	Meaning
0b0	The PE-side memory bandwidth control in EL0 is disabled.
0b1	The PE-side memory bandwidth control in EL0 is enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

HARDLIM, bit [61]

PE-side Maximum Bandwidth Limit Behavior Selection.

HARDLIM	Meaning
0b0	Soft limit: when MPAMBW0_EL1.MAX bandwidth is exceeded, the PE is unregulated unless the downstream memory path is saturated. It is IMPLEMENTATION DEFINED how hardware determines when the downstream memory path is saturated.
0b1	Hard limit: when MPAMBW0_EL1.MAX bandwidth is exceeded, the PE does not use any more bandwidth until the memory bandwidth for the PE falls below MPAMBW0_EL1.MAX.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When MPAMBWIDR_EL1.MAX_LIM == ‘00’, access to this field is RW.
- When MPAMBWIDR_EL1.MAX_LIM == ‘01’, access to this field is RAZ/WI.
- When MPAMBWIDR_EL1.MAX_LIM == ‘10’, access to this field is RAO/WI.

Bits [60:32]

Reserved, RES0.

MAX, bits [31:0]

When MPAMBWIDR_EL1.HAS_HW_SCALE == '1' and MPAMBW0_EL1.HW_SCALE_ENABLE == '1':

31	0
MAX	

MAX, bits [31:0]

Maximum memory bandwidth allocated to the PE when executing at EL0 with its current PARTID.

The value is represented as a multiplier of the available bandwidth for the PE. The value is represented in base-2 fixed-point format.

Bits [31:16] represent the integer part of the value.

Bits [15:(16 - MPAMBWIDR_EL1.BWA_WD)] represent the fractional part of the value. When MPAMBWIDR_EL1.BWA_WD indicates a width less than 16 bits, bits [(15 - MPAMBWIDR_EL1.BWA_WD):0] are RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When MPAMBWIDR_EL1.HAS_HW_SCALE == '0' or MPAMBW0_EL1.HW_SCALE_ENABLE == '0':

31 … 16	15 … 0
RES0	MAX

Bits [31:16]

Reserved, RES0.

MAX, bits [15:0]

Maximum memory bandwidth allocated to the PE when executing at EL0 with its current PARTID.

The value is represented as a fraction of the available bandwidth for the PE. The value is represented in base-2 fixed-point format.

Bits [15:(16 - MPAMBWIDR_EL1.BWA_WD)] represent the fractional part of the value. When MPAMBWIDR_EL1.BWA_WD indicates a width less than 16 bits, bits [(15 - MPAMBWIDR_EL1.BWA_WD):0] are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMBW0_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMBW0_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b101

```
if !IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
        ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAMBW2_EL2.nTRAP_MPAMBW0_EL1 == '0' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = MPAMBW0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
     ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAMBW0_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMBW0_EL1;
```

```
MSR MPAMBW0_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b101

```
if !IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
     ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAMBW2_EL2.nTRAP_MPAMBW0_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        MPAMBW0_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
     ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMBW0_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMBW0_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.12.6 MPAMBW1_EL1, MPAM PE-side Maximum-bandwidth Control Register (EL1)

The MPAMBW1_EL1 characteristics are:

Purpose

Enables software to configure a maximum fraction of memory bandwidth that the PE is permitted to use when executing at EL1 with its current PARTID.

Configuration

This register is present only when FEAT_MPAM_PE_BW_CTRL is implemented. Otherwise, direct accesses to MPAMBW1_EL1 are UNDEFINED.

Attributes

MPAMBW1_EL1 is a 64-bit register.

Field descriptions

HW_SCALE_ENABLE, bit [63]

When MPAMBWIDR_EL1.HAS_HW_SCALE == '1':

Enables hardware bandwidth scaling of the MPAMBW1_EL1.MAX value.

HW_SCALE_ENABLE	Meaning
0b0	PE-side memory bandwidth control hardware scaling in EL1 is disabled.
0b1	PE-side memory bandwidth control hardware scaling in EL1 is enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ENABLED, bit [62]

Enables the PE-side memory bandwidth control when in EL1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ENABLED	Meaning
0b0	The PE-side memory bandwidth control in EL1 is disabled.
0b1	The PE-side memory bandwidth control in EL1 is enabled.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

HARDLIM, bit [61]

PE-side Maximum Bandwidth Limit Behavior Selection.

HARDLIM	Meaning
0b0	Soft limit: when MPAMBW1_EL1.MAX bandwidth is exceeded, the PE is unregulated unless the downstream memory path is saturated. It is IMPLEMENTATION DEFINED how hardware determines when the downstream memory path is saturated.
0b1	Hard limit: when MPAMBW1_EL1.MAX bandwidth is exceeded, the PE does not use any more bandwidth until the memory bandwidth for the PE falls below MPAMBW1_EL1.MAX.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When MPAMBWIDR_EL1.MAX_LIM == ‘00’, access to this field is RW.
- When MPAMBWIDR_EL1.MAX_LIM == ‘01’, access to this field is RAZ/WI.
- When MPAMBWIDR_EL1.MAX_LIM == ‘10’, access to this field is RAO/WI.

Bits [60:32]

Reserved, RES0.

MAX, bits [31:0]

When MPAMBWIDR_EL1.HAS_HW_SCALE == '1' and MPAMBW1_EL1.HW_SCALE_ENABLE == '1':

31 … 0
MAX

MAX, bits [31:0]

Maximum memory bandwidth allocated to the PE when executing at EL1 with its current PARTID.

The value is represented as a multiplier of the available bandwidth for the PE. The value is represented in base-2 fixed-point format.

Bits [31:16] represent the integer part of the value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bits [15:(16 - MPAMBWIDR_EL1.BWA_WD)] represent the fractional part of the value. When MPAMBWIDR_EL1.BWA_WD indicates a width less than 16 bits, bits [(15 - MPAMBWIDR_EL1.BWA_WD):0] are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When MPAMBWIDR_EL1.HAS_HW_SCALE == '0' or MPAMBW1_EL1.HW_SCALE_ENABLE == '0':

31 … 16	15 … 0
RES0	MAX

Bits [31:16]

Reserved, RES0.

MAX, bits [15:0]

Maximum memory bandwidth allocated to the PE when executing at EL1 with its current PARTID.

The value is represented as a fraction of the available bandwidth for the PE. The value is represented in base-2 fixed-point format.

Bits [15:(16 - MPAMBWIDR_EL1.BWA_WD)] represent the fractional part of the value. When MPAMBWIDR_EL1.BWA_WD indicates a width less than 16 bits, bits [(15 - MPAMBWIDR_EL1.BWA_WD):0] are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMBW1_EL1

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the accessor name `MPAMBW1_EL1` or `MPAMBW1_EL12` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMBW1_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b100

```
if !IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
        ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EL2Enabled() && MPAMBW2_EL2.nTRAP_MPAMBW1_EL1 == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EffectiveHCR_EL2_NVx() IN {'111'} then
            X[t, 64] = NVMem[0x908];
        else
            X[t, 64] = MPAMBW1_EL1;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
            ↪'1' then
            UNDEFINED;
        elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif ELIsInHost(EL2) then
            X[t, 64] = MPAMBW2_EL2;
        else
            X[t, 64] = MPAMBW1_EL1;
    elsif PSTATE.EL == EL3 then
        X[t, 64] = MPAMBW1_EL1;
```

MSR MPAMBW1_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b100

```
if !IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
        ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EL2Enabled() && MPAMBW2_EL2.nTRAP_MPAMBW1_EL1 == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EffectiveHCR_EL2_NVx() IN {'111'} then
            NVMem[0x908] = X[t, 64];
        else
            MPAMBW1_EL1 = X[t, 64];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
            ↪'1' then
            UNDEFINED;
        elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif ELIsInHost(EL2) then
            MPAMBW2_EL2 = X[t, 64];
        else
            MPAMBW1_EL1 = X[t, 64];
    elsif PSTATE.EL == EL3 then
        MPAMBW1_EL1 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, MPAMBW1_EL12

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0101	0b100

```
if !IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        X[t, 64] = NVMem[0x908];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) &&
            ↪MPAM3_EL3.TRAPLOWER == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
                UNDEFINED;
            elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
                UNDEFINED;
            elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
                if EL3SDDUndef() then
                    UNDEFINED;
                else
                    AArch64.SystemAccessTrap(EL3, 0x18);
            elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
                if EL3SDDUndef() then
                    UNDEFINED;
                else
                    AArch64.SystemAccessTrap(EL3, 0x18);
            else
                X[t, 64] = MPAMBW1_EL1;
        else
            UNDEFINED;
    elsif PSTATE.EL == EL3 then
        if ELIsInHost(EL2) then
            X[t, 64] = MPAMBW1_EL1;
        else
            UNDEFINED;
```

When FEAT_VHE is implemented MSR MPAMBW1_EL12, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b101	0b1010	0b0101	0b100

```
if !IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() == '101' then
        NVMem[0x908] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ELIsInHost(EL2) then
        if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) &&
            ↪MPAM3_EL3.TRAPLOWER == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            MPAMBW1_EL1 = X[t, 64];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ELIsInHost(EL2) then
        MPAMBW1_EL1 = X[t, 64];
    else
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.12.7 MPAMBW2_EL2, MPAM PE-side Maximum-bandwidth Control Register (EL2)

The MPAMBW2_EL2 characteristics are:

Purpose

Enables software to configure a maximum fraction of memory bandwidth that the PE is permitted to use when executing at EL2 with its current PARTID.

Configuration

This register has no effect if EL2 is not enabled in the current Security state.

This register is present only when FEAT_MPAM_PE_BW_CTRL is implemented. Otherwise, direct accesses to MPAMBW2_EL2 are UNDEFINED.

Attributes

MPAMBW2_EL2 is a 64-bit register.

Field descriptions

HW_SCALE_ENABLE, bit [63]

When MPAMBWIDR_EL1.HAS_HW_SCALE == '1':

Enables hardware bandwidth scaling of the MPAMBW2_EL2.MAX value.

HW_SCALE_ENABLE	Meaning
0b0	PE-side memory bandwidth control hardware scaling in EL2 is disabled.
0b1	PE-side memory bandwidth control hardware scaling in EL2 is enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ENABLED, bit [62]

Enables the PE-side memory bandwidth control when in EL2.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ENABLED	Meaning
0b0	The PE-side memory bandwidth control in EL2 is disabled.
0b1	The PE-side memory bandwidth control in EL2 is enabled.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

HARDLIM, bit [61]

PE-side Maximum Bandwidth Limit Behavior Selection.

HARDLIM	Meaning
0b0	Soft limit: when MPAMBW2_EL2.MAX bandwidth is exceeded, the PE is unregulated unless the downstream memory path is saturated. It is IMPLEMENTATION DEFINED how hardware determines when the downstream memory path is saturated.
0b1	Hard limit: when MPAMBW2_EL2.MAX bandwidth is exceeded, the PE does not use any more bandwidth until the memory bandwidth for the PE falls below MPAMBW2_EL2.MAX.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When MPAMBWIDR_EL1.MAX_LIM == ‘00’, access to this field is RW.
- When MPAMBWIDR_EL1.MAX_LIM == ‘01’, access to this field is RAZ/WI.
- When MPAMBWIDR_EL1.MAX_LIM == ‘10’, access to this field is RAO/WI.

Bits [60:53]

Reserved, RES0.

nTRAP_MPAMBWIDR_EL1, bit [52]

Traps accesses to MPAMBWIDR_EL1 from EL1 to EL2.

nTRAP_MPAMBWIDR_EL1	Meaning
0b0	Accesses to MPAMBWIDR_EL1 from EL1 are trapped to EL2 with EC syndrome value 0x18.
0b1	Accesses to MPAMBWIDR_EL1 from EL1 are not trapped by this mechanism.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

nTRAP_MPAMBW0_EL1, bit [51]

Traps accesses to MPAMBW0_EL1 from EL1 to EL2.

nTRAP_MPAMBW0_EL1	Meaning
`0b0`	Accesses to MPAMBW0_EL1 from EL1 are trapped to EL2 with EC syndrome value `0x18`.
`0b1`	Accesses to MPAMBW0_EL1 from EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

nTRAP_MPAMBW1_EL1, bit [50]

Traps accesses to MPAMBW1_EL1 from EL1 to EL2.

nTRAP_MPAMBW1_EL1	Meaning
`0b0`	Accesses to MPAMBW1_EL1 from EL1 are trapped to EL2 with EC syndrome value `0x18`.
`0b1`	Accesses to MPAMBW1_EL1 from EL1 are not trapped by this mechanism.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

nTRAP_MPAMBWSM_EL1, bit [49]

When FEAT_SME is implemented:

Traps accesses to MPAMBWSM_EL1 from EL1 to EL2.

nTRAP_MPAMBWSM_EL1	Meaning
`0b0`	Accesses to MPAMBWSM_EL1 from EL1 are trapped to EL2 with EC syndrome value `0x18`.
`0b1`	Accesses to MPAMBWSM_EL1 from EL1 are not trapped by this mechanism.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to `'0'`.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [48:32]

Reserved, RES0.

MAX, bits [31:0]

When MPAMBWIDR_EL1.HAS_HW_SCALE == '1' and MPAMBW2_EL2.HW_SCALE_ENABLE == '1':

31 … 0
MAX

MAX, bits [31:0]

Maximum memory bandwidth allocated to the PE when executing at EL2 with its current PARTID.

The value is represented as a multiplier of the available bandwidth for the PE. The value is represented in base-2 fixed-point format.

Bits [31:16] represent the integer part of the value.

Bits [15:(16 - MPAMBWIDR_EL1.BWA_WD)] represent the fractional part of the value. When MPAMBWIDR_EL1.BWA_WD indicates a width less than 16 bits, bits [(15 - MPAMBWIDR_EL1.BWA_WD):0] are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When MPAMBWIDR_EL1.HAS_HW_SCALE == '0' or MPAMBW2_EL2.HW_SCALE_ENABLE == '0':

31 … 16	15 … 0
RES0	MAX

Bits [31:16]

Reserved, RES0.

MAX, bits [15:0]

Maximum memory bandwidth allocated to the PE when executing at EL2 with its current PARTID.

The value is represented as a fraction of the available bandwidth for the PE. The value is represented in base-2 fixed-point format.

Bits [15:(16 - MPAMBWIDR_EL1.BWA_WD)] represent the fractional part of the value. When MPAMBWIDR_EL1.BWA_WD indicates a width less than 16 bits, bits [(15 - MPAMBWIDR_EL1.BWA_WD):0] are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Accessing MPAMBW2_EL2

When the Effective value of HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL2 using the accessor name `MPAMBW2_EL2` or `MPAMBW1_EL1` are not guaranteed to be ordered with respect to accesses using the other accessor name.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMBW2_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0101	0b100

```
if !IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
     ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAMBW2_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMBW2_EL2;
```

```
MSR MPAMBW2_EL2, <Xt>
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0101	0b100

```
if !IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
        ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMBW2_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMBW2_EL2 = X[t, 64];
```

When FEAT_VHE is implemented MRS <Xt>, MPAMBW1_EL1

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b100

```
if !IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) &&
        ↪MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            UNDEFINED;
        elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EL2Enabled() && MPAMBW2_EL2.nTRAP_MPAMBW1_EL1 == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EffectiveHCR_EL2_NVx() IN {'111'} then
            X[t, 64] = NVMem[0x908];
        else
            X[t, 64] = MPAMBW1_EL1;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) &&
             ↪MPAM3_EL3.TRAPLOWER == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif ELIsInHost(EL2) then
            X[t, 64] = MPAMBW2_EL2;
        else
            X[t, 64] = MPAMBW1_EL1;
    elsif PSTATE.EL == EL3 then
        X[t, 64] = MPAMBW1_EL1;
```

When FEAT_VHE is implemented MSR MPAMBW1_EL1, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b100

```
if !IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) &&
         ↪MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAMBW2_EL2.nTRAP_MPAMBW1_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EffectiveHCR_EL2_NVx() IN {'111'} then
        NVMem[0x908] = X[t, 64];
    else
        MPAMBW1_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) &&
         ↪MPAM3_EL3.TRAPLOWER == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif ELIsInHost(EL2) then
        MPAMBW2_EL2 = X[t, 64];
    else
        MPAMBW1_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMBW1_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.12.8 MPAMBW3_EL3, MPAM PE-side Maximum-bandwidth Control Register (EL3)

The MPAMBW3_EL3 characteristics are:

Purpose

Enables software to configure a maximum fraction of memory bandwidth that the PE is permitted to use when executing at EL3 with its current PARTID.

Configuration

This register is present only when FEAT_MPAM_PE_BW_CTRL is implemented. Otherwise, direct accesses to MPAMBW3_EL3 are UNDEFINED.

Attributes

MPAMBW3_EL3 is a 64-bit register.

Field descriptions

HW_SCALE_ENABLE, bit [63]

When MPAMBWIDR_EL1.HAS_HW_SCALE == '1':

Enables hardware bandwidth scaling of the MPAMBW3_EL3.MAX value.

HW_SCALE_ENABLE	Meaning
0b0	PE-side memory bandwidth control hardware scaling in EL3 is disabled.
0b1	PE-side memory bandwidth control hardware scaling in EL3 is enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ENABLED, bit [62]

Enables the PE-side memory bandwidth control when in EL3.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ENABLED	Meaning
0b0	The PE-side memory bandwidth control in EL3 is disabled.
0b1	The PE-side memory bandwidth control in EL3 is enabled.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3, this field resets to '0'.

HARDLIM, bit [61]

PE-side Maximum Bandwidth Limit Behavior Selection.

HARDLIM	Meaning
0b0	Soft limit: when MPAMBW3_EL3.MAX bandwidth is exceeded, the PE is unregulated unless the downstream memory path is saturated. It is IMPLEMENTATION DEFINED how hardware determines when the downstream memory path is saturated.
0b1	Hard limit: when MPAMBW3_EL3.MAX bandwidth is exceeded, the PE does not use any more bandwidth until the memory bandwidth for the PE falls below MPAMBW3_EL3.MAX.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When MPAMBWIDR_EL1.MAX_LIM == ‘00’, access to this field is RW.
- When MPAMBWIDR_EL1.MAX_LIM == ‘01’, access to this field is RAZ/WI.
- When MPAMBWIDR_EL1.MAX_LIM == ‘10’, access to this field is RAO/WI.

Bits [60:50]

Reserved, RES0.

nTRAPLOWER, bit [49]

Traps accesses to MPAMBW2_EL2, MPAMBWCAP_EL2, MPAMBW1_EL1, MPAMBW0_EL1, MPAMBWSM_EL1, MPAMBWIDR_EL1 from any lower EL to EL3.

nTRAPLOWER	Meaning
0b0	Accesses to MPAMBW2_EL2, MPAMBWCAP_EL2, MPAMBW1_EL1, MPAMBW0_EL1, MPAMBWSM_EL1, MPAMBWIDR_EL1 from EL1 are trapped to EL3 with EC syndrome value 0x18.
0b1	Accesses to MPAMBW2_EL2, MPAMBWCAP_EL2, MPAMBW1_EL1, MPAMBW0_EL1, MPAMBWSM_EL1, MPAMBWIDR_EL1 from EL1 are not trapped by this mechanism.

Note

This trap is higher priority than any of the traps controlled by MPAM _EL2 registers.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL3, this field resets to `'0'`.

Bits [48:32]

Reserved, RES0.

MAX, bits [31:0]

When MPAMBWIDR_EL1.HAS_HW_SCALE == '1' and MPAMBW3_EL3.HW_SCALE_ENABLE == '1':

31 … 0
MAX

MAX, bits [31:0]

Maximum memory bandwidth allocated to the PE when executing at EL3 with its current PARTID.

The value is represented as a multiplier of the available bandwidth for the PE. The value is represented in base-2 fixed-point format.

Bits [31:16] represent the integer part of the value.

Bits [15:(16 - MPAMBWIDR_EL1.BWA_WD)] represent the fractional part of the value. When MPAMBWIDR_EL1.BWA_WD indicates a width less than 16 bits, bits [(15 - MPAMBWIDR_EL1.BWA_WD):0] are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When MPAMBWIDR_EL1.HAS_HW_SCALE == '0' or MPAMBW3_EL3.HW_SCALE_ENABLE == '0':

31 … 16	15 … 0
RES0	MAX

Bits [31:16]

Reserved, RES0.

MAX, bits [15:0]

Maximum memory bandwidth allocated to the PE when executing at EL3 with its current PARTID.

The value is represented as a fraction of the available bandwidth for the PE. The value is represented in base-2 fixed-point format.

Bits [15:(16 - MPAMBWIDR_EL1.BWA_WD)] represent the fractional part of the value. When MPAMBWIDR_EL1.BWA_WD indicates a width less than 16 bits, bits [(15 - MPAMBWIDR_EL1.BWA_WD):0] are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMBW3_EL3

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, MPAMBW3_EL3
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b0101	0b100

```
if !IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMBW3_EL3;
```

```
MSR MPAMBW3_EL3, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b110	0b1010	0b0101	0b100

```
if !IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    MPAMBW3_EL3 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.12.9 MPAMBWCAP_EL2, MPAM PE-side Maximum-bandwidth Limit Virtualization Register

The MPAMBWCAP_EL2 characteristics are:

Purpose

Allows software executing at EL2 to provide MPAMBWCAP_EL2.CAP as an upper bound to MPAMBW1_EL1.MAX.

If FEAT_SME is implemented, the upper bound also applies to MPAMBWSM_EL1.MAX when executing at EL1: the maximum bandwidth allowed for the PE is MIN(MPAMBWSM_EL1.MAX, MPAMBWCAP_EL2.CAP).

If the Effective value of HCR_EL2.{E2H,TGE} is not {1,1}:

- The upper bound also applies to MPAMBW0_EL1.MAX: the maximum bandwidth allowed for the PE is MIN(MPAMBW0_EL1.MAX, MPAMBWCAP_EL2.CAP).
- If FEAT_SME is implemented, the upper bound also applies to MPAMBWSM_EL1.MAX when executing at EL0: the maximum bandwidth allowed for the PE is MIN(MPAMBWSM_EL1.MAX, MPAMBWCAP_EL2.CAP).

If MPAMBWCAP_EL2.ENABLED is 1, a PARTID that has used more than min(CAP,MAX) is given no access to additional bandwidth.

Configuration

This register has no effect if EL2 is not enabled in the current Security state.

This register is present only when FEAT_MPAM_PE_BW_CTRL is implemented and MPAMIDR_EL1.HAS_HCR == '1'. Otherwise, direct accesses to MPAMBWCAP_EL2 are UNDEFINED.

Attributes

MPAMBWCAP_EL2 is a 64-bit register.

Field descriptions

HW_SCALE_ENABLE, bit [63]

When MPAMBWIDR_EL1.HAS_HW_SCALE == '1':

Enables hardware bandwidth scaling of the MPAMBWCAP_EL2.CAP value.

HW_SCALE_ENABLE	Meaning
0b0	PE-side memory bandwidth control hardware scaling for EL2 capping is disabled.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

HW_SCALE_ENABLE	Meaning
0b1	PE-side memory bandwidth control hardware scaling for EL2 capping is enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ENABLED, bit [62]

Enables the PE-side memory bandwidth control capping by EL2.

ENABLED	Meaning
0b0	The PE-side memory bandwidth control capping by EL2 is disabled.
0b1	The PE-side memory bandwidth control capping by EL2 is enabled.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL2, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

Bits [61:32]

Reserved, RES0.

CAP, bits [31:0]

When MPAMBWIDR_EL1.HAS_HW_SCALE == '1' and MPAMBWCAP_EL2.HW_SCALE_ENABLE == '1':

31	0
CAP	

CAP, bits [31:0]

Upper bound to the maximum memory bandwidth allocated to the current PARTID in MPAMBW1_EL1.MAX, MPAMBW0_EL1.MAX and MPAMBWSM_EL1.MAX.

The value is represented as a multiplier of the available bandwidth for the PE. The value is represented in base-2 fixed-point format.

Bits [31:16] represent the integer part of the value.

Bits [15:(16 - MPAMBWIDR_EL1.BWA_WD)] represent the fractional part of the value. When MPAMBWIDR_EL1.BWA_WD indicates a width less than 16 bits, bits [(15 - MPAMBWIDR_EL1.BWA_WD):0] are RES0.

The value set in the MAX field must be less than or equal to this upper bound.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When MPAMBWIDR_EL1.HAS_HW_SCALE == '0' or MPAMBWCAP_EL2.HW_SCALE_ENABLE == '0':

31 … 16	15 … 0
RES0	CAP

Bits [31:16]

Reserved, RES0.

CAP, bits [15:0]

Upper bound to the maximum memory bandwidth allocated to the current PARTID in MPAMBW1_EL1.MAX, MPAMBW0_EL1.MAX and MPAMBWSM_EL1.MAX.

The value is represented as a fraction of the available bandwidth for the PE. The value is represented in base-2 fixed-point format.

Bits [15:(16 - MPAMBWIDR_EL1.BWA_WD)] represent the fractional part of the value. When MPAMBWIDR_EL1.BWA_WD indicates a width less than 16 bits, bits [(15 - MPAMBWIDR_EL1.BWA_WD):0] are RES0.

The value set in the MAX field must be less than or equal to this upper bound.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMBWCAP_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMBWCAP_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0101	0b110

```
if !(IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) && MPAMIDR_EL1.HAS_HCR == '1') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x910];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                AArch64.SystemAccessTrap(EL2, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
            ↪'1' then
            UNDEFINED;
        elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            X[t, 64] = MPAMBWCAP_EL2;
    elsif PSTATE.EL == EL3 then
        X[t, 64] = MPAMBWCAP_EL2;
```

MSR MPAMBWCAP_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0101	0b110

```
if !(IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) && MPAMIDR_EL1.HAS_HCR == '1') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x910] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
        ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMBWCAP_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMBWCAP_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.12.10 MPAMBWIDR_EL1, MPAM PE-side Bandwidth Controls ID Register

The MPAMBWIDR_EL1 characteristics are:

Purpose

Indicates the supported PE-side memory bandwidth parameter values.

Configuration

This register is present only when FEAT_MPAM_PE_BW_CTRL is implemented. Otherwise, direct accesses to MPAMBWIDR_EL1 are UNDEFINED.

Attributes

MPAMBWIDR_EL1 is a 64-bit register.

Field descriptions

HAS_HW_SCALE, bit [63]

Indicates whether hardware support for auto-scaling of MPAMBWn_ELx.MAX, MPAMBWSM_EL1.MAX and MPAMBWCAP_EL2.CAP limits is available.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_HW_SCALE	Meaning
0b0	Hardware support for auto-scaling is not implemented.
0b1	Hardware support for auto-scaling is implemented.

Access to this field is RO.

Bits [62:32]

Reserved, RES0.

MAX_LIM, bits [31:30]

Indicates the implemented maximum-bandwidth limit partitioning behaviors.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

MAX_LIM	Meaning
0b00	Both soft limit and hard limit behaviors are implemented.
0b01	Soft limit behavior is implemented.
0b10	Hard limit behavior is implemented.
0b11	Reserved.

Access to this field is RO.

Bits [29:6]

Reserved, RES0.

BWA_WD, bits [5:0]

Indicates the number of implemented bits in the bandwidth allocation fields MPAMBWn_ELx.MAX, MPAMBWSM_EL1.MAX and MPAMBWCAP_EL2.CAP.

The value of this field is an IMPLEMENTATION DEFINED choice of:

BWA_WD	Meaning
0b000001..0b010000	Number of implemented bits in the bandwidth allocation fields.

Access to this field is RO.

Accessing MPAMBWIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMBWIDR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0100	0b101

```
if !IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
        ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAMBW2_EL2.nTRAP_MPAMBWIDR_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = MPAMBWIDR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
     ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAMBWIDR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMBWIDR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.12.11 MPAMBWSM_EL1, MPAM Streaming Mode Bandwidth Control Register (EL1)

The MPAMBWSM_EL1 characteristics are:

Purpose

Enables software to configure a maximum fraction of memory bandwidth that the PE is permitted for SME memory accesses labelled with values from MPAMSM_EL1.

Configuration

This register is present only when FEAT_MPAM_PE_BW_CTRL is implemented and FEAT_SME is implemented. Otherwise, direct accesses to MPAMBWSM_EL1 are UNDEFINED.

Attributes

MPAMBWSM_EL1 is a 64-bit register.

Field descriptions

HW_SCALE_ENABLE, bit [63]

When MPAMBWIDR_EL1.HAS_HW_SCALE == '1':

Enables hardware bandwidth scaling of the MPAMBWSM_EL1.MAX value.

HW_SCALE_ENABLE	Meaning
0b0	PE-side memory bandwidth control hardware scaling for streaming PARTIDs is disabled.
0b1	PE-side memory bandwidth control hardware scaling for streaming PARTIDs is enabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ENABLED, bit [62]

Enables the PE-side memory bandwidth control for streaming PARTIDs.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ENABLED	Meaning
0b0	PE-side memory bandwidth control for streaming PARTIDs is disabled.
0b1	PE-side memory bandwidth control for streaming PARTIDs is enabled.

The reset behavior of this field is:

- On a Warm reset:
  - When the highest implemented Exception level is EL1, this field resets to '0'.
  - Otherwise, this field resets to an architecturally UNKNOWN value.

HARDLIM, bit [61]

PE-side Maximum-bandwidth Limit Behavior Selection.

HARDLIM	Meaning
0b0	Soft limit: when MPAMBWSM_EL1.MAX bandwidth is exceeded, the PE is unregulated unless the downstream memory path is saturated. It is IMPLEMENTATION DEFINED how hardware determines when the downstream memory path is saturated.
0b1	Hard limit: when MPAMBWSM_EL1.MAX bandwidth is exceeded, the PE does not use any more bandwidth until the memory bandwidth for the PE falls below MPAMBWSM_EL1.MAX.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When MPAMBWIDR_EL1.MAX_LIM == ‘00’, access to this field is RW.
- When MPAMBWIDR_EL1.MAX_LIM == ‘01’, access to this field is RAZ/WI.
- When MPAMBWIDR_EL1.MAX_LIM == ‘10’, access to this field is RAO/WI.

Bits [60:32]

Reserved, RES0.

MAX, bits [31:0]

When MPAMBWIDR_EL1.HAS_HW_SCALE == '1' and MPAMBWSM_EL1.HW_SCALE_ENABLE == '1':

31 … 0
MAX

MAX, bits [31:0]

Maximum memory bandwidth allocated to the partition selected by MPAMSM_EL1.PARTID_D.

The value is represented as a multiplier of the available bandwidth for the PE. The value is represented in base-2 fixed-point format.

Bits [31:16] represent the integer part of the value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Bits [15:(16 - MPAMBWIDR_EL1.BWA_WD)] represent the fractional part of the value. When MPAMBWIDR_EL1.BWA_WD indicates a width less than 16 bits, bits [(15 - MPAMBWIDR_EL1.BWA_WD):0] are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When MPAMBWIDR_EL1.HAS_HW_SCALE == '0' or MPAMBWSM_EL1.HW_SCALE_ENABLE == '0':

31 … 16	15 … 0
RES0	MAX

Bits [31:16]

Reserved, RES0.

MAX, bits [15:0]

Maximum memory bandwidth allocated to the partition selected by MPAMSM_EL1.PARTID_D.

The value is represented as a fraction of the available bandwidth for the PE. The value is represented in base-2 fixed-point format.

Bits [15:(16 - MPAMBWIDR_EL1.BWA_WD)] represent the fractional part of the value. When MPAMBWIDR_EL1.BWA_WD indicates a width less than 16 bits, bits [(15 - MPAMBWIDR_EL1.BWA_WD):0] are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMBWSM_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMBWSM_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b111

```
if !(IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) && IsFeatureImplemented(FEAT_SME)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
        ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAMBW2_EL2.nTRAP_MPAMBWSM_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = MPAMBWSM_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
     ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAMBWSM_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMBWSM_EL1;
```

```
MSR MPAMBWSM_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b111

```
if !(IsFeatureImplemented(FEAT_MPAM_PE_BW_CTRL) && IsFeatureImplemented(FEAT_SME)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
     ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAMBW2_EL2.nTRAP_MPAMBWSM_EL1 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        MPAMBWSM_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
        ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && EL3SDDUndefPriority() && MPAMBW3_EL3.nTRAPLOWER == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MPAMBW3_EL3.nTRAPLOWER == '0' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMBWSM_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMBWSM_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.12.12 MPAMHCR_EL2, MPAM Hypervisor Control Register (EL2)

The MPAMHCR_EL2 characteristics are:

Purpose

Controls the PARTID virtualization features of MPAM.

Configuration

This register has no effect if EL2 is not enabled in the current Security state.

This register is present only when FEAT_MPAM is implemented and MPAMIDR_EL1.HAS_HCR == '1'. Otherwise, direct accesses to MPAMHCR_EL2 are UNDEFINED.

Attributes

MPAMHCR_EL2 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

TRAP_MPAMIDR_EL1, bit [31]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p0 is implemented:

Trap accesses from EL1 to MPAMIDR_EL1 to EL2.

TRAP_MPAMIDR_EL1	Meaning
0b0	This control does not cause any instructions to be trapped.
0b1	Direct accesses to MPAMIDR_EL1 from EL1 are trapped to EL2.

The reset behavior of this field is:

- On a Warm reset:
  - When EL3 is not implemented, this field resets to '1'.
  - When EL3 is implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [30:9]

Reserved, RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

GSTAPP_PLK, bit [8]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p0 is implemented:

Make the PARTIDs at EL0 the same as the PARTIDs at EL1. When executing at EL0, EL2 is enabled, HCR_EL2.TGE == 0 and GSTAPP_PLK = 1, MPAM1_EL1 is used instead of MPAM0_EL1 to generate MPAM labels for memory requests.

GSTAPP_PLK	Meaning
0b0	MPAM0_EL1 is used to generate MPAM labels when executing at EL0.
0b1	MPAM1_EL1 is used to generate MPAM labels when executing at EL0 with EL2 enabled and HCR_EL2.TGE == 0. Otherwise MPAM0_EL1 is used.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [7:2]

Reserved, RES0.

EL1_VPMEN, bit [1]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p0 is implemented:

Enable virtualized identifiers in MPAM1_EL1 when executing at EL1. This bit also enables virtualized identifiers when MPAM1_EL1 is used to generate MPAM identifiers for memory requests at EL0 due to GSTAPP_PLK == 1.

EL1_VPMEN	Meaning
0b0	MPAM1_EL1.PARTID_I and MPAM1_EL1.PARTID_D are physical PARTIDs used to label memory system requests.
0b1	MPAM1_EL1.PARTID_I and MPAM1_EL1.PARTID_D are virtual PARTIDs used to index the MPAMVPMn_EL2.PhyPARTID fields to map the virtual PARTID into a physical PARTID to label memory system requests.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL0_VPMEN, bit [0]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p0 is implemented:

Enable virtualized identifiers in MPAM0_EL1 unless the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}.

When the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, EL0_VPMEN is ignored and MPAM0_EL1 identifiers are not virtualized.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

When MPAMHCR_EL2.GSTAPP_PLK == 1 and HCR_EL2.TGE == 0, MPAM1_EL1 is used as the source of identifiers and the virtualization of identifiers in MPAM1_EL1 is controlled by MPAMHCR_EL2.EL1_VPMEN.

EL0_VPMEN	Meaning
0b0	MPAM0_EL1.PARTID_I and MPAM0_EL1.PARTID_D are physical PARTIDs used to label memory system requests.
0b1	MPAM0_EL1.PARTID_I and MPAM0_EL1.PARTID_D are virtual PARTIDs used to index the MPAMVPMn_EL2.PhyPARTID fields to map the virtual PARTID into a physical PARTID to label memory system requests.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing MPAMHCR_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMHCR_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_MPAM) && MPAMIDR_EL1.HAS_HCR == '1') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x930];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
     ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        X[t, 64] = MPAMHCR_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMHCR_EL2;
```

MSR MPAMHCR_EL2, <Xt>

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0100	0b000

```
if !(IsFeatureImplemented(FEAT_MPAM) && MPAMIDR_EL1.HAS_HCR == '1') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x930] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
     ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMHCR_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMHCR_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.12.13 MPAMIDR_EL1, MPAM ID Register (EL1)

The MPAMIDR_EL1 characteristics are:

Purpose

Indicates the maximum PARTID and PMG values supported in the implementation and the support for other optional features.

Configuration

This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAMIDR_EL1 are UNDEFINED.

Attributes

MPAMIDR_EL1 is a 64-bit register.

Field descriptions

Bits [63:62]

Reserved, RES0.

HAS_SDEFLT, bit [61]

HAS_SDEFLT indicates support for MPAM3_EL3.SDEFLT bit.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_SDEFLT	Meaning
0b0	The SDEFLT bit is not implemented in MPAM3_EL3.
0b1	The SDEFLT bit is implemented in MPAM3_EL3.

When MPAM3_EL3.SDEFLT == 1, accesses from the Secure Execution state use the default PARTID, PARTID == 0.

Access to this field is RO.

HAS_FORCE_NS, bit [60]

HAS_FORCE_NS indicates support for MPAM3_EL3.FORCE_NS bit.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

HAS_FORCE_NS	Meaning
0b0	The FORCE_NS bit is not implemented in MPAM3_EL3.
0b1	The FORCE_NS bit is implemented in MPAM3_EL3.

When MPAM3_EL3.FORCE_NS == 1, accesses from the Secure Execution state have MPAM_NS == 1.

Access to this field is RO.

SP4, bit [59]

Supports 4 MPAM PARTID spaces.

The value of this field is an IMPLEMENTATION DEFINED choice of:

SP4	Meaning
0b0	MPAM supports 2 PARTID spaces.
0b1	MPAM supports 4 PARTID spaces.

Access to this field is RO.

HAS_TIDR, bit [58]

HAS_TIDR indicates support for MPAM2_EL2.TIDR bit.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_TIDR	Meaning
0b0	The TIDR bit is not implemented in MPAM2_EL2.
0b1	The TIDR bit is implemented in MPAM2_EL2.

Note

Arm recommends that when the MPAM version is MPAM v0.1 or MPAM v1.1, MPAMIDR_EL1.HAS_TIDR is 1 and that the MPAM2_EL2.TIDR field is implemented.

Access to this field is RO.

HAS_ALTSP, bit [57]

HAS_ALTSP indicates support for alternative PARTID spaces.

The value of this field is an IMPLEMENTATION DEFINED choice of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

HAS_ALTSP	Meaning
0b0	Alternative PARTID spaces are not implemented.
0b1	Alternative PARTID spaces are implemented with control bits in MPAM3_EL3 and MPAM2_EL2.

Access to this field is RO.

HAS_BW_CTRL, bit [56]

HAS_BW_CTRL indicates support for PE-side bandwidth controls.

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_BW_CTRL	Meaning
0b0	PE-side bandwidth controls are not implemented.
0b1	PE-side bandwidth controls are implemented.

FEAT_MPAM_PE_BW_CTRL implements the functionality identified by the value 0b1.

Access to this field is RO.

Bits [55:40]

Reserved, RES0.

PMG_MAX, bits [39:32]

The largest value of PMG that the implementation can generate. The PMG_I and PMG_D fields of every MPAMn_ELx must implement at least enough bits to represent PMG_MAX.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Bits [31:21]

Reserved, RES0.

VPMR_MAX, bits [20:18]

When MPAMIDR_EL1.HAS_HCR == '1':

Indicates the maximum register index n for the MPAMVPM<n>_EL2 registers.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Otherwise:

Reserved, RAZ.

HAS_HCR, bit [17]

HAS_HCR indicates that the PE implementation supports MPAM virtualization, including MPAMHCR_EL2, MPAMVPMV_EL2, and MPAMVPM<n>_EL2 with n in the range 0 to VPMR_MAX. Must be 0 if EL2 is not implemented in either Security state.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The value of this field is an IMPLEMENTATION DEFINED choice of:

HAS_HCR	Meaning
0b0	MPAM virtualization is not supported.
0b1	MPAM virtualization is supported.

Access to this field is RO.

Bit [16]

Reserved, RES0.

PARTID_MAX, bits [15:0]

The largest value of PARTID that the implementation can generate. The PARTID_I and PARTID_D fields of every MPAMn_ELx must implement at least enough bits to represent PARTID_MAX.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is RO.

Accessing MPAMIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMIDR_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0100	0b100

```
if !IsFeatureImplemented(FEAT_MPAM) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
        ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_MPAM) && MPAMIDR_EL1.HAS_HCR == '1' &&
        ↪MPAMHCR_EL2.TRAP_MPAMIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_MPAM) && MPAMIDR_EL1.HAS_TIDR == '1' &&
        ↪MPAM2_EL2.TIDR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = MPAMIDR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
        ↪'1' then
        UNDEFINED;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAMIDR_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMIDR_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.12.14 MPAMSM_EL1, MPAM Streaming Mode Register

The MPAMSM_EL1 characteristics are:

Purpose

Holds information to generate MPAM labels for memory requests that are:

- Issued due to the execution of SME load and store instructions.
- Issued when the PE is in Streaming SVE mode due to the execution of SVE and SIMD&FP load and store instructions and SVE prefetch instructions.

If an implementation uses a shared SMCU, then the MPAM labels in this register have precedence over the labels in MPAM0_EL1, MPAM1_EL1, MPAM2_EL2, and MPAM3_EL3.

If an implementation includes an SMCU that is not shared with other PEs, then it is IMPLEMENTATION DEFINED whether the MPAM labels in this register have precedence over the labels in MPAM0_EL1, MPAM1_EL1, MPAM2_EL2, and MPAM3_EL3.

The MPAM labels in this register are only used if MPAM1_EL1.MPAMEN (MPAM 1.0) or MPAMCTL_EL1.MPAMEN (MPAM 2.0) is 1.

For memory requests issued from EL0, the MPAM PARTID in this register is virtual and mapped into a physical PARTID when all of the following are true:

- EL2 is implemented and enabled in the current Security state, and the Effective value of HCR_EL2.{E2H, TGE} is not {1, 1}.
- The MPAM virtualization option is implemented and MPAMHCR_EL2.EL0_VPMEN is 1.

For memory requests issued from EL1, the MPAM PARTID in this register is virtual and mapped into a physical PARTID when all of the following are true:

- EL2 is implemented and enabled in the current Security state.
- The MPAM virtualization option is implemented and MPAMHCR_EL2.EL1_VPMEN is 1.

Configuration

This register is present only when FEAT_MPAM is implemented and FEAT_SME is implemented. Otherwise, direct accesses to MPAMSM_EL1 are UNDEFINED.

Attributes

MPAMSM_EL1 is a 64-bit register.

Field descriptions

Bits [63:48]

Reserved, RES0.

PMG_D, bits [47:40]

Performance monitoring group property for PARTID_D.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [39:32]

Reserved, RES0.

PARTID_D, bits [31:16]

Partition ID for requests issued due to the execution at any Exception level of SME load and store instructions and, when the PE is in Streaming SVE mode, SVE and SIMD&FP load and store instructions and SVE prefetch instructions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [15:0]

Reserved, RES0.

Accessing MPAMSM_EL1

None of the fields in this register are permitted to be cached in a TLB.

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMSM_EL1
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b011

```
if !(IsFeatureImplemented(FEAT_MPAM) && IsFeatureImplemented(FEAT_SME)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
        ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_MPAM) && MPAM2_EL2.EnMPAMSM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        X[t, 64] = MPAMSM_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
        ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAMSM_EL1;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMSM_EL1;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MSR MPAMSM_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b1010	0b0101	0b011

```
if !(IsFeatureImplemented(FEAT_MPAM) && IsFeatureImplemented(FEAT_SME)) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
        ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && IsFeatureImplemented(FEAT_MPAM) && MPAM2_EL2.EnMPAMSM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        MPAMSM_EL1 = X[t, 64];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
        ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMSM_EL1 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMSM_EL1 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.12.15 MPAMVPM0_EL2, MPAM Virtual PARTID Mapping Register 0

The MPAMVPM0_EL2 characteristics are:

Purpose

MPAMVPM0_EL2 provides mappings from virtual PARTIDs 0 - 3 to physical PARTIDs.

MPAMIDR_EL1.VPMR_MAX field gives the index of the highest implemented MPAMVPM<n>_EL2 register. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If MPAMIDR_EL1.VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, MPAMVPM0_EL2.

Virtual PARTID mapping is enabled by MPAMHCR_EL2.EL1_VPMEN for the PARTID in MPAM1_EL1 and by MPAMHCR_EL2.EL0_VPMEN for the PARTID in MPAM0_EL1.

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is valid only when the MPAMVPMV_EL2.VPM_V bit in bit position n is set to 1.

Configuration

This register has no effect if EL2 is not enabled in the current Security state.

This register is present only when (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p0 is implemented) and MPAMIDR_EL1.HAS_HCR == '1'. Otherwise, direct accesses to MPAMVPM0_EL2 are UNDEFINED.

Attributes

MPAMVPM0_EL2 is a 64-bit register.

Field descriptions

PhyPARTID3, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 3. PhyPARTID3 gives the mapping of virtual PARTID 3 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID2, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 2. PhyPARTID2 gives the mapping of virtual PARTID 2 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID1, bits [31:16]

Virtual PARTID Mapping Entry for virtual PARTID 1. PhyPARTID1 gives the mapping of virtual PARTID 1 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

PhyPARTID0, bits [15:0]

Virtual PARTID Mapping Entry for virtual PARTID 0. PhyPARTID0 gives the mapping of virtual PARTID 0 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPM0_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMVPM0_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b000

```
if !((IsFeatureImplemented(FEAT_MPAMv0p1) || IsFeatureImplemented(FEAT_MPAMv1p0)) && MPAMIDR_EL1.HAS_HCR
     ↪== '1') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x940];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
         ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAMVPM0_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMVPM0_EL2;
```

```
MSR MPAMVPM0_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b000

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if !((IsFeatureImplemented(FEAT_MPAMv0p1) || IsFeatureImplemented(FEAT_MPAMv1p0)) && MPAMIDR_EL1.HAS_HCR
     ↪== '1') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x940] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
         ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMVPM0_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMVPM0_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.12.16 MPAMVPM1_EL2, MPAM Virtual PARTID Mapping Register 1

The MPAMVPM1_EL2 characteristics are:

Purpose

MPAMVPM1_EL2 provides mappings from virtual PARTIDs 4 - 7 to physical PARTIDs.

MPAMIDR_EL1.VPMR_MAX field gives the index of the highest implemented MPAMVPM0_EL2 to MPAMVPM7_EL2 registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If MPAMIDR_EL1.VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, MPAMVPM0_EL2.

Virtual PARTID mapping is enabled by MPAMHCR_EL2.EL1_VPMEN for the PARTID in MPAM1_EL1 and by MPAMHCR_EL2.EL0_VPMEN for the PARTID in MPAM0_EL1.

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is valid only when the MPAMVPMV_EL2.VPM_V bit in bit position n is set to 1.

Configuration

This register has no effect if EL2 is not enabled in the current Security state.

This register is present only when ((IsFeatureImplemented(FEAT_MPAMv0p1) || IsFeatureImplemented(FEAT_MPAMv1p0)) && (MPAMIDR_EL1.HAS_HCR == '1')) && (UInt(MPAMIDR_EL1.VPMR_MAX) > 0). Otherwise, direct accesses to MPAMVPM1_EL2 are UNDEFINED.

Attributes

MPAMVPM1_EL2 is a 64-bit register.

Field descriptions

PhyPARTID7, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 7. PhyPARTID7 gives the mapping of virtual PARTID 7 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID6, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 6. PhyPARTID6 gives the mapping of virtual PARTID 6 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID5, bits [31:16]

Virtual PARTID Mapping Entry for virtual PARTID 5. PhyPARTID5 gives the mapping of virtual PARTID 5 to a physical PARTID.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID4, bits [15:0]

Virtual PARTID Mapping Entry for virtual PARTID 4. PhyPARTID4 gives the mapping of virtual PARTID 4 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPM1_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMVPM1_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b001

```
if !((IsFeatureImplemented(FEAT_MPAMv0p1) || IsFeatureImplemented(FEAT_MPAMv1p0)) && MPAMIDR_EL1.HAS_HCR
     ↪== '1' && UInt(MPAMIDR_EL1.VPMR_MAX) > 0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x948];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
         ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAMVPM1_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMVPM1_EL2;
```

```
MSR MPAMVPM1_EL2, <Xt>
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b001

```
if !((IsFeatureImplemented(FEAT_MPAMv0p1) || IsFeatureImplemented(FEAT_MPAMv1p0)) && MPAMIDR_EL1.HAS_HCR
     ↪== '1' && UInt(MPAMIDR_EL1.VPMR_MAX) > 0) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x948] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
         ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMVPM1_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMVPM1_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.12.17 MPAMVPM2_EL2, MPAM Virtual PARTID Mapping Register 2

The MPAMVPM2_EL2 characteristics are:

Purpose

MPAMVPM2_EL2 provides mappings from virtual PARTIDs 8 - 11 to physical PARTIDs.

MPAMIDR_EL1.VPMR_MAX field gives the index of the highest implemented MPAMVPM0_EL2 to MPAMVPM7_EL2 registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If MPAMIDR_EL1.VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, MPAMVPM0_EL2.

Virtual PARTID mapping is enabled by MPAMHCR_EL2.EL1_VPMEN for the PARTID in MPAM1_EL1 and by MPAMHCR_EL2.EL0_VPMEN for the PARTID in MPAM0_EL1.

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is valid only when the MPAMVPMV_EL2.VPM_V bit in bit position n is set to 1.

Configuration

This register has no effect if EL2 is not enabled in the current Security state.

This register is present only when ((IsFeatureImplemented(FEAT_MPAMv0p1) || IsFeatureImplemented(FEAT_MPAMv1p0)) && (MPAMIDR_EL1.HAS_HCR == '1')) && (UInt(MPAMIDR_EL1.VPMR_MAX) > 1). Otherwise, direct accesses to MPAMVPM2_EL2 are UNDEFINED.

Attributes

MPAMVPM2_EL2 is a 64-bit register.

Field descriptions

PhyPARTID11, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 11. PhyPARTID11 gives the mapping of virtual PARTID 11 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID10, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 10. PhyPARTID10 gives the mapping of virtual PARTID 10 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID9, bits [31:16]

Virtual PARTID Mapping Entry for virtual PARTID 9. PhyPARTID9 gives the mapping of virtual PARTID 9 to a physical PARTID.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID8, bits [15:0]

Virtual PARTID Mapping Entry for virtual PARTID 8. PhyPARTID8 gives the mapping of virtual PARTID 8 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPM2_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMVPM2_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b010

```
if !((IsFeatureImplemented(FEAT_MPAMv0p1) || IsFeatureImplemented(FEAT_MPAMv1p0)) && MPAMIDR_EL1.HAS_HCR
     ↪== '1' && UInt(MPAMIDR_EL1.VPMR_MAX) > 1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x950];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
         ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAMVPM2_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMVPM2_EL2;
```

```
MSR MPAMVPM2_EL2, <Xt>
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b010

```
if !((IsFeatureImplemented(FEAT_MPAMv0p1) || IsFeatureImplemented(FEAT_MPAMv1p0)) && MPAMIDR_EL1.HAS_HCR
     ↪== '1' && UInt(MPAMIDR_EL1.VPMR_MAX) > 1) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x950] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
         ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMVPM2_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMVPM2_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.12.18 MPAMVPM3_EL2, MPAM Virtual PARTID Mapping Register 3

The MPAMVPM3_EL2 characteristics are:

Purpose

MPAMVPM3_EL2 provides mappings from virtual PARTIDs 12 - 15 to physical PARTIDs.

MPAMIDR_EL1.VPMR_MAX field gives the index of the highest implemented MPAMVPM<n>_EL2 registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If MPAMIDR_EL1.VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, MPAMVPM0_EL2.

Virtual PARTID mapping is enabled by MPAMHCR_EL2.EL1_VPMEN for the PARTID in MPAM1_EL1 and by MPAMHCR_EL2.EL0_VPMEN for the PARTID in MPAM0_EL1.

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is valid only when the MPAMVPMV_EL2.VPM_V bit in bit position n is set to 1.

Configuration

This register has no effect if EL2 is not enabled in the current Security state.

This register is present only when ((IsFeatureImplemented(FEAT_MPAMv0p1) || IsFeatureImplemented(FEAT_MPAMv1p0)) && (MPAMIDR_EL1.HAS_HCR == '1')) && (UInt(MPAMIDR_EL1.VPMR_MAX) > 2). Otherwise, direct accesses to MPAMVPM3_EL2 are UNDEFINED.

Attributes

MPAMVPM3_EL2 is a 64-bit register.

Field descriptions

PhyPARTID15, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 15. PhyPARTID15 gives the mapping of virtual PARTID 15 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID14, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 14. PhyPARTID14 gives the mapping of virtual PARTID 14 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID13, bits [31:16]

Virtual PARTID Mapping Entry for virtual PARTID 13. PhyPARTID13 gives the mapping of virtual PARTID 13 to a physical PARTID.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID12, bits [15:0]

Virtual PARTID Mapping Entry for virtual PARTID 12. PhyPARTID12 gives the mapping of virtual PARTID 12 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPM3_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMVPM3_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b011

```
if !((IsFeatureImplemented(FEAT_MPAMv0p1) || IsFeatureImplemented(FEAT_MPAMv1p0)) && MPAMIDR_EL1.HAS_HCR
     ↪== '1' && UInt(MPAMIDR_EL1.VPMR_MAX) > 2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x958];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
         ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAMVPM3_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMVPM3_EL2;
```

```
MSR MPAMVPM3_EL2, <Xt>
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b011

```
if !((IsFeatureImplemented(FEAT_MPAMv0p1) || IsFeatureImplemented(FEAT_MPAMv1p0)) && MPAMIDR_EL1.HAS_HCR
     ↪== '1' && UInt(MPAMIDR_EL1.VPMR_MAX) > 2) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x958] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
         ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMVPM3_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMVPM3_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.12.19 MPAMVPM4_EL2, MPAM Virtual PARTID Mapping Register 4

The MPAMVPM4_EL2 characteristics are:

Purpose

MPAMVPM4_EL2 provides mappings from virtual PARTIDs 16 - 19 to physical PARTIDs.

MPAMIDR_EL1.VPMR_MAX field gives the index of the highest implemented MPAMVPM<n>_EL2 registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If MPAMIDR_EL1.VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, MPAMVPM0_EL2.

Virtual PARTID mapping is enabled by MPAMHCR_EL2.EL1_VPMEN for the PARTID in MPAM1_EL1 and by MPAMHCR_EL2.EL0_VPMEN for the PARTID in MPAM0_EL1.

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is valid only when the MPAMVPMV_EL2.VPM_V bit in bit position n is set to 1.

Configuration

This register has no effect if EL2 is not enabled in the current Security state.

This register is present only when ((IsFeatureImplemented(FEAT_MPAMv0p1) || IsFeatureImplemented(FEAT_MPAMv1p0)) && (MPAMIDR_EL1.HAS_HCR == '1')) && (UInt(MPAMIDR_EL1.VPMR_MAX) > 3). Otherwise, direct accesses to MPAMVPM4_EL2 are UNDEFINED.

Attributes

MPAMVPM4_EL2 is a 64-bit register.

Field descriptions

PhyPARTID19, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 19. PhyPARTID19 gives the mapping of virtual PARTID 19 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID18, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 18. PhyPARTID18 gives the mapping of virtual PARTID 18 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID17, bits [31:16]

Virtual PARTID Mapping Entry for virtual PARTID 17. PhyPARTID17 gives the mapping of virtual PARTID 17 to a physical PARTID.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID16, bits [15:0]

Virtual PARTID Mapping Entry for virtual PARTID 16. PhyPARTID16 gives the mapping of virtual PARTID 16 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPM4_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMVPM4_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b100

```
if !((IsFeatureImplemented(FEAT_MPAMv0p1) || IsFeatureImplemented(FEAT_MPAMv1p0)) && MPAMIDR_EL1.HAS_HCR
     ↪== '1' && UInt(MPAMIDR_EL1.VPMR_MAX) > 3) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x960];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
         ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAMVPM4_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMVPM4_EL2;
```

```
MSR MPAMVPM4_EL2, <Xt>
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b100

```
if !((IsFeatureImplemented(FEAT_MPAMv0p1) || IsFeatureImplemented(FEAT_MPAMv1p0)) && MPAMIDR_EL1.HAS_HCR
     ↪== '1' && UInt(MPAMIDR_EL1.VPMR_MAX) > 3) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x960] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
         ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMVPM4_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMVPM4_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.12.20 MPAMVPM5_EL2, MPAM Virtual PARTID Mapping Register 5

The MPAMVPM5_EL2 characteristics are:

Purpose

MPAMVPM5_EL2 provides mappings from virtual PARTIDs 20 - 23 to physical PARTIDs.

MPAMIDR_EL1.VPMR_MAX field gives the index of the highest implemented MPAMVPM<n>_EL2 registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If MPAMIDR_EL1.VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, MPAMVPM0_EL2.

Virtual PARTID mapping is enabled by MPAMHCR_EL2.EL1_VPMEN for the PARTID in MPAM1_EL1 and by MPAMHCR_EL2.EL0_VPMEN for the PARTID in MPAM0_EL1.

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is valid only when the MPAMVPMV_EL2.VPM_V bit in bit position n is set to 1.

Configuration

This register has no effect if EL2 is not enabled in the current Security state.

This register is present only when ((IsFeatureImplemented(FEAT_MPAMv0p1) || IsFeatureImplemented(FEAT_MPAMv1p0)) && (MPAMIDR_EL1.HAS_HCR == '1')) && (UInt(MPAMIDR_EL1.VPMR_MAX) > 4). Otherwise, direct accesses to MPAMVPM5_EL2 are UNDEFINED.

Attributes

MPAMVPM5_EL2 is a 64-bit register.

Field descriptions

PhyPARTID23, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 23. PhyPARTID23 gives the mapping of virtual PARTID 23 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID22, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 22. PhyPARTID22 gives the mapping of virtual PARTID 22 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID21, bits [31:16]

Virtual PARTID Mapping Entry for virtual PARTID 21. PhyPARTID21 gives the mapping of virtual PARTID 21 to a physical PARTID.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID20, bits [15:0]

Virtual PARTID Mapping Entry for virtual PARTID 20. PhyPARTID20 gives the mapping of virtual PARTID 20 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPM5_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMVPM5_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b101

```
if !((IsFeatureImplemented(FEAT_MPAMv0p1) || IsFeatureImplemented(FEAT_MPAMv1p0)) && MPAMIDR_EL1.HAS_HCR
     ↪== '1' && UInt(MPAMIDR_EL1.VPMR_MAX) > 4) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x968];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
         ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAMVPM5_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMVPM5_EL2;
```

```
MSR MPAMVPM5_EL2, <Xt>
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b101

```
if !((IsFeatureImplemented(FEAT_MPAMv0p1) || IsFeatureImplemented(FEAT_MPAMv1p0)) && MPAMIDR_EL1.HAS_HCR
     ↪== '1' && UInt(MPAMIDR_EL1.VPMR_MAX) > 4) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x968] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
         ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMVPM5_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMVPM5_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.12.21 MPAMVPM6_EL2, MPAM Virtual PARTID Mapping Register 6

The MPAMVPM6_EL2 characteristics are:

Purpose

MPAMVPM6_EL2 provides mappings from virtual PARTIDs 24 - 27 to physical PARTIDs.

MPAMIDR_EL1.VPMR_MAX field gives the index of the highest implemented MPAMVPM<n>_EL2 registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If MPAMIDR_EL1.VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, MPAMVPM0_EL2.

Virtual PARTID mapping is enabled by MPAMHCR_EL2.EL1_VPMEN for the PARTID in MPAM1_EL1 and by MPAMHCR_EL2.EL0_VPMEN for the PARTID in MPAM0_EL1.

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is valid only when the MPAMVPMV_EL2.VPM_V bit in bit position n is set to 1.

Configuration

This register has no effect if EL2 is not enabled in the current Security state.

This register is present only when ((IsFeatureImplemented(FEAT_MPAMv0p1) || IsFeatureImplemented(FEAT_MPAMv1p0)) && (MPAMIDR_EL1.HAS_HCR == '1')) && (UInt(MPAMIDR_EL1.VPMR_MAX) > 5). Otherwise, direct accesses to MPAMVPM6_EL2 are UNDEFINED.

Attributes

MPAMVPM6_EL2 is a 64-bit register.

Field descriptions

PhyPARTID27, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 27. PhyPARTID27 gives the mapping of virtual PARTID 27 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID26, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 26. PhyPARTID26 gives the mapping of virtual PARTID 26 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID25, bits [31:16]

Virtual PARTID Mapping Entry for virtual PARTID 25. PhyPARTID25 gives the mapping of virtual PARTID 25 to a physical PARTID.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID24, bits [15:0]

Virtual PARTID Mapping Entry for virtual PARTID 24. PhyPARTID24 gives the mapping of virtual PARTID 24 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPM6_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMVPM6_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b110

```
if !((IsFeatureImplemented(FEAT_MPAMv0p1) || IsFeatureImplemented(FEAT_MPAMv1p0)) && MPAMIDR_EL1.HAS_HCR
     ↪== '1' && UInt(MPAMIDR_EL1.VPMR_MAX) > 5) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x970];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
         ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAMVPM6_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMVPM6_EL2;
```

```
MSR MPAMVPM6_EL2, <Xt>
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b110

```
if !((IsFeatureImplemented(FEAT_MPAMv0p1) || IsFeatureImplemented(FEAT_MPAMv1p0)) && MPAMIDR_EL1.HAS_HCR
     ↪== '1' && UInt(MPAMIDR_EL1.VPMR_MAX) > 5) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x970] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
         ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMVPM6_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMVPM6_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.12.22 MPAMVPM7_EL2, MPAM Virtual PARTID Mapping Register 7

The MPAMVPM7_EL2 characteristics are:

Purpose

MPAMVPM7_EL2 provides mappings from virtual PARTIDs 28 - 31 to physical PARTIDs.

MPAMIDR_EL1.VPMR_MAX field gives the index of the highest implemented MPAMVPM<n>_EL2 registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If MPAMIDR_EL1.VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, MPAMVPM0_EL2.

Virtual PARTID mapping is enabled by MPAMHCR_EL2.EL1_VPMEN for the PARTID in MPAM1_EL1 and by MPAMHCR_EL2.EL0_VPMEN for MPAM0_EL1.

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is valid only when the MPAMVPMV_EL2.VPM_V bit in bit position n is set to 1.

Configuration

This register has no effect if EL2 is not enabled in the current Security state.

This register is present only when ((IsFeatureImplemented(FEAT_MPAMv0p1) || IsFeatureImplemented(FEAT_MPAMv1p0)) && (MPAMIDR_EL1.HAS_HCR == '1')) && (UInt(MPAMIDR_EL1.VPMR_MAX) == 7). Otherwise, direct accesses to MPAMVPM7_EL2 are UNDEFINED.

Attributes

MPAMVPM7_EL2 is a 64-bit register.

Field descriptions

PhyPARTID31, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 31. PhyPARTID31 gives the mapping of virtual PARTID 31 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID30, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 30. PhyPARTID30 gives the mapping of virtual PARTID 30 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID29, bits [31:16]

Virtual PARTID Mapping Entry for virtual PARTID 29. PhyPARTID29 gives the mapping of virtual PARTID 29 to a physical PARTID.

The reset behavior of this field is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID28, bits [15:0]

Virtual PARTID Mapping Entry for virtual PARTID 28. PhyPARTID28 gives the mapping of virtual PARTID 28 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPM7_EL2

Accesses to this register use the following encodings in the System register encoding space:

```
MRS <Xt>, MPAMVPM7_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b111

```
if !((IsFeatureImplemented(FEAT_MPAMv0p1) || IsFeatureImplemented(FEAT_MPAMv1p0)) && MPAMIDR_EL1.HAS_HCR
     ↪== '1' && UInt(MPAMIDR_EL1.VPMR_MAX) == 7) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x978];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
         ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAMVPM7_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMVPM7_EL2;
```

```
MSR MPAMVPM7_EL2, <Xt>
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0110	0b111

```
if !((IsFeatureImplemented(FEAT_MPAMv0p1) || IsFeatureImplemented(FEAT_MPAMv1p0)) && MPAMIDR_EL1.HAS_HCR
     ↪== '1' && UInt(MPAMIDR_EL1.VPMR_MAX) == 7) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x978] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
         ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMVPM7_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMVPM7_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

D24.12.23 MPAMVPMV_EL2, MPAM Virtual Partition Mapping Valid Register

The MPAMVPMV_EL2 characteristics are:

Purpose

Valid bits for virtual PARTID mapping entries. Each bit m corresponds to virtual PARTID mapping entry m in the MPAMVPM<n>_EL2 registers where n = m » 2.

Configuration

This register has no effect if EL2 is not enabled in the current Security state.

This register is present only when (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p0 is implemented) and MPAMIDR_EL1.HAS_HCR == '1'. Otherwise, direct accesses to MPAMVPMV_EL2 are UNDEFINED.

Attributes

MPAMVPMV_EL2 is a 64-bit register.

Field descriptions

Bits [63:32]

Reserved, RES0.

VPM_V<m> , bits [m], for m = 31 to 0

Contains valid bit for virtual PARTID mapping entry corresponding to virtual PARTID<m>.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPMV_EL2

Accesses to this register use the following encodings in the System register encoding space:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MRS <Xt>, MPAMVPMV_EL2
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0100	0b001

```
if !((IsFeatureImplemented(FEAT_MPAMv0p1) || IsFeatureImplemented(FEAT_MPAMv1p0)) && MPAMIDR_EL1.HAS_HCR
     ↪== '1') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        X[t, 64] = NVMem[0x938];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
         ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        X[t, 64] = MPAMVPMV_EL2;
elsif PSTATE.EL == EL3 then
    X[t, 64] = MPAMVPMV_EL2;
```

```
MSR MPAMVPMV_EL2, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b100	0b1010	0b0100	0b001

```
if !((IsFeatureImplemented(FEAT_MPAMv0p1) || IsFeatureImplemented(FEAT_MPAMv1p0)) && MPAMIDR_EL1.HAS_HCR
     ↪== '1') then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EffectiveHCR_EL2_NVx() IN {'1x1'} then
        NVMem[0x938] = X[t, 64];
    elsif EffectiveHCR_EL2_NVx() IN {'xx1'} then
        if HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
            if EL3SDDUndef() then
                UNDEFINED;
            else
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && EL3SDDUndefPriority() && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER ==
        ↪'1' then
        UNDEFINED;
    elsif HaveEL(EL3) && IsFeatureImplemented(FEAT_MPAM) && MPAM3_EL3.TRAPLOWER == '1' then
        if EL3SDDUndef() then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMVPMV_EL2 = X[t, 64];
elsif PSTATE.EL == EL3 then
    MPAMVPMV_EL2 = X[t, 64];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Part E
The AArch32 Application Level Architecture

Chapter E1
The AArch32 Application Level Programmers' Model

This chapter gives an Application level description of the programmers' model for software executing in AArch32 state. This means it describes execution in EL0 when EL0 is using AArch32. It contains the following sections:

- About the Application level programmers' model.
- The Application level programmers' model in AArch32 state.
- Advanced SIMD and floating-point instructions.
- About the AArch32 System register interface.
- Exceptions.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E1.1 About the Application level programmers' model

This chapter contains the programmers' model information required for the development of applications that execute in AArch32 state.

The information in this chapter is distinct from the system information required to service and support application execution under an operating system, or higher level of system software. However, some knowledge of that system information is needed to put the Application level programmers' model into context.

Depending on the implementation, the architecture supports multiple levels of execution privilege. These privilege levels are indicated by different *Exception levels* that number upwards from EL0, where EL0 corresponds to the lowest privilege level and is often described as *unprivileged*. The Application level programmers' model is the programmers' model for software executing at EL0. For more information, see Arm architectural concepts.

System software determines the Exception level, and therefore the level of privilege, at which application software runs. When an operating system supports execution at both EL1 and EL0, an application usually runs unprivileged. This has the following effects:

- It means that the operating system can allocate system resources to an application in a unique or shared manner.
- It provides a degree of protection from other processes, and so helps protect the operating system from malfunctioning software.

This chapter indicates where some System level understanding is helpful, and if appropriate it gives a reference to the System level description.

Application level software is generally unaware of its Security state, and of any virtualization. For more information, see The Armv8-A security model and The effect of implementing EL2 on the Exception model.

Note

- When an implementation includes EL3, application and operating system software normally executes in Non-secure state.
- Older documentation, describing implementations or architecture versions that support only two privilege levels, often refers to execution at EL1 as *privileged* execution.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E1.2 The Application level programmers' model in AArch32 state

The following sections give more information about the application level programmers' model in AArch32 state:

- Instruction sets, arithmetic operations, and register files.
- Core data types and arithmetic in AArch32 state.
- The general-purpose registers, and the PC, in AArch32 state.
- Process state, PSTATE.
- Jazelle support.

E1.2.1 Instruction sets, arithmetic operations, and register files

The T32 and A32 instruction sets both provide a wide range of integer arithmetic and logical operations, that operate on a register file of sixteen 32-bit registers, that are comprised of the AArch32 general-purpose registers and the PC. As described in The general-purpose registers, and the PC, in AArch32 state, these registers include the registers SP (R13) and LR (R14), which have specialized uses. Core data types and arithmetic in AArch32 state gives more information about these operations.

In addition, an implementation that implements the T32 and A32 instruction sets includes both:

- Scalar floating-point instructions.
- The Advanced SIMD vector instructions.

Floating-point and vector instructions operate on a separate common register file, described in The SIMD and floating-point register file. Advanced SIMD and floating-point instructions gives more information about these instructions.

E1.2.2 Core data types and arithmetic in AArch32 state

When executing in AArch32 state, a PE supports the following data types in memory:

Byte 8 bits.

Halfword 16 bits.

Word 32 bits.

Doubleword 64 bits.

PE registers are 32 bits in size. The instruction sets provide instructions that use the following data types for data held in registers:

- 32-bit pointers.
- Unsigned or signed 32-bit integers.
- Unsigned 16-bit or 8-bit integers, held in zero-extended form.
- Signed 16-bit or 8-bit integers, held in sign-extended form.
- Two 16-bit integers packed into a register.
- Four 8-bit integers packed into a register.
- Unsigned or signed 64-bit integers held in two registers.

Load and store operations can transfer bytes, halfwords, or words to and from memory. Loads of bytes or halfwords zero-extend or sign-extend the data as it is loaded, as specified in the appropriate load instruction.

The instruction sets include load and store operations that transfer two or more words to and from memory. Software can load and store doublewords using these instructions.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Note

For information about the atomicity of memory accesses, see Atomicity in the Arm architecture.

When any of the data types is described as *unsigned*, the N-bit data value represents a non-negative integer in the range 0 to 2^N-1, using normal binary format.

When any of these types is described as *signed*, the N-bit data value represents an integer in the range $-2^{(N-1)}$ to $+2^{(N-1)}$-1, using two's complement format.

The instructions that operate on packed halfwords or bytes include some multiply instructions that use only one of two halfwords, and SIMD instructions that perform parallel addition or subtraction on all of the halfwords or bytes.

Note

These SIMD instructions operate on values held in the general-purpose registers, and must not be confused with the Advanced SIMD instructions that operate on a separate register file that provides registers of up to 128 bits.

Direct instruction support for 64-bit integers is limited, and most 64-bit operations require sequences of two or more instructions to synthesize them.

E1.2.2.1 Integer arithmetic

The instruction set provides a wide range of operations on the values in registers, including bitwise logical operations, shifts, additions, subtractions, multiplications, and divisions. The pseudocode described in Arm Pseudocode Definition defines these operations, usually in one of three ways:

- By direct use of the pseudocode operators and built-in functions defined in Operators.
- By use of pseudocode helper functions defined in the main text.
- By a sequence of the form:
  1. Use of the `SInt()`, `UInt()`, and `Int()` built-in functions defined in Converting bitstrings to integers to convert the bitstring contents of the instruction operands to the unbounded integers that they represent as two's complement or unsigned integers.
  2. Use of mathematical operators, built-in functions and helper functions on those unbounded integers to calculate other such integers.
  3. Use of either the bitstring extraction operator defined in Bitstring concatenation and slicing or of the saturation helper functions described in Pseudocode description of saturation to convert an unbounded integer result into a bitstring result that can be written to a register.

E1.2.2.1.1 Shift and rotate operations

The following types of shift and rotate operations are used in instructions:

Logical Shift Left

The `LSL()` pseudocode function moves each bit of a bitstring left by a specified number of bits. Zeros are shifted in at the right end of the bitstring. Bits that are shifted off the left end of the bitstring are discarded, except that the last such bit can be produced as a carry output.

Logical Shift Right

The `LSR()` pseudocode function moves each bit of a bitstring right by a specified number of bits. Zeros are shifted in at the left end of the bitstring. Bits that are shifted off the right end of the bitstring are discarded, except that the last such bit can be produced as a carry output.

Arithmetic Shift Right

The `ASR()` pseudocode function moves each bit of a bitstring right by a specified number of bits. Copies of the leftmost bit are shifted in at the left end of the bitstring. Bits that are shifted off the right end of the bitstring are discarded, except that the last such bit can be produced as a carry output.

Rotate Right The `ROR()` pseudocode function moves each bit of a bitstring right by a specified number of bits. Each bit that is shifted off the right end of the bitstring is re-introduced at the left end. The last bit shifted off the right end of the bitstring can be produced as a carry output.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Rotate Right with Extend

The `RRX()` pseudocode function moves each bit of a bitstring right by one bit. A carry input is shifted in at the left end of the bitstring. The bit shifted off the right end of the bitstring can be produced as a carry output.

E1.2.2.1.2 Pseudocode description of addition and subtraction

In pseudocode, addition and subtraction can be performed on any combination of unbounded integers and bitstrings, provided that if they are performed on two bitstrings, the bitstrings must be identical in length. The result is another unbounded integer if both operands are unbounded integers, and a bitstring of the same length as the bitstring operand or operands otherwise. For the definition of these operations, see Addition and subtraction.

The main addition and subtraction instructions can produce status information about both unsigned carry and signed overflow conditions. When necessary, multi-word additions and subtractions can be synthesized from this status information. In pseudocode, the `AddWithCarry()` function provides an addition with a carry input and a set of output Condition flags including carry output and overflow:

An important property of the `AddWithCarry()` function is that if:

```
(result, nzcv) = AddWithCarry(x, NOT(y), carry_in)
```

Then:

- If `carry_in == '1'`, then `result == x-y` with:
  - `nzcv<0> == '1'` if signed overflow occurred during the subtraction.
  - `nzcv<1> == '1'` if unsigned borrow did not occur during the subtraction, that is, if $x \geq y$.
- If `carry_in == '0'`, then `result == x-y-1` with:
  - `nzcv<0> == '1'` if signed overflow occurred during the subtraction.
  - `nzcv<1> == '1'` if unsigned borrow did not occur during the subtraction, that is, if $x \geq y$.

Taken together, this means that the `carry_in` and `nzcv<1>` output in `AddWithCarry()` calls can act as NOT borrow flags for subtractions as well as carry flags for additions.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E1.2.2.1.3 Pseudocode description of saturation

Some instructions perform *saturating arithmetic*, that is, if the result of the arithmetic overflows the destination signed or unsigned N-bit integer range, the result produced is the largest or smallest value in that range, rather than wrapping around modulo 2^N. This is supported in pseudocode by:

- The `SignedSatQ()` and `UnsignedSatQ()` functions when an operation requires, in addition to the saturated result, a Boolean argument that indicates whether saturation occurred.
- The `SignedSat()` and `UnsignedSat()` functions when only the saturated result is required.

`SatQ(i, N, unsigned)` returns either `UnsignedSatQ(i, N)` or `SignedSatQ(i, N)` depending on the value of its third argument, and `Sat(i, N, unsigned)` returns either `UnsignedSat(i, N)` or `SignedSat(i, N)` depending on the value of its third argument.

E1.2.3 The general-purpose registers, and the PC, in AArch32 state

In the AArch32 Application level view, a PE has:

- Fifteen general-purpose 32-bit registers, R0 to R14, of which R13 and R14 have alternative names reflecting how they are, or can be, used:
  - R13 is usually identified as SP.
  - R14 is usually identified as LR.
- The PC (*Program Counter*), which can be described as R15.

The specialized uses of the SP (R13), LR (R14), and PC (R15) are:

SP, the stack pointer

The PE uses SP as a pointer to the active stack.

In the T32 instruction set, some instructions cannot access SP. Instructions that can access SP can use SP as a general-purpose register.

The A32 instruction set provides more general access to SP, and it can be used as a general-purpose register.

Note

Using SP for any purpose other than as a stack pointer might break the requirements of operating systems, debuggers, and other software systems, causing them to malfunction.

Software can refer to SP as R13.

LR, the link register

The link register can be used to hold return link information, and some cases described in this manual require this use of the LR. When software does not require the LR for linking, it can use it for other purposes. Software can refer to LR as R14.

PC, the Program Counter

- When executing an A32 instruction, PC reads as the address of the current instruction plus 8.
- When executing a T32 instruction, PC reads as the address of the current instruction plus 4.
- Writing an address to PC causes a branch to that address.

Most T32 instructions cannot access PC.

The A32 instruction set provides more general access to the PC, and many A32 instructions can use the PC as a general-purpose register. However, Arm deprecates the use of PC for any purpose other than as the Program Counter. See Writing to the PC for more information.

Software can refer to PC as R15.

See AArch32 general-purpose registers, the PC, and the Special-purpose registers for the system level view of these registers.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Note

In general, Arm strongly recommends using the names SP, LR, and PC instead of R13, R14, and R15. However, sometimes it is simpler to use the R13-R15 names when referring to a group of registers. For example, it is simpler to refer to *registers R8 to R15*, rather than to *registers R8 to R12, the SP, LR, and PC*. These two descriptions of the group of registers have the same meaning.

E1.2.3.1 Writing to the PC

In the T32 and A32 instruction sets, many data-processing instructions can write to the PC. Writes to the PC are handled as follows:

- In T32 state, the following 16-bit T32 instruction encodings branch to the value written to the PC:
  - Encoding T2 of ADD, ADDS (register).
  - Encoding T1 of MOV, MOVS (register).

  The value written to the PC is forced to be halfword-aligned by ignoring its least significant bit, treating that bit as being 0.
- The `B`, `BL`, `CBNZ`, `CBZ`, `TBB`, and `TBH` instructions remain in the same instruction set state and branch to the value written to the PC.

  The definition of each of these instructions ensures that the value written to the PC is correctly aligned for the current instruction set state.
- The `BLX` (immediate) instruction switches between T32 and A32 states and branches to the value written to the PC. Its definition ensures that the value written to the PC is correctly aligned for the new instruction set state.
- The following instructions write a value to the PC, treating that value as an interworking address to branch to, with low-order bits that determine the new instruction set state:
  - `BLX` (register), `BX`, and `BXJ`.
  - `LDR` instructions with `<Rt>` equal to the PC.
  - `POP` and all forms of `LDM` except `LDM` (exception return), when the register list includes the PC.
  - In A32 state only, `ADC`, `ADD`, `ADR`, `AND`, `ASR` (immediate), `BIC`, `EOR`, `LSL` (immediate), `LSR` (immediate), `MOV`, `MVN`, `ORR`, `ROR` (immediate), `RRX`, `RSB`, `RSC`, `SBC`, and `SUB` instructions with `<Rd>` equal to the PC and without flag-setting specified.

  For details of how an interworking address specifies the new instruction set state and instruction address, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.

  Note

  The register-shifted register instructions, which are available only in the A32 instruction set and are summarized in Data-processing register (register shift), are CONSTRAINED UNPREDICTABLE if they attempt to write to the PC, see Using R15 by instruction.

- Some instructions are treated as exception return instructions, and write both the PC and the CPSR. For more information, including which instructions are exception return instructions, see Exception return to an Exception level using AArch32.
- Some instructions cause an exception, and the exception handler address is written to the PC as part of the exception entry.

E1.2.3.2 Pseudocode description of operations on the AArch32 general-purpose registers and the PC

In pseudocode, the uses of the `R[]` function, with an index parameter `n`, are:

- Reading or writing R0-R12, SP, and LR, using n = 0-12, 13, and 14 respectively.
- Reading the PC, using n = 15.

Pseudocode description of general-purpose register and PC operations describes accesses to these registers.

Descriptions of A32 store instructions that store the PC value use the `PCStoreValue()` pseudocode function to specify the PC value stored by the instruction.

Writing an address to the PC causes either a simple branch to that address or an *interworking* branch that also selects the instruction set to execute after the branch. A simple branch is performed by the `BranchWritePC()` function.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

An interworking branch is performed by the `BXWritePC()` function.

The `LoadWritePC()` and `ALUWritePC()` functions are used for two cases where the behavior was systematically modified between architecture versions.

E1.2.4 Process state, PSTATE

Process state or PSTATE is an abstraction of process state information. All of the instruction sets provide instructions that operate on elements of PSTATE.

Note

In this chapter, references to PSTATE link to the more appropriate of:

- The Application-level view of PSTATE given in this section.
- The System-level description in Process state, PSTATE.

The following PSTATE information is accessible at EL0:

The condition flags

Flag-setting instructions set these. They are:

N Negative Condition flag. If the result of the instruction is regarded as a two's complement signed integer, the PE sets this to:

- 1 if the result is negative.
- 0 if the result is positive or zero.

Z Zero Condition flag. Set to:

- 1 if the result of the instruction is zero.
- 0 otherwise.

A result of zero often indicates an equal result from a comparison.

C Carry Condition flag. Set to:

- 1 if the instruction results in a carry condition, for example an unsigned overflow that is the result of an addition.
- 0 otherwise.

V Overflow Condition flag. Set to:

- 1 if the instruction results in an overflow condition, for example a signed overflow that is the result of an addition.
- 0 otherwise.

Conditional instructions test the N, Z, C, and V Condition flags, combining them with the *Condition code* for the instruction, to determine whether the instruction must be executed. In this way, execution of the instruction is conditional on the result of a previous operation. For more information about conditional execution, see Conditional execution.

The overflow or saturation flag

Q Some instructions can set this. For those instructions that can, the PE:

- Sets it to 1 if the instruction indicates overflow or saturation.
- Leaves it unchanged otherwise.

For more information, see Pseudocode description of saturation.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The greater than or equal flags

GE[3:0] The instructions described in Parallel addition and subtraction instructions update these to indicate the results from individual bytes or halfwords of the operation. These flags can control a later `SEL` instruction. For more information, see SEL.

PSTATE also contains *PE state controls*. There is no direct access to these from application level instructions, but they can be changed by side-effects of application level instructions. They are:

Instruction set state

J, T The current instruction set state, as shown in Table E1-1. The J bit is RES0, see the Note in this section.

Table E1-1 PSTATE.{J, T} encoding

J	T	Instruction set state
0	0	A32
0	1	T32

A32 The PE is executing the A32 instruction set, summarized in A32 Instruction Set Encoding.

T32 The PE is executing the T32 instruction set, summarized in T32 Instruction Set Encoding.

Note

Encoding with J==1 before Armv8, Jazelle, and T32EE states

In previous versions of the Arm architecture, the encoding {1, 0} selected Jazelle state, and encoding {1, 1} selected T32EE state. From the introduction of Armv8, the architecture does not support either of these states, and these are encodings for unimplemented instruction set states, see Unimplemented instruction sets.

AArch32 state requires a Trivial Jazelle implementation, see Trivial implementation of the Jazelle extension.

The IT block state

IT[7:0] The If-Then controls for the T32 `IT` instruction, which applies to the *IT block* of instructions that immediately follow the `IT` instruction. See IT for a description of the `IT` instruction and its associated `IT` block.

For more information about the use of PSTATE.IT, see Use of PSTATE.IT.

Endianness mapping

E For data accesses, controls the endianness:

0 Little-endian.

1 Big-endian.

If an implementation does not provide:

- Big-endian support for data accesses, this bit is RES0.
- Little-endian support for data accesses, this bit is RES1.

Instruction fetches are always little-endian, and ignore PSTATE.E.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Timing control bits

DIT *Data Independent Timing* (DIT) bit. For more information, see About the DIT bit.

This bit is implemented only when FEAT_DIT is implemented.

On a reset to AArch32 state, this bit is set to 0.

E1.2.4.1 Accessing PSTATE fields at EL0

The following sections describe which PSTATE fields can be directly accessed at EL0, and how they can be accessed:

- The Application Program Status Register, APSR.
- The SETEND instruction.

E1.2.4.1.1 The Application Program Status Register, APSR

At EL0, some PSTATE fields can be accessed using the Special-purpose *Application Program Status Register* (APSR). The APSR can be directly read using the MRS instruction, and directly written using the MSR (register) and MSR (immediate) instructions.

The APSR bit assignments are:

N, Z, C, V, bits [31:28]

The PSTATE Condition flags.

Q, bit [27] The PSTATE overflow or saturation flag.

Bits[26:24] Reserved, RES0. Software can use MSR instructions that write the top byte of the APSR without using a read-modify-write sequence. If it does this, it must write zeros to bits[26:24].

Bits[23:20, 15:0]

Reserved bits that are allocated to system features, or are available for future expansion. Unprivileged execution ignores writes to fields that are accessible only at EL1 or higher. However, application level software that writes to the APSR must treat reserved bits as *Do-Not-Modify* (DNM) bits. For more information about the reserved bits, see The Current Program Status Register, CPSR.

These bits are UNKNOWN on a Read, and it is permitted that, on a read of APSR:

- Bit[22] returns the value of PSTATE.PAN.
- Bit[9] returns the value of PSTATE.E.
- Bits[8:6] return the value of PSTATE.{A,I,F}, the mask bits.
- Bits[4:0] return the value of PSTATE.M[4:0]. Bit[4] is RES1 indicating that the PE is in AArch32 state.

Note

This is an exception to the general rule that an UNKNOWN field must not return information that cannot be obtained, at the current Privilege level, by an architected mechanism.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

GE[3:0], bits [19:16]

The PSTATE greater than or equal flags.

The other PSTATE fields cannot be accessed by using the APSR.

The system level alias for the APSR is the CPSR. The CPSR is a superset of the APSR. See The Current Program Status Register, CPSR.

Writes to the PSTATE fields have side-effects on various aspects of PE operation. All of these side-effects, except side-effects on memory accesses associated with fetching instructions, are synchronous to the APSR write. This means they are guaranteed:

- Not to be visible to earlier instructions in the Execution stream.
- To be visible to later instructions in the Execution stream.

E1.2.4.1.2 The SETEND instruction

The T32 and A32 instruction sets both include an instruction to manipulate PSTATE.E:

`SETEND BE` Sets PSTATE.E to 1, for big-endian operation.

`SETEND LE` Sets PSTATE.E to 0, for little-endian operation.

The `SETEND` instruction is unconditional. For more information, see SETEND. Arm deprecates use of the `SETEND` instruction.

E1.2.4.2 Use of PSTATE.IT

PSTATE.IT provides the If-Then controls for the T32 `IT` instruction, which applies to the *IT block* of instructions that immediately follow the `IT` instruction.

PSTATE.IT divides into two subfields:

IT[7:5] Holds the *base condition* for the current IT block. The base condition is the top three bits of the Condition code specified by the `<firstcond>` field of the `IT` instruction.

IT[4:0] Encodes:

- Implicitly, the size of the IT block. This is the number of instructions that are to be conditionally executed. The size of the block is indicated by the position of the least significant 1 in this field, as shown in Table E1-2.
- For each instruction in the IT block, the least significant bit of the Condition code. This is encoded in the IT block entries that Table E1-2 shows as N*x*.

Note

Changing the least significant bit of a Condition code from 0 to 1 has the effect of inverting the Condition code.

Both subfields are all zeros when no IT block is active.

When an `IT` instruction is executed, PSTATE.IT is set according to the `<firstcond>` field of the instruction and the *Then* and *Else* (T and E) parameters in the instruction, see IT. This means that, on executing an IT instruction, the initial state of PSTATE.IT depends on the number of instructions in the IT block, as Table E1-2 shows:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table E1-2 Initial state of PSTATE.IT on executing an IT instruction

Number of instructions in IT block	PSTATE.IT bits[a]						Notes
	[7:5]	[4]	[3]	[2]	[1]	[0]	
4	cond_base	N1	N2	N3	N4	1	-
3	cond_base	N1	N2	N3	1	0	-
2	cond_base	N1	N2	1	0	0	-
1	cond_base	N1	1	0	0	0	-
Not executing an `IT` instruction	000	0	0	0	0	0	No IT block is active

a. Combinations of the IT bits not shown in this table are reserved.

In Table E1-2, N1 refers to the first instruction in the IT block, and N2, N3, and N4 refer to the second, third, and fourth instructions in the IT block if they are present,

When permitted, an instruction in an IT block is conditional, see Conditional instructions and Conditional execution. The Condition code used is the current value of IT[7:4]. When an instruction in an IT block completes its execution normally, PSTATE.IT[4:0] is left-shifted by one bit, so that PSTATE[4] always relates to the next instruction to be executed.

Table E1-3 shows how PSTATE.IT during the execution of an `IT` instruction with four instructions in the IT block.

Table E1-3 Updates to PSTATE.IT when executing an IT instruction with a four-instruction IT block

IT block instruction being executed	PSTATE.IT bits[a]						Notes
	[7:5]	[4]	[3]	[2]	[1]	[0]	
First	cond_base	N1	N2	N3	N4	1	-
Second	cond_base	N2	N3	N4	1	0	-
Third	cond_base	N3	N4	1	0	0	-
Fourth	cond_base	N4	1	0	0	0	-
Not executing an `IT` instruction	000	0	0	0	0	0	No IT block is active

a. Combinations of the IT bits not shown in this table are reserved.

A few instructions, for example `BKPT`, cannot be conditional and therefore are always executed ignoring the current value of PSTATE.IT.

For details of what happens if an instruction in an IT block takes an exception, see Overview of exception entry.

An instruction that might complete its normal execution by branching is only permitted in an IT block as the last instruction in the block. This means that normal execution of the instruction always results in PSTATE.IT advancing to execution where no IT block is active.

Implementations can provide a set of ITD control fields, SCTLR.ITD, SCTLR_EL1.ITD, and HSCTLR.ITD, to disable use of IT for some instructions, making them UNDEFINED. When an implementation includes ITD control fields, Changes to an ITD control by an instruction in an IT block describes the permitted CONSTRAINED UNPREDICTABLE behaviors if an instruction in an IT block changes the value of an ITD control to disable the use of the `IT` instruction.

On a branch or an exception return, if PSTATE.IT is set to a value that is not consistent with the instruction stream being branched to or returned to, then instruction execution is CONSTRAINED UNPREDICTABLE.

PSTATE.IT affects instruction execution only in T32 state. In A32 state, PSTATE.IT must be `0b00000000`, otherwise the behavior is CONSTRAINED UNPREDICTABLE.

For more information, see CONSTRAINED UNPREDICTABLE behavior associated with IT instructions and PSTATE.IT.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E1.2.4.2.1 Changes to an ITD control by an instruction in an IT block

In an implementation that includes SCTLR.ITD, SCTLR_EL1.ITD, and HSCTLR.ITD controls, if an instruction in an IT block changes an ITD control so that the `IT` instruction using the IT block would be disabled, then one of the following behaviors applies:

- The change to the ITD field, once synchronized, has no effect on the execution of instructions in the current IT block, but applies only to any subsequent execution of an `IT` instruction to which the control applies.
- Synchronizing the change to the ITD field guarantees that all bits of PSTATE.IT are cleared to 0.

In addition, after the change to the ITD field has been synchronized, any remaining instructions in the IT block that would be made UNDEFINED by the new value of ITD are either:

- Executed normally.
- Treated as UNDEFINED.

The choice between the options described in this section is determined by the implementation, and any choice can vary between different changes to an ITD control by an instruction in an IT block.

E1.2.4.3 Pseudocode description of PSTATE PE state fields

The pseudocode function `CurrentInstrSet()` returns the current instruction set. The pseudocode function `SelectInstrSet()` selects a new instruction set.

PSTATE.IT advances after normal execution of an IT block instruction. This is described by the `AArch32.ITAdvance()` pseudocode function.

The pseudocode function `InITBlock()` tests whether the current instruction is in an IT block. The pseudocode function `LastInITBlock()` tests whether the current instruction is the last instruction in an IT block.

The `BigEndian()` pseudocode function tests whether big-endian data memory accesses are currently selected.

E1.2.5 About the DIT bit

A *data-independent-time sequence of code* is the sequence of instructions executed from the first instruction executed after a change of CPSR.DIT from 0 to 1 through the last instruction executed before a change of CPSR.DIT from 1 to 0, inclusive. Instructions speculatively executed between the change of CPSR.DIT from 0 to 1 and the change of CPSR.DIT from 1 to 0 are considered part of the data-independent-time sequence of code.

A *data-independent-time resource* is any of:

- a general-purpose register other than the zero register (ZR)
- Stack Pointer register
- SIMD&FP register
- the NZCV flags

Note

The collection of NZCV is considered a single data-indepedent-time resource.

A *data-independent-time instruction* is an instruction that is required to honor the data-independent timing behaviors described in this section.

A *data-independent-time* value is any of:

- Any given value (V) read from memory by the Explicit Memory Read Effect of a data-independent-time instruction (I1) that is executed within a data-independent-time sequence of code (C), where V is not consumed by any other instruction (I2) for which any of the following apply:
  - I2 is not a data-independent-time instruction
  - I2 is not executed in C
  - V is used to form the address or Logical Address Tag of a Memory Effect generated by I2

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- Any given value (V) written to a data-independent-time resource by a data-independent-time instruction (I1) that is executed within a data-independent-time sequence of code (C), where V is not consumed by any other instruction (I2) for which any of the following apply:
  - I2 is not a data-independent-time instruction
  - I2 is not executed in C
  - V is used to form the address or Logical Address Tag of a Memory Effect generated by I2
- Any given value (V) read from a data-independent-time resource by a data-independent-time instruction (I1) that is executed within a data-independent-time sequence of code (C) and none of the following apply:
  - V was produced by another instruction (I0) and any of the following apply:
    - I0 is not a data-independent-time instruction
    - I0 is not executed in C
  - V is used to form the address or Logical Address Tag of a Memory Effect produced by I1.
  - V is not consumed by any other instruction (I2) for which any of the following apply:
    - I2 is not a data-independent-time instruction
    - I2 is not executed in C
    - V is used to form the address or Logical Address Tag of a Memory Effect generated by I2
- Any given value (V) written to memory by the Explicit Memory Write Effect of a data-independent-time instruction (I1) that is executed within a data-independent-time sequence of code (C) and none of the following apply:
  - V was produced by another instruction (I0) and any of the following apply:
    - I0 is not a data-independent-time instruction
    - I0 is not executed in C
  - V is used to form the address or Logical Address Tag of a Memory Effect produced by I1.
  - V was sourced from the zero register.
  - V is not consumed by any other instruction (I2) for which any of the following apply:
    - I2 is not a data-independent-time instruction
    - I2 is not executed in C
    - V is used to form the address or Logical Address Tag of a Memory Effect generated by I2
- Any given value (V) speculatively read from memory at a Location (L) as the result of executing a data-independent-time sequence of code, where L is not explicitly read by the data-independent-time sequence of code. Examples of this include:
  - Reading additional bytes from memory while reading the bytes required by an Explicit Memory Read Effect
  - Hardware-generated prefetches
  - Prefetch memory instructions

The execution time of a given data-independent-time sequence of code must be independent of all data-independent-time values.

The execution time of a data-independent-time instruction (I1) executed within a data-independent-time sequence of code must be independent of all data-independent-time values consumed or produced by I1.

The time to respond to an asynchronous exception taken from within a data-independent-time sequence of code must be independent of all data-independent-time values.

The execution of a data-independent-time sequence of code (C1) must not cause an irreversible change in microarchitectural state of the PE that would permit a sequence of code (C2) that is executed after C1 to determine any data-independent-time value accessed during the execution of C1. For the purposes of this requirement, a data-independent-time value written by an Explicit Memory Write Effect that is held in caches is not considered an irreversible change in microarchitectural state.

The execution of a data-independent-time sequence of code (C1) must not alter the state of PMU counters in a way that allows a sequence of code (C2) that is executed after C1 to determine any data-independent-time value accessed during C1.

Note

In cases where an instruction is architecturally required to only read or write a subset of a data-independent-time resource, the entire data-independent-time resource is considered to have been read or written for the sake of these requirements.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Note

The architecture does not guarantee that the execution time of a data-independent-time sequence of code is independent of data-independent-time values if those values are read-from or written-to a peripheral.

Note

The classification of a value being a data-independent-time value due to having been read from a memory location (L) by an Explicit Memory Read Effect or written to L by an Explicit Memory Write Effect is based on the particular Effect generated by a data-independent-time instruction executed from within a data-independent-time sequence of code. The value held in L is not itself considered a data-independent-time value. As a result, it is possible for the execution time of a data-independent-time sequence of code to vary based on the value held in L if L is accessed via any of the following:

- Outside the data-independent-time sequence of code; either architecturally or speculatively
- Within a data-independent-time sequence of code but not as a data-independent-time value

Note

When the value of CPSR.DIT is 0, the architecture makes no statement about the timing properties of any instructions.

A corresponding DIT bit is added to PSTATE in AArch64 state, and to CPSR in AArch32 state.

When an exception is taken from AArch32 state to AArch32 state, CPSR.DIT is copied to SPSR.DIT.

When an exception is taken from AArch32 state to AArch64 state, CPSR.DIT is copied to SPSR_ELx.DIT.

When an exception returns to AArch32 state from AArch32 state, SPSR.DIT is copied to CPSR.DIT.

When an exception returns to AArch32 state from AArch64 state, SPSR_ELx.DIT is copied to CPSR.DIT.

CPSR.DIT bit can be written using an `MSR` instruction at any Exception Level in AArch32 state, and read using an `MRS` instruction at any Exception Level.

E1.2.6 Jazelle support

The architecture requires AArch32 state to include a trivial implementation of the Jazelle Extension, as described in Trivial implementation of the Jazelle extension.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E1.3 Advanced SIMD and floating-point instructions

In general, the architecture requires implementation of Advanced SIMD and floating-point instructions in the T32 and A32 instruction sets, but see Implications of not including Advanced SIMD and floating-point support.

The Advanced SIMD instructions perform packed *Single Instruction Multiple Data* (SIMD) operations, on either integer or single-precision floating-point operands. The floating-point instructions perform single-precision or double-precision scalar floating-point operations. When FEAT_FP16 is implemented, half-precision floating-point can also be used for data processing.

These instructions permit *floating-point exceptions*, such as Overflow or Divide by Zero, to be handled without trapping. When handled in this way, a floating-point exception causes a cumulative status register bit to be set to 1 and a default result to be produced by the operation. The architecture also optionally supports the trapping of floating-point exceptions. For more information about floating-point exceptions, see Floating-point exceptions and exception traps.

The Advanced SIMD and floating-point instructions also provide the following conversion functions:

- Between half-precision floating-point and single-precision floating point, in both directions.
- From double-precision, floating-point to single-precision floating point or integer.
- When FEAT_AA32BF16 is implemented, between single-precision floating-point and BFloat16 floating-point.

Some Advanced SIMD instructions support polynomial arithmetic over {0, 1}, as described in Polynomial arithmetic over {0, 1}.

For system level information about the Advanced SIMD and floating-point implementation, see Advanced SIMD and floating-point support.

The following sections give more information about the Advanced SIMD and floating-point instructions:

- Floating-point standards, and terminology.
- The SIMD and floating-point register file.
- Data types supported by the Advanced SIMD implementation.
- Advanced SIMD and floating-point System registers.
- Floating-point data types and arithmetic.
- Flushing denormalized numbers to zero.
- Floating-point exceptions and exception traps.
- Controls of Advanced SIMD operation that do not apply to floating-point operation.
- Implications of not including Advanced SIMD and floating-point support.
- Pseudocode description of floating-point operations.

E1.3.1 The SIMD and floating-point register file

The Advanced SIMD and floating-point instructions use the same register file, which comprises 32 registers. This is distinct from the register file that holds the general-purpose registers and the PC.

The Advanced SIMD and floating-point views of the register file are different. The following sections describe these different views. Figure E1-1 shows the views of the register file, and the way the word, doubleword, and quadword registers overlap.

E1.3.1.1 Advanced SIMD views of the register file

Advanced SIMD can view this register file as:

- Sixteen 128-bit quadword registers, `Q0-Q15`.
- Thirty-two 64-bit doubleword registers, `D0-D31`.

These views can be used simultaneously. For example, a program might hold 64-bit vectors in `D0` and `D1` and a 128-bit vector in `Q1`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E1.3.1.2 Floating-point views of the register file

The Advanced SIMD and floating-point register file consists of thirty-two doubleword registers, which can be viewed as:

- Thirty-two 64-bit doubleword registers, `D0-D31`. This view is also available to Advanced SIMD instructions.
- Thirty-two 32-bit single word registers, `S0-S31`. Only half of the set is accessible in this view.

Note

In AArch32 state, half-precision floating point values are always represented using the bottom 16 bits of a single word register, `S0-S31`. When a half-precision value is written to a single word register, the top 16 bits of that register are set to 0.

The two views can be used simultaneously.

E1.3.1.3 SIMD and Floating-point register file mapping onto registers

Figure E1-1 shows the different views of the SIMD and floating-point register file, and the relationship between them.

Figure E1-1 SIMD and floating-point register file, AArch32 operation

The mapping between the registers is as follows:

- `S<2n>` maps to the least significant half of `D<n>`.
- `S<2n+1>` maps to the most significant half of `D<n>`.
- `D<2n>` maps to the least significant half of `Q<n>`.
- `D<2n+1>` maps to the most significant half of `Q<n>`.

For example, software can access the least significant half of the elements of a vector in `Q6` by referring to `D12`, and the most significant half of the elements by referring to `D13`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E1.3.1.4 Pseudocode description of the SIMD and Floating-point register file

The functions `_Dclone`, `S[]`, and `D[]` provide the `S0-S31`, `D0-D31`, and `Q0-Q15` views of the Advanced SIMD and floating-point registers:

The `Din[]` function returns a doubleword register from the `_Dclone[]` copy of the SIMD and Floating-point register file, and the `Qin[]` function returns a quadword register from that register file.

Note

The `CheckAdvSIMDEnabled()` function copies the `D[]` register file to `_Dclone[]`, see Pseudocode description of enabling SIMD and floating-point functionality.

E1.3.2 Data types supported by the Advanced SIMD implementation

Advanced SIMD instructions can operate on integer and floating-point data, and the implementation defines a set of data types that support the required data formats. Advanced SIMD vector formats in AArch32 state describes these formats.

E1.3.2.1 Advanced SIMD vectors

In an implementation that includes support for Advanced SIMD operation, a register can hold one or more packed elements, all of the same size and type. The combination of a register and a data type describes a vector of elements. The vector is considered to be an array of elements of the data type specified in the instruction. The number of elements in the vector is implied by the size of the data elements and the size of the register.

Vector indices are in the range 0 to (number of elements - 1). An index of 0 refers to the least significant end of the vector. In Advanced SIMD vector formats in AArch32 state, Figure A1-3 shows the Advanced SIMD vector formats.

E1.3.2.1.1 Pseudocode description of Advanced SIMD vectors

The pseudocode function `Elem[]` accesses the element of a specified index and size in a vector.

E1.3.3 Advanced SIMD and floating-point System registers

The Advanced SIMD and floating-point instructions have a shared register space for System registers. The only register in this space that is accessible at the Application level is the FPSCR.

Writes to the FPSCR can have side-effects on various aspects of PE operation. All of these side-effects are synchronous to the FPSCR write. This means they are guaranteed not to be visible to earlier instructions in the Execution stream, and they are guaranteed to be visible to later instructions in the Execution stream.

See Advanced SIMD and floating-point System registers for the system level view of the registers.

These registers can be described as the *SIMD and floating-point System registers*.

E1.3.4 Floating-point data types and arithmetic

The T32 and A32 floating-point instructions support single-precision (32-bit) and double-precision (64-bit) data types and arithmetic as defined by the IEEE 754 floating-point standard. They also support the half-precision (16-bit) floating-point data type for data storage, by supporting conversions between single-precision and half-precision data types. When FEAT_FP16 is implemented, it also supports the half-precision floating-point data type for data processing operations. When FEAT_AA32BF16 is implemented, it also supports the BFloat16 floating-point storage format.

Arm standard floating-point arithmetic means IEEE 754 floating-point arithmetic with the restrictions described in Floating-point support, including supporting only the input and output values described in Arm standard floating-point input and output values.

The AArch32 Advanced SIMD instructions support single-precision and, when FEAT_FP16 is implemented, half-precision Arm standard floating-point arithmetic.

The following sections describe the Advanced SIMD and floating-point formats:

- Half-precision floating-point formats.
- Single-precision floating-point format.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- Double-precision floating-point format.
- BFloat16 floating-point format.

The following sections describe features of Advanced SIMD and floating-point processing:

- Flushing denormalized numbers to zero.
- NaN handling and the Default NaN.

E1.3.5 Flushing denormalized numbers to zero

Calculations involving denormalized numbers and Underflow exceptions can reduce the performance of floating-point processing. For many algorithms, replacing the denormalized operands and Intermediate results with zeros can recover this performance, without significantly affecting the accuracy of the final result. Arm floating-point implementations allow denormalized numbers to be flushed to zero to permit this optimization. If a number satisfies the condition `0 < Abs(result) < MinNorm`, it is treated as a denormalized number.

`MinNorm` is defined as follows:

- For half-precision numbers, `MinNorm` is 2^{-14}.
- For single-precision and BFloat16 numbers, `MinNorm` is 2^{-126}.
- For double-precision numbers, `MinNorm` is 2^{-1022}.

Flushing denormals to zero is incompatible with the IEEE 754 standard, and must not be used when IEEE 754 compatibility is a requirement. Enabling flushing of denormals to zero must be done with care. Although it can improve performance on some algorithms, there are significant limitations on its use. These are application-dependent:

- On many algorithms, it has no noticeable effect, because the algorithm does not usually process denormalized numbers.
- On other algorithms, it can cause exceptions to occur and can seriously reduce the accuracy of the results of the algorithm.

E1.3.5.1 Flushing denormalized inputs to zero

If flushing denormalized inputs to zero is enabled for an instruction and a data type, and an input to that instruction is a denormalized number of that data type, the input operand is flushed to zero, and its sign bit is not changed.

If a floating-point operation has an input denormalized number that is flushed to zero, for all purposes within the instruction other than calculating Input Denormal floating-point exceptions, all inputs that are denormalized numbers are treated as though they were zero with the same sign as the input.

For Advanced SIMD and floating-point instructions, other than `FABS` and `FNEG`, that process half-precision inputs, flushing denormalized inputs to zero can be controlled as follows:

- If FPSCR.FZ16 is 0, denormalized half-precision inputs are not flushed to zero.
- If FPSCR.FZ16 is 1, flushing denormalized inputs to zero occurs as follows:
  - If an instruction does not convert a half-precision input to a higher precision output, all input denormalized numbers are flushed to zero.
  - If an instruction converts a half-precision input to a higher precision output, input denormalized numbers are not flushed to zero.

For Advanced SIMD and scalar floating-point instructions, other than `FABS` and `FNEG`, that process single-precision, or double-precision inputs, flushing denormalized inputs to zero can be controlled as follows:

- If FPSCR.FZ is 0, flushing denormalized inputs to zero occurs as follows:
  - For Advanced SIMD floating-point instructions, all single-precision and double-precision inputs that are denormalized numbers are flushed to zero.
  - For scalar floating-point instructions, single-precision and double-precision inputs that are denormalized numbers are not flushed to zero.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- If FPSCR.FZ is 1, for all T32 and A32 instructions, single-precision and double-precision inputs that are denormalized numbers are flushed to zero.

If FEAT_AA32BF16 is implemented, for Advanced SIMD and scalar floating-point instructions, other than `FABS` and `FNEG`, that process BFloat16 inputs, flushing denormalized inputs to zero is treated as follows:

- Instructions that convert from single-precision floating-point values to BFloat16 format flush denormalized inputs to zero.
- For any value of FPSCR.FZ, VDOT (vector), VDOT (by element), and VMMLA instructions flush all BFloat16 inputs that are denormalized numbers to zero.

E1.3.5.2 Flushing to zero of denormalized numbers as Intermediate results of some BFloat16 instructions

BFloat16 arithmetic instructions VDOT (by element), VDOT (vector), and VMMLA, convert BFloat16 input values to IEEE single-precision format, and calculate N-way dot-products, accumulating the products in single-precision accumulators.

If FEAT_AA32BF16 is implemented, for Advanced SIMD and floating-point instructions, if a BFloat16 arithmetic instruction processes an Intermediate result that is a single-precision denormalized number, the Intermediate result is unconditionally flushed to zero.

E1.3.5.3 Flushing denormalized outputs to zero

If flushing denormalized outputs to zero is enabled for an instruction and a data type, and an output from that instruction is a denormalized number of that data type, the output operand is flushed to zero, and its sign bit is not changed.

If a floating-point operation has an output denormalized number that is flushed to zero, for all purposes within the instruction other than calculating floating-point exceptions, all outputs that are denormalized numbers are treated as though they were zero with the same sign as the output.

For Advanced SIMD and floating-point instructions, other than `FABS` and `FNEG`, that generate half-precision outputs, flushing denormalized outputs to zero can be controlled as follows:

- If FPSCR.FZ16 is 0, denormalized half-precision outputs are not flushed to zero.
- If FPSCR.FZ16 is 1, flushing denormalized outputs to zero occurs as follows:
  - — If the instruction does not convert a half-precision input to a higher precision output, all output denormalized numbers are flushed to zero.
  - — If the instruction converts a half-precision input to a higher precision output, output denormalized numbers are not flushed to zero.

For Advanced SIMD and scalar floating-point instructions, other than `FABS` and `FNEG`, that process single-precision, or double-precision inputs, flushing denormalized outputs to zero can be controlled as follows:

- If FPSCR.FZ is 0, flushing denormalized outputs to zero occurs as follows:
  - — For Advanced SIMD floating-point instructions, all single-precision and double-precision outputs that are denormalized numbers are flushed to zero.
  - — For scalar floating-point instructions, single-precision and double-precision outputs that are denormalized numbers are not flushed to zero.
- If FPSCR.FZ is 1, for all T32 and A32 instructions, single-precision and double-precision outputs that are denormalized numbers are flushed to zero.

If FEAT_AA32BF16 is implemented, for Advanced SIMD and scalar floating-point instructions, other than `FABS` and `FNEG`, that generate BFloat16 outputs, flushing denormalized outputs to zero can be controlled as follows:

- BFloat16 arithmetic instructions flush denormalized outputs to zero.
- If FPSCR.FZ is 1, instructions that convert from single-precision floating-point values to BFloat16 format flush denormalized outputs to zero.
- VDOT (vector), VDOT (by element), and VMMLA instructions flush all BFloat16 outputs that are denormalized numbers to zero regardless of the value of FPSCR.FZ.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E1.3.6 NaN handling and the Default NaN

The IEEE 754 standard defines a NaN as a number with all exponent bits set to 1 and a nonzero number in the mantissa, and specifies that the sign bit of a NaN has no significance.

The Arm architecture additionally defines:

- A BFloat16 NaN, that follows the encoding in BFloat16 floating-point format.
- A *Default NaN*, compliant with the IEEE 754 standard, that follows the encoding in Table E1-4.

For a Quiet NaN output derived from a signaling NaN operand, the most significant fraction bit is set to 1.

A PE is forbidden to generate a NaN whose value is strongly correlated to the values of non-NaN inputs as a speculative result of a floating-point calculation not involving NaN inputs.

E1.3.6.1 The Default NaN

The Default NaN is encoded as described in Table E1-4.

Table E1-4 Default NaN encoding

	Half-precision, IEEE format	Single-precision	Double-precision	BFloat16
Sign bit	0	0	0	0
Exponent	`0x1F`	`0xFF`	`0x7FF`	`0xFF`
Fraction	Bit[9] == 1, bits[8:0] == 0	Bit[22] == 1, bits[21:0] == 0	Bit[51] == 1, bits[50:0] == 0	Bit[6] == 1, bits[5:0] == 0

If FPSCR.DN is 1, for Advanced SIMD and floating-point instructions other than `FABS`, `FMAX*`, `FMIN*` and `FNEG`, if any input to a floating-point operation performed by the instruction is a NaN, the output of the floating-point operation is the Default NaN.

For `FABS`, `FNEG`, `FMAX*`, and `FMIN*`, Default NaN behavior is explained in the instruction description.

If FPSCR.DN is 0, for floating-point processing the Default NaN is not used for NaN propagation.

If VDOT (vector), VDOT (by element), and VMMLA instructions generate a NaN, the NaN is the default NaN. regardless of the setting of FPSCR.DN.

If a floating-point instruction performs a floating-point operation, and that instruction generates an untrapped Invalid Operation floating-point exception for a reason other than one of the inputs being a signaling NaN, the output is the Default NaN.

E1.3.6.2 NaN handling

The IEE 754 standard does not specify which input NaN is used as the output NaN. Therefore, where the Arm architecture specifies which input NaN to use, this is an addition to the requirements in the IEEE 754 standard.

Depending on the operation, the exact value of a derived Quiet NaN output might have both a different sign and a different number of fraction bits from its source. See instruction descriptions for details.

E1.3.6.3 NaN propagation

If an output NaN is derived from one of the operands, how the input NaN propagates to the output depends on the instruction and the number of operands.

If an output NaN is derived from an input NaN and if the size of the output format is the same as the input format, then all of the following apply:

- If the input NaN is a Quiet NaN, the output NaN is the same as the input NaN.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- If the input NaN is a signaling NaN, the output NaN is derived as follows:
  - If the handling of a signaling NaN by the instruction generates an Invalid Operation exception, the output NaN is the quieted version of the input NaN.
  - If the handling of a signaling NaN by the instruction does not generate an Invalid Operation exception, the output NaN is the same as the input NaN. This case applies for FABS, FNEG, and FTSSEL instructions.

If an output NaN is derived from an input NaN and if the size of the output format is larger than the input format, all of the following apply:

- If the input NaN is a Quiet NaN, the output NaN is the same as the input NaN except that the mantissa is zero-extended in the low-order bit to fit the output format, and the exponent field is set to all ones.
- If the input NaN is a signaling NaN, the output NaN is the quieted version of the input NaN, except that the mantissa is zero-extended in the low-order bits and the exponent field is set to all ones.

If an output NaN is derived from an input NaN and if the size of the output format is smaller than the input format, all of the following apply:

- If the input NaN is a Quiet NaN, the output NaN is the same as the input NaN except that the mantissa is truncated in the lower-order bits to fit the output format, and the exponent field is set to all ones.
- If the input NaN is a signaling NaN, the output NaN is the quieted version of the input NaN except that the mantissa is truncated in the lower-order bits to fit the output format, and the exponent field is set to all ones.

For the following descriptions, when an operand is described as *first* this relates to the left-to-right ordering of the arguments of the pseudocode function that describes the operation.

If FPSCR.DN is 0, for Advanced SIMD, floating-point, or BFloat16 instructions that perform a floating-point operation, other than `FABS`, `FNEG`, `FMAX*`, and `FMIN*`, NaN outputs that derive from NaN inputs are derived as follows:

- If all of the following apply, an instruction outputs a quiet NaN derived from the first signaling NaN operand:
  - At least one operand is a signaling NaN.
  - The instruction is not trapped.
- If all of the following apply, an instruction outputs a Quiet NaN derived from the first NaN operand:
  - At least one operand is a NaN, but none of the operands is a signaling NaN.
  - The instruction is not trapped.

If an output NaN is derived from an input NaN, the pseudocode functions `FPAbs()`, and `FPNeg()` can change the sign of the NaN,

E1.3.7 Rounding

The rounding mode specifies how the exact result of a floating-point operation is rounded to a value in the destination format.

The rounding mode is either determined by the rounding mode control field FPSCR.RMode or by the instruction.

The rounding mode control field FPSCR.RMode can select the following rounding modes:

- Round to Nearest (RN) mode.
- Round towards Plus Infinity (RP) mode.
- Round towards Minus Infinity (RM) mode.
- Round towards Zero (RZ) mode.

The following two additional rounding modes are not selected by FPSCR.RMode, but are used by some instructions:

- Round to Odd mode.
- Round to Nearest with ties to away mode.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E1.3.7.1 Round to Nearest mode

Round to Nearest rounding mode rounds the exact result of a floating-point operation to a value that is representable in the destination format as follows:

- If the value before rounding has an absolute value that is too large to represent in the output format, the rounded value is an Infinity. The sign of the rounded value is the same as the sign of the value before rounding.
- If the value before rounding has an absolute value that is not too large to represent in the output format, the result is calculated as follows:
  - If the two nearest floating-point numbers bracketing the value before rounding are equally near, the result is the number with an even least significant digit.
  - If the two nearest floating-point numbers bracketing the value before rounding are not equally near, the result is the floating-point number nearest to the value before rounding.

Advanced SIMD arithmetic always uses the Round to Nearest setting, regardless of the value of the RMode bits.

E1.3.7.2 Round towards Plus Infinity mode

Round towards Plus Infinity rounding mode rounds the exact result of a floating-point operation to a value that is representable in the destination format. The result is the floating-point number in the output format that is closest to and not less than the value before rounding. The result can be plus infinity.

E1.3.7.3 Round towards Minus Infinity mode

Round towards Minus Infinity rounding mode rounds the exact result of a floating-point operation to a value that is representable in the destination format. The result is the number in the output format that is closest to and not greater than the value before rounding. The result can be minus infinity.

E1.3.7.4 Round towards Zero mode

Round towards Zero rounding mode rounds the exact result of a floating-point operation to a value that is representable in the destination format. The result is the floating-point number in the output format that is closest to and not greater in absolute value than the value before rounding.

E1.3.7.5 Round to Nearest with Ties to Away

Round to Nearest with Ties to Away rounding mode is used by the VCVTA (Advanced SIMD), VCVTA (floating-point), VRINTA (Advanced SIMD), and VRINTA (floating-point) instructions.

Round to Nearest with Ties to Away rounding mode rounds the exact result of a floating-point operation to a value that is representable in the destination format as follows:

- If the value before rounding has an absolute value that is too large to represent in the output format, the rounded value is an Infinity, the sign of the rounded value is the same as the sign of the value before rounding.
- If the value before rounding has an absolute value that is not too large to represent in the output format, the result is calculated as follows:
  - If the two nearest floating-point numbers bracketing the value before rounding are equally near, the result is the larger number.
  - If the two nearest floating-point numbers bracketing the value before rounding are not equally near, the result is the floating-point number nearest to the value before rounding.

E1.3.7.6 Round to Odd mode

Round to Odd mode is not defined by IEEE 754.

For BFloat16 instructions, if an intermediate format has at least two more bits of precision than the result format, Round to Odd mode is used and operates as follows:

- If the rounded value is inexact, the least significant bit of the fraction is set to 1.
- If the value is too large to represent in the single-precision format, the rounded value is a single-precision Infinity, the sign of the rounded value is the same as the sign of the value before rounding.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E1.3.8 Floating-point exceptions and exception traps

Execution of a floating-point instruction, or execution of an Advanced SIMD instruction that performs floating-point operations, can generate an exceptional condition, called a *floating-point exception*.

Note

An Advanced SIMD instruction that operates on floating-point values can perform multiple floating-point operations. Therefore, this section describes the handling of a floating-point exception on an *operation*, rather than on an *instruction*.

The architecture does not support asynchronous reporting of floating-point exceptions.

For each of the following floating-point exceptions, it is IMPLEMENTATION DEFINED whether an implementation includes synchronous exception generation:

- Input Denormal.
- Inexact.
- Underflow.
- Overflow.
- Divide by Zero.
- Invalid Operation.

If an implementation does not support synchronous exception generation from a floating-point exception, then that synchronous exception is never generated, and all statements about synchronous exception generation from that floating-point exception do not apply to the implementation.

If an implementation supports synchronous exception generation for a floating-point exception, then the registers that are presented to the exception handler are consistent with the state of the PE immediately before the instruction that caused the exception.

On return from a synchronous floating-point exception, software might not restore the cumulative exception flags.

Trapped floating-point exceptions are taken to the following levels:

- If a trapped floating-point exception occurs at EL0, the exception level it is taken to is as follows:
  - If EL2 is using AArch32 and HCR.TGE is 1, the exception is taken to EL2.
  - If EL2 is using AArch64 and HCR_EL2.TGE is 1, the exception is taken to EL2
  - Otherwise, the exception is taken to EL1
- If a trapped floating-point exception occurs at EL1, it is taken to EL1.
- If a trapped floating-point exception occurs at EL2, it is taken to EL2.
- If a trapped floating-point exception occurs at EL3, it is taken to EL3.

If a trapped floating-point exception is taken to an Exception level that is using AArch64, then it is reported in the ELR_ELx for the target Exception level, as described in Exception entry.

If the exception is taken to an Exception level that is using AArch32, then it is taken as an Undefined Instruction exception, see Undefined Instruction exception. The FPEXC identifies the floating-point exceptions that occurred since the corresponding status bits in that register were last set to 0.

E1.3.8.1 Input Denormal exceptions

The cumulative floating-point exception bit FPSCR.IDC, and the trap enable bit FPSCR.IDE both relate to Input Denormal exceptions.

If a single-precision or double-precision floating-point input is flushed to zero, an Input Denormal exception is generated.

If a half-precision floating-point value is flushed to zero, an Input Denormal exception is not generated.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E1.3.8.2 Inexact exceptions

The cumulative floating-point exception bit FPSCR.IXC, and the trap enable bit FPSCR.IXE both relate to Inexact exceptions.

If a denormalized output is flushed to zero, an Inexact exception is not generated.

If a result is not flushed to zero, and the result does not equal the result computed with unbounded exponent range and unbounded precision, then an Inexact exception is generated.

E1.3.8.3 Underflow exceptions

The cumulative floating-point exception bit FPSR.UFC, and the trap enable bit FPSCR.UFE both relate to Underflow exceptions.

For the purpose of Underflow floating-point exception generation, a denormalized number is detected before rounding is applied.

If the result of a floating-point operation is a denormalized number that is not flushed to zero, then the Underflow exception is generated as follows:

- If the result is exact, an Underflow floating-point exception is generated only if FPSCR.UFE is 1.
- If the result is inexact, an Underflow floating-point exception is always generated.

If the result of a floating-point operation is a denormalized number that is flushed to zero, then the Underflow floating-point exception is generated. The Underflow exception is not trapped regardless of the value of FPSCR.UFE.

E1.3.8.4 Overflow exceptions

The cumulative floating-point exception bit FPSCR.OFC, and the trap enable bit FPSCR.OFE both relate to Overflow exceptions.

If the output of an instruction rounded with an unbounded exponent is greater than the maximum normalized number for the output precision, an Overflow exception is generated.

If an untrapped Overflow exception is generated, the result is determined by the rounding mode and the sign of the result before rounding as follows:

- Round to Nearest carries all overflows to infinity with the sign of the result before rounding.
- Round towards Plus Infinity carries negative overflows to the most negative finite number of the output precision, and carries positive overflows to plus infinity.
- Round towards Minus Infinity carries positive overflows to the largest finite number of the output precision, and carries negative overflows to minus infinity.
- Round towards Zero carries all overflows to the output precision's largest finite number with the sign of the result before rounding.

E1.3.8.5 Divide by Zero exceptions

The cumulative floating-point exception bit FPSCR.DZC, and the trap enable bit FPSCR.DZE both relate to Divide by Zero exceptions.

If a floating-point operation divides a finite nonzero number by zero, a Divide by Zero exception is generated.

For the purpose of Divide by Zero exception generation, testing for zero occurs after flushing of denormalized numbers to zero.

A denormalized dividend that is flushed to zero is treated as zero and prevents Divide by Zero from occurring.

If the dividend is a finite nonzero, normalized number, and the divisor is a denormalized number, the divisor is treated as zero and causes Divide by Zero to occur.

For the reciprocal and reciprocal square root estimate functions, the dividend is assumed to be +1.0. This means that a zero or denormalized operand to these functions causes generation of a Divide by Zero floating-point exception.

If a floating-point operation divides a finite nonzero number by zero, and the Divide by Zero exception is untrapped, the result is a correctly signed infinity.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E1.3.8.6 Invalid Operation exceptions

The cumulative floating-point exception bit FPSCR.IOC, and the trap enable bit FPSCR.IOE both relate to Invalid Operation exceptions.

For any floating-point instruction that performs a floating-point operation, if any of the following apply, the instruction generates an Invalid Operation exception:

- At least one operand is a signaling NaN, and the instruction is not `FABS` or `FNEG`.
- Subtracting two equal-magnitude infinities that are of the same sign.
- Multiplying a zero by an infinity.
- Dividing a zero by a zero.
- Dividing an infinity by an infinity.
- Taking the square root of an operand that is less than zero.

For the purpose of Invalid Operation Exception generation, testing for zero occurs after flushing of denormalized numbers to zero. So a denormalized input that is flushed to zero is treated as zero.

If the input is one of: a Quiet NaN, an infinity, or a number that overflows the values that can be represented in the output format, and if another exception is not generated to signal the condition, then a conversion from floating-point to either integer or fixed-point format, generates an Invalid Operation exception.

For the signaling compare instructions FCMPE and FCCMPE, if either of the source operands is any type of NaN, the instruction generates an Invalid Operation floating-point exception.

E1.3.8.7 Floating-point exception traps

For Advanced SIMD instructions, and for floating-point instructions when floating-point exception trapping is not supported, these are non-trapping exceptions and the data-processing instructions do not generate any trapped exceptions.

For floating-point instructions when floating-point exception trapping is supported:

- The floating-point exceptions can be trapped, by setting trap enable bits in the FPSCR, see Floating-point exceptions and exception traps, and:
  - When a trap is not enabled the corresponding floating-point exception updates the corresponding FPSCR cumulative bit, but does not generate an exception.
  - When a trap is enabled the corresponding floating-point exception does not update the FPSCR, but generates an exception. In this case, bits in the FPEXC indicate which floating-point exceptions have occurred.
- The definition of the Underflow floating-point exception is different in the trapped and cumulative exception cases. In the trapped case, the definition is:
  - The trapped Underflow floating-point exception occurs if the absolute value of the result of an operation, produced before rounding, is less than the minimum positive normalized number for the destination precision, regardless of whether the rounded result is inexact.
- As with cumulative exceptions, higher priority trapped exceptions can prevent lower priority exceptions from occurring, as described in Combinations of floating-point exceptions.
- For Invalid Operation floating-point exceptions, for details of which Quiet NaN is produced as the default result, see NaN handling and the Default NaN.
- For Overflow floating-point exceptions, the sign bit of the default result is determined normally for the overflowing operation.
- For Divide by Zero floating-point exceptions, the sign bit of the default result is determined normally for a division. This means it is the exclusive-OR of the sign bits of the two operands.

Table E1-5 shows the results of untrapped floating-point exceptions. That table uses the following abbreviations:

MaxNorm The maximum normalized number of the destination precision.

RM Round towards Minus Infinity mode, as defined in the IEEE 754 standard.

RN Round to Nearest mode, as defined in the IEEE 754 standard.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

RP Round towards Plus Infinity mode, as defined in the IEEE 754 standard.

RZ Round towards Zero mode, as defined in the IEEE 754 standard.

For more information about the IEEE 754 descriptions of the rounding modes, see Floating-point standards, and terminology.

Table E1-5 Results of untrapped floating-point exceptions

Exception type	Default result for positive sign		Default result for negative sign	
IOC, Invalid Operation	Quiet NaN		Quiet NaN	
DZC, Divide by Zero	+infinity		-infinity	
OFC, Overflow	RN, RP: RM, RZ:	+infinity +MaxNorm	RN, RM: RP, RZ:	-infinity -MaxNorm
UFC, Underflow	Normal rounded result		Normal rounded result	
IXC, Inexact	Normal rounded result		Normal rounded result	
IDC, Input Denormal	Normal rounded result		Normal rounded result	

E1.3.8.8 Combinations of floating-point exceptions

Many pseudocode functions perform *floating-point operations*, including `FixedToFP()`, `FPAdd()`, `FPCompare()`, `FPCompareEQ()`, `FPCompareGE()`, `FPCompareGT()`, `FPDiv()`, `FPMax()`, `FPMin()`, `FPMul()`, `FPMulAdd()`, `FPRecipEstimate()`, `FPRecipStep()`, `FPRSqrtEstimate()`, `FPRSqrtStep()`, `FPSqrt()`, `FPSub()`, and `FPToFixed()`. All of these operations can generate floating-point exceptions.

Note

`FPAbs()` and `FPNeg()` are not classified as *floating-point operations* because:

- They cannot generate floating-point exceptions.
- The floating-point operation behavior described in the following sections does not apply to them:
  - Flushing denormalized numbers to zero.
  - NaN handling and the Default NaN.

More than one exception might occur on the same operation. The architecture permits only the following combinations of floating-point exceptions:

- Overflow with Inexact.
- Underflow with Inexact.
- Input Denormal with one or more of the other floating-point exception types.

The priority order of these floating-point exceptions is that the Inexact exception is treated as lowest priority, and the Input Denormal exception is treated as highest priority.

When none of the floating-point exceptions caused by an operation is trapped, any floating-point exception that occurs causes the associated cumulative bit in the FPSCR to be set.

When one or more floating-point exceptions caused by an operation is trapped, the behavior of the instruction depends on the priority of the exceptions:

- If the higher priority floating-point exception is trapped, its trap handler is called. It is IMPLEMENTATION DEFINED whether any information about the lower priority floating-point exception is provided.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Note

Information about the lower priority floating-point exception might be provided in:

— The FPEXC, if the exception generated by the trap is taken to an Exception level that is using AArch32.

— The ESR_ELx.ISS field, if the exception generated by the trap is taken to an Exception level that is using AArch64.

However, information might be provided in another IMPLEMENTATION DEFINED way, for example using an IMPLEMENTATION DEFINED register.

Apart from this, the lower priority floating-point exception is ignored in this case.

- If the higher priority floating-point exception is untrapped, its cumulative bit is set to 1 and its default result is evaluated. Then the lower priority floating-point exception is handled normally, using this default result.

Some floating-point instructions specify more than one floating-point operation, as indicated by the pseudocode descriptions of the instruction. In such cases, a floating-point exception on one operation is treated as higher priority than a floating-point exception on another operation if the occurrence of the second floating-point exception depends on the result of the first operation. Otherwise, it is CONSTRAINED UNPREDICTABLE which floating-point exception is treated as higher priority.

For example, a `VMLA.F32` instruction specifies a floating-point multiplication followed by a floating-point addition. The addition can generate Overflow, Underflow and Inexact floating-point exceptions, all of which depend on both operands to the addition and so are treated as lower priority than any floating-point exception on the multiplication. The same applies to Invalid Operation floating-point exceptions on the addition caused by adding opposite-signed infinities. The addition can also generate an Input Denormal floating-point exception, caused by the addend being a denormalized number while in Flush-to-zero mode. It is CONSTRAINED UNPREDICTABLE which of an Input Denormal floating-point exception on the addition and a floating-point exception on the multiplication is treated as higher priority, because the occurrence of the Input Denormal floating-point exception does not depend on the result of the multiplication. The same applies to an Invalid Operation floating-point exception on the addition caused by the addend being a signaling NaN.

Note

The `VFMA` instruction performs a vector addition and a vector multiplication as a single operation. The `VFMS` instruction performs a vector subtraction and a vector multiplication as a single operation.

E1.3.9 Controls of Advanced SIMD operation that do not apply to floating-point operation

Earlier architectures permitted implementation of either, both, or neither of the Advanced SIMD and floating-point additions to the base instruction set, and provided some controls that applied to the Advanced SIMD functionality but not to the floating-point functionality. From the introduction of Armv8, Advanced SIMD functionality cannot be separated from floating-point functionality, but in AArch32 state these controls function as they did in earlier architectures. This means they apply only to the following instructions and instruction encodings:

- All instructions with encodings defined in:
  — Advanced SIMD data-processing, for the T32 instruction set.
  — Advanced SIMD data-processing, for the A32 instruction set.
- All instructions with encodings defined in:
  — Advanced SIMD element or structure load/store, for the T32 instruction set.
  — Advanced SIMD element or structure load/store, for the A32 instruction set.
- The form of the `VDUP` instruction described in VDUP (general-purpose register).
- The byte and halfword forms of the VMOV instructions described in each of:
  — VMOV (general-purpose register to scalar).
  — VMOV (scalar to general-purpose register).

The controls of this functionality are:

- The CPACR.ASEDIS field.
- The HCPTR.TASE field.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

In an implementation that supports Advanced SIMD functionality, support for each of these controls is optional:

- If the CPACR.ASEDIS control is not supported then the CPACR.ASEDIS field is RAZ/WI. This is equivalent to the control permitting the execution of Advanced SIMD instructions at EL1 and EL0.
- If the HCPTR.TASE control is not supported then the HCPTR.TASE field is RAZ/WI. This means the HCPTR does not provide a control that can trap Non-secure execution of Advanced SIMD instructions to Hyp mode.

E1.3.10 Implications of not including Advanced SIMD and floating-point support

In general, the architecture requires the inclusion of the Advanced SIMD and floating-point instructions in all instruction sets. Exceptionally, for implementation targeting specialized markets, Arm might produce or license an Armv8-A implementation that does not provide any support for Advanced SIMD and floating-point instructions. In such an implementation, in AArch32 state:

- Each of the CPACR.{cp10, cp11} fields is RES0.
- The CPACR.ASEDIS bit is RES0.
- Each of the HCPTR.{TASE, TCP10, TCP11} fields is RES1.
- Each of the NSACR.{NSASEDIS, cp10, cp11} fields is RES0.
- The FPEXC register is UNDEFINED.

E1.3.11 Pseudocode description of floating-point operations

The following subsections contain pseudocode definitions of the floating-point functionality supported by the architecture:

- Generation of specific floating-point values.
- Floating-point negation and absolute value.
- Floating-point value unpacking.
- Floating-point exception and NaN handling.
- Floating-point rounding.
- Selection of Arm standard floating-point arithmetic.
- Floating-point comparisons.
- Floating-point maximum and minimum.
- Floating-point addition and subtraction.
- Floating-point multiplication and division.
- Floating-point fused multiply-add.
- Floating-point reciprocal estimate and step.
- Floating-point square root.
- Floating-point reciprocal square root estimate and step.
- Floating-point conversions.

E1.3.11.1 Generation of specific floating-point values

The following pseudocode functions generate specific floating-point values. The `sign` argument is `'0'` for the positive version and `'1'` for the negative version:

- `FPInfinity()`.
- `FPMaxNormal()`.
- `FPZero()`.
- `FPTwo()`.
- `FPThree()`.
- `FPDefaultNaN()`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E1.3.11.2 Floating-point negation and absolute value

The floating-point negation and absolute value operations affect only the sign bit. They do not treat NaN operands specially, nor denormalized number operands when flush-to-zero is selected.

The floating-point negation operation is described by the pseudocode function `FPNeg()`. The floating-point absolute value operation is described by the pseudocode function `FPAbs()`.

E1.3.11.3 Floating-point value unpacking

The `FPUnpack()` function determines the type of a floating-point number, defined by `FPType{}`, and its numerical value. It also does flush-to-zero processing on input operands.

E1.3.11.4 Floating-point exception and NaN handling

The `FPProcessException()` procedure checks whether a floating-point exception is trapped, and handles it accordingly. The floating-point exception types are defined by `FPExc{}`.

The `FPProcessNaN()` function processes a NaN operand, producing the correct result value and generating an Invalid Operation floating-point exception if necessary. The `FPProcessNaNs()` function performs the standard NaN processing for a two-operand operation. The `FPProcessNaNs3()` function performs the standard NaN processing for a three-operand operation.

E1.3.11.5 Floating-point rounding

The `FPRound()` function rounds and encodes a floating-point result to a specified destination format. This includes processing Overflow, Underflow and Inexact floating-point exceptions and performing flush-to-zero processing on result values.

E1.3.11.6 Selection of Arm standard floating-point arithmetic

The `StandardFPCR()` function returns the FPSCR value that selects Arm standard floating-point arithmetic. Most of the arithmetic functions have a Boolean `fpscr_controlled` argument that is `TRUE` for Floating-point operations and `FALSE` for Advanced SIMD operations, and that selects between using the real FPSCR value and this value.

E1.3.11.7 Floating-point comparisons

The `FPCompare()` function compares two floating-point numbers, producing a {N, Z, C, V} Condition flags result as shown in Table E1-6:

Table E1-6 Effect of a Floating-point comparison on the Condition flags

Comparison result	N	Z	C	V
Equal	0	1	1	0
Less than	1	0	0	0
Greater than	0	0	1	0
Unordered	0	0	1	1

This result defines the operation of the `VCMP` floating-point instruction. The `VCMP` instruction writes these flag values in the FPSCR. After using a `VMRS` instruction to transfer them to the APSR, they can control conditional execution as shown in Table F1-1.

The `FPCompareEQ()`, `FPCompareGE()`, and `FPCompareGT()` functions describe the operation of Advanced SIMD instructions that perform floating-point comparisons.

E1.3.11.8 Floating-point maximum and minimum

The `FPMax()` function returns the maximum of two floating-point numbers. The `FPMin()` function returns the minimum of two floating-point numbers.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E1.3.11.9 Floating-point addition and subtraction

The `FPAdd()` function adds two floating-point numbers. The `FPSub()` function subtracts one floating-point number from another floating-point number.

E1.3.11.10 Floating-point multiplication and division

The `FPMul()` function multiplies two floating-point numbers. The `FPDiv()` function divides one floating-point number by another floating-point number.

E1.3.11.11 Floating-point fused multiply-add

The `FPMulAdd()` function performs a floating-point fused multiply-add.

E1.3.11.12 Floating-point reciprocal estimate and step

The Advanced SIMD implementation includes instructions that support Newton-Raphson calculation of the reciprocal of a number.

The `VRECPE` instruction produces the initial estimate of the reciprocal. It uses the pseudocode functions:

- `FPRecipEstimate()`.
- `UnsignedRecipEstimate()`.

Table E1-7 shows the results where input values are out of range.

Table E1-7 VRECPE results for out of range inputs

Number type	Input Vm[i]	Result Vd[i]
Integer	<= `0x7FFF_FFFF`	`0xFFFF_FFFF`
Floating-point	NaN	Default NaN
Floating-point	±0 or denormalized number	±infinity[a]
Floating-point	±infinity	±0
Floating-point	Absolute value >= 2^{126}	±0

a. FPSCR.DZC is set to 1

The Newton-Raphson iteration:

$xn+1 = xn(2-dxn)$

converges to $(1/d)$ if $x0$ is the result of `VRECPE` applied to d.

The `VRECPS` instruction performs a (2 - op1 × op2) calculation and can be used with a multiplication to perform a step of this iteration. The functionality of this instruction is defined by the `FPRecipStep()` pseudocode function.

Table E1-8 shows the results where input values are out of range.

Table E1-8 VRECPS results for out of range inputs

Input Vn[i]	Input Vm[i]	Result Vd[i]
Any NaN	-	Default NaN
-	Any NaN	Default NaN

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Input Vn[i]	Input Vm[i]	Result Vd[i]
±0.0 or denormalized number	±infinity	2.0
±infinity	±0.0 or denormalized number	2.0

E1.3.11.13 Floating-point square root

The `FPSqrt()` function returns the square root of a floating-point number.

E1.3.11.14 Floating-point reciprocal square root estimate and step

The Advanced SIMD implementation includes instructions that support Newton-Raphson calculation of the reciprocal of the square root of a number.

The `VRSQRTE` instruction produces the initial estimate of the reciprocal of the square root. It uses the pseudocode functions:

- `FPRSqrtEstimate()`.
- `UnsignedRSqrtEstimate()`.

Table E1-9 shows the results where input values are out of range.

Table E1-9 VRSQRTE results for out of range inputs

Number type	Input Vm[i]	Result Vd[i]
Integer	<= `0x3FFF_FFFF`	`0xFFFF_FFFF`
Floating-point	NaN, -(normalized number), -infinity	Default NaN
Floating-point	-0 or -(denormalized number)	- infinity[a]
Floating-point	+0 or +(denormalized number)	+infinity[a]
Floating-point	+infinity	+0

a. FPSCR.DZC is set to 1.

The Newton-Raphson iteration:

$xn+1 = xn(3-dxn^2)/2$

converges to $(1/\sqrt{d})$ if $x0$ is the result of `VRSQRTE` applied to d.

The `VRSQRTS` instruction performs a $(3 - op1 \times op2)/2$ calculation and can be used with two multiplications to perform a step of this iteration. The functionality of this instruction is defined by the `FPRSqrtStep()` pseudocode function.

Table E1-10 shows the results where input values are out of range.

Table E1-10 VRSQRTS results for out of range inputs

Input Vn[i]	Input Vm[i]	Result Vd[i]
Any NaN	-	Default NaN
-	Any NaN	Default NaN
±0.0 or denormalized number	±infinity	1.5
±infinity	±0.0 or denormalized number	1.5

`FPRSqrtStep()` calls the `FPHalvedSub()` pseudocode function.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E1.3.11.15 Floating-point conversions

The `FPConvert()` pseudocode function performs conversions between half-precision, single-precision, and double-precision floating-point numbers.

The `FPToFixed()` and `FixedToFP()` functions perform conversions between floating-point numbers and integers or fixed-point numbers.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E1.4 About the AArch32 System register interface

AArch32 state provides a System register encoding space, which is indexed by the parameter set {`coproc`, `opc1`, `CRn`, `CRm`, `opc2`}, and a set of System register access instructions. This encoding space is used for:

- System registers.
- System instructions, for:
  - Cache and branch predictor maintenance.
  - Address translation.
  - TLB maintenance.

This encoding space uses only the `coproc` values `0b111x`.

Note

The encoding space with `coproc` values `0b101x` is redefined to provide Advanced SIMD and floating-point functionality.

The `coproc` encodings provide access to System register encoding space as follows:

- The (`coproc` ==`0b1111`) encodings provide system control functionality, by providing access to System registers and System instructions. This includes architecture and feature identification, as well as control, status information and configuration support.

  The following sections give a general description of these encodings:
  - About the System registers for VMSAv8-32.
  - Organization of registers in the (coproc == `0b1111`) encoding space.
  - Functional grouping of VMSAv8-32 System registers.

  These encodings also provide:
  - The Performance Monitor registers. For more information, see The Performance Monitors Extension. The Activity Monitor registers. For more information, see The Activity Monitors Extension.
- The (`coproc` ==`0b1110`) encodings provide access to additional registers, which support:
  - Debug, see AArch32 Self-hosted Debug.
  - The Jazelle identification registers, see Jazelle support.

UNPREDICTABLE, CONSTRAINED UNPREDICTABLE, and UNDEFINED behavior for AArch32 System register accesses gives information more information about permitted accesses to the System registers in AArch32 state.

Most functionality in the (`coproc` ==`0b111x`) encoding space cannot be accessed by software executing at EL0. This manual clearly identifies those functions that can be accessed at EL0.

For more information:

- About this encoding space, including the naming of the parameters that index the space, see The AArch32 System register interface.
- About the System interface access instructions, see System register access instructions.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E1.5 Exceptions

The Arm architecture uses the following terms to describe various types of exceptional condition:

Exceptions

In the Arm architecture, an *exception* causes entry to EL1, EL2, or EL3. If the Exception level that is entered is using AArch32, it also causes entry to the PE mode in which the exception must be taken. A software handler for the exception is then executed.

Note

The term *floating-point exception* does not use this meaning of *exception*. This term is described later in this list.

Exceptions include:

- Reset.
- Interrupts.
- Memory system aborts.
- Undefined instructions.
- Supervisor calls (SVCs), Secure Monitor calls (SMCs), and Hypervisor calls (HVCs).
- Debug exceptions.

Most details of exception handling are not visible to application level software, and are described in Handling exceptions that are taken to an Exception level using AArch32. In an implementation that includes all the Exception levels, aspects that are visible to application level software are:

- The `SVC` instruction causes a Supervisor Call exception. This provides a mechanism for unprivileged software to make a call to the operating system, or other system component that is accessible only at EL1.
- The `SMC` instruction causes a Secure Monitor Call exception, but only if software execution is at EL1 or higher. Unprivileged software can only cause a Secure Monitor Call exception by methods defined by the operating system, or by another component of the software system that executes at EL1 or higher.
- The `HVC` instruction causes a Hypervisor Call exception, but only if software execution is at EL1 or higher. Unprivileged software can only cause a Hypervisor Call exception by methods defined by the hypervisor, or by another component of the software system that executes at EL1 or higher.
- The `BKPT` instruction causes a Breakpoint Instruction exception, which is taken as a Prefetch Abort exception. This provides a mechanism for a debugger to insert breakpoints into unprivileged software, or for unprivileged software to make a call into a debugger that is accessible at EL1.
- The `WFI` (Wait for Interrupt) instruction provides a hint that nothing needs to be done until an interrupt or another WFI wakeup event occurs, see Wait For Interrupt. This means the hardware might enter a low-power state until the wakeup event occurs.
- The `WFE` (Wait for Event) instruction provides a hint that nothing needs to be done until either an `SEV` instruction generates an event, or another WFE wakeup event occurs, see Wait For Event and Send Event. This means the hardware might enter a low-power state until the wakeup event occurs.

Floating-point exceptions

These relate to exceptional conditions encountered during floating-point arithmetic, such as Divide by Zero or Overflow. For more information, see:

- Floating-point exceptions and exception traps.
- The FPEXC and FPSCR register descriptions.
- ANSI/IEEE Std. 754, *IEEE Standard for Binary Floating-Point Arithmetic*.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Chapter E2
The AArch32 Application Level Memory Model

This chapter gives an application level description of the memory model for software executing in AArch32 state. This means it describes the memory model for execution in EL0 when EL0 is using AArch32 in the following sections:

- About the Arm memory model.
- Atomicity in the Arm architecture.
- Definition of the memory model.
- Ordering of translation table walks.
- Caches and memory hierarchy.
- Alignment support.
- Endian support.
- Memory types and attributes.
- Mismatched memory attributes.
- Synchronization and semaphores

Note

In this chapter, System register names usually link to the description of the register in AArch32 System Register Descriptions, for example SCTLR.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E2.1 About the Arm memory model

The Arm architecture is a weakly ordered memory architecture that permits the observation and completion of memory accesses in a different order from the program order. The following sections of this chapter provide the complete definition of the memory model, this introduction is not intended to contradict the definition found in those sections. In general, the basic principles of the memory model are:

- To provide a memory model that has similar weaknesses to those found in the memory models used by high-level programming languages such as C or Java. For example, by permitting independent memory accesses to be reordered as seen by other observers.
- To avoid the requirement for multi-copy atomicity in the majority of memory types.
- The provision of instructions and memory barriers to compensate for the lack of multi-copy atomicity in the cases where it would be needed.
- The use of address, data, and control dependencies in the creation of order so as to avoid having excessive numbers of barriers or other explicit instructions in common situations where some order is required by the programmer or the compiler.

This section contains:

- Address space.
- Memory type overview.

E2.1.1 Address space

Address calculations are performed using 32-bit registers. Supervisory software determines the valid address range.

Attempting to access an address that is not valid generates an MMU fault.

Address calculations are performed modulo 2^{32}.

The result of an address calculation is UNKNOWN if it overflows or underflows the 32-bit address range A[31:0].

Memory accesses use the `MemA[]`, `MemO[]`, `MemU[]`, and `MemU_unpriv[]` pseudocode functions:

- The `MemA[]` function makes an aligned access of the required type.
- The `MemO[]` function makes an ordered access of the required type.
- The `MemU[]` function makes an unaligned access of the required type
- The `MemU_unpriv[]` function makes an unaligned, unprivileged access of the required type.

Each of these functions calls `Mem_with_type[]` function, which specifies the required access. This calls `AArch32.MemSingle[]`, which performs an atomic, little-endian read of `size` bytes.

The `AccessDescriptor` type defines the different access types and attributes.

Note

- The AArch32 System Level Memory Model and The AArch32 Virtual Memory System Architecture include descriptions of memory system features that are transparent to the application, including memory access, address translation, memory maintenance instructions, and alignment checking and the associated fault handling. These chapters also reference pseudocode descriptions of these operations.
- For references to the pseudocode that relates to memory accesses, see Basic memory access, Unaligned memory access, and Aligned memory access.

E2.1.2 Memory type overview

The architecture provides the following mutually-exclusive memory types:

Normal This is generally used for the majority of memory operations, both read/write and read-only operations.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Device The Arm architecture forbids speculative reads of any type of Device memory. This means Device memory types are suitable attributes for read-sensitive locations.

Locations of the memory map that are assigned to peripherals are usually assigned the Device memory attribute.

Device memory has additional attributes that have the following effects:

- They prevent aggregation of reads and writes, maintaining the number and size of the specified memory accesses. See Gathering.
- They preserve the access order and synchronization requirements, both for accesses to a single peripheral and where there is a synchronization requirement on the observability of one or more memory write and read accesses. See Reordering
- They indicate whether a write can be acknowledged other than at the end point. See Early Write Acknowledgement.

For more information on Normal memory and Device memory, see Memory types and attributes.

Note

Earlier versions of the Arm architecture defined a single Device memory type and a Strongly-ordered memory type. A Note in Device memory describes how these memory types map onto the memory types used from the introduction of Armv8.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E2.2 Atomicity in the Arm architecture

Atomicity is a feature of memory accesses, described as *atomic* accesses. The Arm architecture description refers to two types of atomicity, *single-copy atomicity* and *multi-copy atomicity*. The atomicity requirements for memory accesses depend on the memory type, and whether the access is explicit or implicit. For more information, see:

- Requirements for single-copy atomicity.
- Properties of single-copy atomic accesses.
- Multi-copy atomicity.
- Requirements for multi-copy atomicity.
- Concurrent modification and execution of instructions.

For more information about the memory types, see Memory type overview.

E2.2.1 Requirements for single-copy atomicity

In AArch32 state, the single-copy atomic PE accesses are:

- All byte accesses.
- All halfword accesses to halfword-aligned locations.
- All word accesses to word-aligned locations.
- Memory accesses caused by `LDREXD` and `STREXD` instructions to doubleword-aligned locations.

`LDM`, `LDC`, `LDRD`, `STM`, `STC`, `STRD`, `PUSH`, `POP`, `RFE`, `SRS`, `VLDM`, `VLDR`, `VSTM`, and `VSTR` instructions are executed as a sequence of word-aligned word accesses. Each 32-bit word access is guaranteed to be single-copy atomic. The architecture does not require subsequences of two or more word accesses from the sequence to be single-copy atomic.

`LDRD` and `STRD` accesses to 64-bit aligned locations are 64-bit single-copy atomic as seen by translation table walks and accesses to translation tables.

Note

This requirement has been added to avoid the need for complex measures to avoid atomicity issues when changing translation table entries, without creating a requirement that all locations in the memory system are 64-bit single-copy atomic. This addition means:

- The system designer must ensure that all writable memory locations that might be used to hold translations, such as bulk SDRAM, can be accessed with 64-bit single-copy atomicity.
- Software must ensure that translation tables are not held in memory locations that cannot meet this atomicity requirement, such as peripherals that are typically accessed using a narrow bus.

This requirement places no burden on read-only memory locations for which reads have no side effects, since it is impossible to detect the size of memory accesses to such locations.

Advanced SIMD element and structure loads and stores are executed as a sequence of accesses of the element or structure size. The architecture requires the element accesses to be single-copy atomic if and only if both:

- The element size is 32 bits, or smaller.
- The elements are naturally-aligned.

Accesses to 64-bit elements or structures that are 32-bit aligned are executed as a sequence of 32-bit accesses, each of which is single-copy atomic. The architecture does not require subsequences of two or more 32-bit accesses from the sequence to be single-copy atomic.

When an access is not single-copy atomic by the rules described in this section, it is executed as a sequence of one or more accesses that aggregate to the size of the original access. Each of the accesses in this sequence is single-copy atomic, at least at the byte level.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Note

In this section, the terms *before the write operation* and *after the write operation* mean before or after the write operation has had its effect on the coherence order of the bytes of the memory location accessed by the write operation.

If, according to these rules, an instruction is executed as a sequence of accesses, a synchronous Data Abort exception or Debug state entry can be taken during that sequence. This causes execution of the instruction to be abandoned. See Data Abort exception and, when FEAT_LSMAOC is implemented, Taking an interrupt or other exception during a multiple-register load or store.

If the synchronous Data Abort exception is returned from using the preferred return address, the instruction that generated the sequence of accesses is re-executed and so any access that was performed before the exception was taken is repeated. This also applies to an exit from Debug state.

Note

The exception behavior for these multiple access instructions means they are not suitable for use for writes to memory for the purpose of software synchronization.

For implicit accesses:

- Cache linefills and evictions have no effect on the single-copy atomicity of explicit transactions or instruction fetches.
- Instruction fetches are single-copy atomic:
  - At 32-bit granularity in A32 state.
  - At 16-bit granularity in T32 state.

  Concurrent modification and execution of instructions describes additional constraints on the behavior of instruction fetches.
- Translation table walks are performed using accesses that are single-copy atomic:
  - At 32-bit granularity when using Short-descriptor format translation tables.
  - At 64-bit granularity when using Long-descriptor format translation tables.

E2.2.2 Properties of single-copy atomic accesses

A memory access instruction that is single-copy atomic has the following properties:

1. For a pair of overlapping single-copy atomic store instructions, all of the overlapping writes generated by one of the stores are Coherence-after the corresponding overlapping writes generated by the other store.
2. For a single-copy atomic load instruction L_1 that overlaps a single-copy atomic store instruction S_2, if one of the overlapping reads generated by L_1 Reads-from-memory one of the overlapping writes generated by S_2, then none of the overlapping writes generated by S_2 are Coherence-after the corresponding overlapping reads generated by L_1.

For more information, see Definition of the memory model.

E2.2.3 Multi-copy atomicity

In a multiprocessing system, writes to a memory location are *multi-copy atomic* if the following conditions are both true:

- All writes to the same location are *serialized*, meaning they are observed in the same order by all observers, although some observers might not observe all of the writes.
- A read of a location does not return the value of a write until all observers observe that write.

Note

Writes that are not coherent are not multi-copy atomic.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E2.2.4 Requirements for multi-copy atomicity

For Normal memory, writes are not required to be multi-copy atomic.

For Device memory, writes are not required to be multi-copy atomic.

The memory model is Other-multi-copy-atomic. For more information, see External visibility requirement.

E2.2.5 Concurrent modification and execution of instructions

The architecture limits the set of instructions that can be executed by one thread of execution as they are being modified by another thread of execution without requiring explicit synchronization.

Concurrent modification and execution of instructions can lead to the resulting instruction performing any behavior that can be achieved by executing any sequence of instructions that can be executed from the same Exception level, except where the instruction before modification and the instruction after modification are:

- When executing the A32 instruction set, a `B`, `BKPT`, `BL`, `HVC`, `ISB`, `NOP`, `SMC`, or `SVC` instruction.
- When executing the T32 instruction set. a 16-bit `B`, `BKPT`, `BLX`, `BX`, `NOP`, or `SVC` instruction.

In addition, for the 32-bit T32 instructions, for which Instruction encodings describes the meaning of {`hw1`, `hw2`}:

- `hw1` of a 32-bit `BL` (immediate) instruction can be concurrently modified to `hw1` of another `BL` (immediate) instruction:
  - This means that some of the most significant bits of the immediate value can be modified.
- `hw1` of a 32-bit `BLX` (immediate) instruction can be concurrently modified to `hw1` of another `BLX` immediate instruction:
  - This means that some of the most significant bits of the immediate value can be modified.
- `hw1` of a 32-bit `BL` (immediate) or `BLX` (immediate) instruction can be concurrently modified to a T32 16-bit `B`, `BX`, `BLX`, `BKPT`, or `SVC` instruction. This modification also works in reverse.
- `hw2` of a 32-bit `BL` (immediate) instruction can be concurrently modified to `hw2` of another `BL` (immediate) instruction with a different immediate:
  - This means that some bits of the immediate value, including the least significant bits, can be modified.
- `hw2` of a 32-bit `BLX` (immediate) instruction can be concurrently modified to `hw2` of another `BLX` (immediate) instruction with a different immediate:
  - This means that some bits of the immediate value, including the least significant bits, can be modified.
- `hw2` of a 32-bit `B` (immediate) instruction with a condition field can be concurrently modified to `hw2` of another 32-bit `B` (immediate) instruction with a condition field with a different immediate:
  - This means that some bits of the immediate value, including the least significant bits, can be modified.
- `hw2` of a 32-bit `B` (immediate) instruction without a condition field can be concurrently modified to `hw2` of another 32-bit `B` (immediate) instruction without a condition field:
  - This means that some bits of the immediate value, including the least significant bits, can be modified.

Note

- In the T32 instruction set:
  - The only encodings of `BKPT` and `SVC` are 16-bit.
  - The only encoding of `BL` is 32-bit.
- The `ISB` instruction can be concurrently modified and executed in the A32 and A64 instruction sets, but not in the T32 instruction set.

For the instructions explicitly identified in this section, the architecture guarantees that, after modification of the instruction, behavior is consistent with execution of either:

- The instruction originally fetched.
- A fetch of the modified instruction.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The instructions to which this applies are the `B`, `BL`, `NOP`, `BKPT`, `SVC`, `HVC`, and `SMC` instructions.

For both instruction sets, if one thread of execution changes a conditional branch instruction to another conditional branch instruction, and the change affects both the condition field and the branch target, execution of the changed instruction by another thread of execution before the change is synchronized can lead to either:

- The old condition being associated with the new target address.
- The new condition being associated with the old target address.

These possibilities apply regardless of whether the condition, either before or after the change to the branch instruction, is the *always* condition.

For all other instructions, to avoid UNPREDICTABLE or CONSTRAINED UNPREDICTABLE behavior, instruction modifications must be explicitly synchronized before they are executed. The required synchronization is as follows:

1. No PE must be executing an instruction when another PE is modifying that instruction.
2. To ensure that the modified instructions are observable, a PE that is writing the instructions must issue the following sequence of instructions and operations:

```
; Coherency example for self-modifying code
; Enter this code with <Rt> containing a new 32-bit instruction,
; to be held in Cacheable space at a location pointed to by Rn. Use STRH in the first
; line instead of STR for a 16-bit instruction.
STR <Rt>, [Rn]
DCCMVAU Rn                      ; Clean data cache by MVA to point of unification (PoU)
DSB                         ; Ensure visibility of the data stored
ICIMVAU Rn                      ; Invalidate instruction cache by VA to PoU
BPIMVA Rn                       ; Invalidate branch predictor by MVA to PoU
DSB
```

Note

- The `DCCMVAU` operation is not required if the area of memory is either Non-cacheable or Write-Through Cacheable.
- If the contents of physical memory differ between the mappings, changing the mapping of VAs to PAs can cause the instructions to be concurrently modified by one PE and executed by another PE. If the modifications affect instructions other than those listed as being acceptable for modification, synchronization must be used to avoid UNPREDICTABLE or CONSTRAINED UNPREDICTABLE behavior.

3. In a multiprocessor system, the `ICIMVAU` and `BPIMVA` are broadcast to all PEs within the Inner Shareable domain of the PE running this sequence. However, once the modified instructions are observable, each PE that is executing the modified instructions must issue the following instruction to ensure execution of the modified instructions:

```
        ISB                             ; Synchronize fetched instruction stream
```

For more information about the required synchronization operation, see Synchronization and coherency issues between data and instruction accesses.

For information about the Memory Effects generated by instruction fetches, see External visibility requirement.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E2.3 Definition of the memory model

This section describes observation and ordering in the memory model. It contains the following subsections:

- Basic and dependency definitions.
- Ordering relations and constraints.
- Ordering of instruction fetches.
- Restrictions on the effects of speculation.
- Memory barriers.

For more information on endpoint ordering of memory accesses, see Reordering.

In the memory model, the Shareability memory attribute indicates the degree to which hardware must ensure memory coherency between a set of observers, see Memory types and attributes.

The architecture defines additional memory attributes and associated behaviors, which are defined in the system level section of this manual. See:

- The AArch32 System Level Memory Model.
- The AArch32 Virtual Memory System Architecture.

See also Mismatched memory attributes.

E2.3.1 Basic and dependency definitions

The memory model provides a set of definitions that are used to construct conditions on the permitted sequences of accesses to memory. See Ordering requirements defined by the formal concurrency model for more information.

E2.3.2 Ordering relations and constraints

See Ordering requirements defined by the formal concurrency model for more information.

E2.3.3 Ordering of instruction fetches

For two memory locations A and B, if A has been written to with an updated value and been made coherent with the instruction fetches of the shareability domain before B has been written to with an updated value by an observer in the same shareability domain, then where, for an observer in the shareability domain, an instruction read from B appears in program order before an instruction fetched from A, if the instruction read from B contains the updated value of B then the instruction read from A appearing later in program order will contain the updated value of A.

A write has been made coherent with an instruction fetch of a shareability domain when:

CTR.{DIC, IDC} == {0, 0} The location written to has been cleaned to the Point of unification (PoU) from the data cache, and that clean is complete for the shareability domain. Subsequently the location has been invalidated to the Point of unification (PoU) from the instruction cache, and that invalidation is complete for the shareability domain.

CTR.{DIC, IDC} == {0, 1} The write is complete for the shareability domain. Subsequently the location has been invalidated to the Point of unification (PoU) from the instruction cache, and that invalidation is complete for the shareability domain.

CTR.{DIC, IDC} == {1, 1} The write is complete for the shareability domain.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Note

Microarchitecturally, this means that these situations cannot both be true in an implementation:

- After delays in fetching from memory, the instruction queue can have entries written into it out of order.
- For an implementation:
  - When CTR.DIC == 0, if there is an outstanding entry in the instruction queue, then later entries in the instruction queue are not impacted by the ICIMVAU instructions of a different core.
  - When CTR.DIC == 1, if there is a write to the location that is held in the queue when there is an outstanding entry in the instruction queue for an older entry, then the instruction queue does not have entries invalidated from it.

E2.3.4 AArch32 restrictions on the effects of speculation

The Arm architecture places certain restrictions on the effects of speculation. These are:

- Each load from a location using a particular VA after an exception return that is a Context Synchronization event will not speculatively read an entry from earlier in the coherence order for the location being loaded from than the entry generated by the latest store to that location using the same VA before the exception exit.
- Each load from a location using a particular VA after an exception entry that is a Context Synchronization event will not speculatively read an entry from earlier in the coherence order for the location being loaded from than the entry generated by the latest store to that location using the same VA before the exception entry.
- Any load from a location using a particular VA before an exception entry that is a Context Synchronization event will not speculatively read data from a store to the same location using the same VA after the exception entry.
- Any load from a location using a particular VA before an exception return that is a Context Synchronization event will not speculatively read data from a store to the same location using the same VA after the exception exit.
- When data is loaded under speculation with a Translation fault, it cannot be used to form an address or generate condition codes to be used by other instructions in the speculative sequence.
- When data is loaded under speculation from a location that does not have a valid translation for the translation regime being speculated in, the data cannot be used to form an address or generate condition codes to be used by other instructions in the speculative sequence.
- When data is loaded as a result of speculative accesses made after `TLBI` + `DSB` + `ERET` using a translation that was invalidated by the TLBI, the data cannot be used to form an address, generate condition codes, or generate SVE predicate values to be used by other instructions in the speculative sequence. The execution timing of any other instructions in the speculative sequence is not a function of the data loaded.
- Changes to System registers must not occur speculatively in a way that can affect a speculative memory access that can cause a change to the microarchitectural state.
- Changes to Special-purpose registers can occur speculatively.
- Execute-never controls apply to speculative instruction fetching. See Access permissions for instruction execution.
- When writing new instructions to memory, there is no requirement for an `SB` instruction to prevent speculative execution of the old code. See Instruction cache maintenance instructions.

Note

The prohibition of using data loaded under speculation with faults to form addresses, condition codes or SVE predicate values does not prohibit the use of value predicted data from such locations for such purposes, so long as the training of the data value prediction was from the hardware defined context that is using the prediction. A consequence of this is that training of value prediction cannot be based on data loaded under speculation with a translation or Permission fault.

E2.3.4.1 AArch32 Speculative Store Bypass Safe

When FEAT_SSBS is implemented, CPSR.SSBS is a control that can be set by software to indicate whether hardware is use, in a manner that is potentially speculatively exploitable, a speculative value in a register that has been loaded from memory using a load instruction that speculatively read an entry for the location being loaded from, where the entry that

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

is speculatively read is from earlier in the coherence order than the entry generated by the latest store to that location using the same virtual address as the load instruction.

A speculative value in a register is used in a potentially speculatively exploitable manner if it is used to form an address, generate condition codes, or generate SVE predicate values to be used by other instructions in the speculative sequence or if the execution timing of any other instructions in the speculative sequence is a function of the data loaded under speculation.

When the value of CPSR.SSBS is 0, hardware is not permitted to use speculative register values in a potentially speculatively exploitable manner if the speculative read that loads the register is from earlier in the coherence order than the entry generated by the latest store to that location using the same virtual address as the load instruction.

When the value of CPSR.SSBS is 1, hardware is permitted to use speculative register values in a potentially speculatively exploitable manner if the speculative read that loads the register is from earlier in the coherence order than the entry generated by the latest store to that location using the same virtual address as the load instruction.

Note

- If speculation is permitted, then cache timing side channels can lead to addresses being derived using reads of address values that have been speculatively loaded from memory to a register.
- Before the introduction of FEAT_SSBS, the control bit SPSR.SSBS was RES0, and therefore software written without awareness of FEAT_SSBS is expected to program it to 0. This means that CPSR.SSBS will not be set, so the PE will not be permitted to use speculative loads with outstanding memory disambiguation issues for any subsequent speculative memory accesses if there is any possibility of those subsequent memory accesses creating a cache timing side channel.

E2.3.4.2 Definition of exploitative control of speculative execution

The execution of some code (code1) can exploitatively control speculative execution of some other code (code2) if and only if all of the following apply:

- The actions of code1 can influence, in a manner that is not hard-to-determine, the prediction of multi-bit values that determine speculative execution of code2 to cause an irreversible change to the microarchitectural state of the PE that is indicative of some architectural state accessible to the execution context of code2.
- code1 has control in determining the choice of the architectural state that the irreversible change to the microarchitectural state is indicative of.
- The irreversible changes to the microarchitectural state of the PE can be measured by code executing in an execution context other than that of code2 to allow the retrieval of the architectural state in a computationally feasible manner that is not hard-to-determine.

E2.3.4.3 Definition of predictive leakage to speculative execution

The execution of some code (code1) can predictively leak to some other code (code2) if and only if all of the following apply:

- The execution of code1 influences, in a manner that is not hard-to-determine, the predictive microarchitectural structures of the implementation that predict multi-bit values, not binary choices, to behave in a way that is indicative of some architectural state accessible to the execution context of code1.
- The predictive microarchitectural structures of the implementation impact the timing of the speculative execution of code2 in a way that enables code2 to recover the architectural state in a manner that is not hard-to-determine.
- code1 and code2 are not collaborating to communicate using the mechanisms in the previous two bullets.

Note

Mechanisms to prevent the influence and the state recovery being “not hard-to-determine” are left open to implementations. Examples could include the complete separation of prediction resources, or the isolation of the predictions using a cryptographic or pseudo-random mechanism to separate each context.

E2.3.4.4 Further restrictions on the effects of speculation from Armv8.5

Further restrictions on speculation are introduced by some additional architectural features as described here.

FEAT_CSV3 introduces these restrictions:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- An attempt to access data under speculation, where the data is inaccessible given the current state of the PE cannot be used to alter the architectural or microarchitectural state in a manner that allows the value of the inaccessible data to be recovered by code architecturally executed through means such as measuring the timing of code. This restriction applies to any of the following cases which make the data inaccessible:
  - reading from a Location with a Permission or Domain fault.
  - directly reading from a system register that is not architecturally readable.
  - reading from a SIMD&FP register that is not accessible due to traps configured by a more privileged Exception level.
  - indirectly reading from a system register containing a Pointer Authentication Key for which the corresponding Pointer Authentication instructions are disabled.

  Note

  The current state of the PE is defined as the EL, values of all special-purpose registers, and values of all system registers under which the hardware attempts to access the data, which may be on the speculative execution path. If on the speculative execution path, the hardware must respect any changes to the PE state that occur on the speculative path, respecting the same rules that would be required if executing on the non-speculative path.

  Note

  The following is a list of examples in which use of the inaccessible data could lead to recovery of the value of the inaccessible data:

  - to form an address or generate condition codes to be used in the speculative sequence.
  - to form the register value used for the comparison of a compare-and-branch, test-and-branch, or compare-and-swap instruction in the speculative sequence. Note that this is to cover conditional instructions that do not use condition codes to determine their outcomes.
  - by prediction mechanisms such as Cache Prefetch or Data Value predictions.
  - as an input to an instruction in the speculative sequence where the execution timing of the instruction is dependent on the data value.

  Note

  It is permissible for a value of zero to be produced by the load or register read and consumed by an instruction newer than the load in the speculative sequence even if the value of zero results in different execution timing compared to non-zero values.

  Note

  As the effects of speculation are not architecturally visible, this restriction level requires that the effect of any speculation cannot give rise to side channels that will leak the values of memory locations, System registers, or Special-purpose registers to a level of privilege that would otherwise not be able to determine those values.

- Changes to System registers must not occur speculatively in a way that can affect a speculative memory access that can cause a change to the microarchitectural state.

  Note

  Changes to Special-purpose registers can occur speculatively.

FEAT_CSV2 and FEAT_CSV2_1p1 introduce a range of additional restrictions:

If FEAT_CSV2 is implemented:

- Code running in one hardware-defined context (context1) cannot either exploitatively control, or predictively leak to, the speculative execution of code in a different hardware-defined context (context2), as a result of the behavior of any of the following resources:
  - Branch target prediction based on the branch targets used in context1.
    - This applies to both direct and indirect branches, including return instructions, but excludes the prediction of the direction of a conditional branch.
  - Data Value predictions based on data value from execution in context1.

    Note

    The prediction of the PSTATE.{N,Z,C,V} values is not considered a data value for this purpose.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- Virtual address-based cache prefetch predictions generated as a result of execution in context1.
- Any other prediction mechanisms, other than Branch, Data Value, or Cache Prefetch predictions.
  In this definition, the hardware-defined context is determined by:
  - The Exception level.
  - The Security state.
  - When executing at EL1, if EL2 is implemented and enabled in the current Security state, the VMID.
  - When executing at EL0, whether the EL1&0 or the EL2&0 translation regime is in use.
  - When executing at EL0 and using the EL1&0 translation regime, the ASID and, if EL2 is implemented and enabled in the current Security state, the VMID.
  - When executing at EL0 and using the EL2&0 translation regime, the ASID.

If FEAT_CSV2_1p1 is implemented:

- Code running in one hardware-defined context (context1) cannot either exploitatively control, or predictively leak to, the speculative execution of code in a different hardware-defined context (context2) as a result of the behavior of branch target prediction based on the branch history used in context1.
- Branch or data values trained from one instruction address cannot exploitatively control, or predictively leak to, the speculative execution of code from a different address.

E2.3.5 Memory barriers

The Arm architecture is a weakly ordered memory architecture that supports out of order completion. *Memory barrier* is the general term applied to an instruction, or sequence of instructions, that forces synchronization events by a PE with respect to retiring load/store instructions. The memory barriers defined by the architecture provide a range of functionality, including:

- Ordering of load/store instructions.
- Completion of load/store instructions.
- Context synchronization.

The following subsections describe the memory barrier instructions:

- Instruction Synchronization Barrier.
- Data Memory Barrier.
- Data Synchronization Barrier.
- Speculation Barrier.
- Consumption of Speculative Data Barrier.
- Speculative Store Bypass Barrier.
- Physical Speculative Store Bypass Barrier.
- Trace Synchronization Barrier.
- Shareability and access limitations on the data barrier operations.
- Load-Acquire, Store-Release.

Note

Depending on the required synchronization, a program might use memory barriers on their own, or it might use them in conjunction with cache maintenance and memory management instructions that in general are available only when software execution is at EL1 or higher.

The `DMB` and `DSB` memory barriers affect reads and writes to the memory system generated by load/store instructions and data or unified cache maintenance instructions being executed by the PE.

AArch32 state also supports the legacy barrier instructions CP15DMB, CP15DSB, and CP15ISB. These instructions are executed as `MCR`s using the appropriate encoding, and are accessible from EL0. However, for performance reasons Arm deprecates any use of these operations, and strongly recommends that software uses the `DMB`, `DSB`, and `ISB` instructions

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

described in this section instead. Optionally, an implementation can support a CP15BEN control that supervisory software can use to disable use of these instructions, meaning the corresponding MCR encodings are UNDEFINED. When the CP15BEN control is supported, setting one of the following CP15BEN fields to 0 makes execution of CP15DMB, CP15DSB, and CP15ISB UNDEFINED:

- SCTLR_EL1.CP15BEN, for execution of these instructions at EL0 using AArch32 when EL1 is using AArch64.
- SCTLR.CP15BEN, for execution of these instructions at EL0 or EL1 when EL1 is using AArch32.
- HSCTLR.CP15BEN, for execution of these instructions at EL2 when EL2 is using AArch32.

E2.3.5.1 Instruction Synchronization Barrier

An ISB instruction ensures that all instructions that come after the ISB instruction in program order are fetched from the cache or memory after the ISB instruction has completed. Using an ISB ensures that the effects of context-changing operations executed before the ISB are visible to the instructions fetched after the ISB instruction. Examples of context-changing operations that require the insertion of an ISB instruction to ensure the effects of the operation are visible to instructions fetched after the ISB instruction are:

- Completed cache and TLB maintenance instructions.
- Changes to System registers.

Any context-changing operations appearing in program order after the ISB instruction take effect only after the ISB has been executed.

The pseudocode function for the operation of an ISB is `InstructionSynchronizationBarrier()`.

See also Memory barriers.

E2.3.5.2 Data Memory Barrier

The DMB instruction is a memory barrier instruction that ensures the relative order of memory accesses before the barrier with memory accesses after the barrier. The DMB instruction does not ensure the completion of any of the memory accesses for which it ensures relative order.

The full definition of the DMB instruction is covered formally in the Definition of the memory model and this introduction to the DMB instruction is not intended to contradict that section.

The basic principle of a DMB instruction is to introduce order between memory accesses that are specified to be affected by the DMB options supplied as arguments to the DMB instruction. The DMB instruction ensures that all affected memory accesses by the PE executing the DMB instruction that appear in program order before the DMB instruction and those which originate from a different PE, to the extent required by the DMB options, which have been Observed-by the PE before the DMB instruction is executed, are Observed-by each PE, to the extent required by the DMB options, before any affected memory accesses that appear in program order after the DMB instruction are Observed-by that PE.

The use of a DMB instruction creates order between the Memory effects of instructions as described in the definition of Barrier-ordered-before.

The pseudocode function for the operation of a DMB instruction is `DataMemoryBarrier()`.

E2.3.5.3 Data Synchronization Barrier

A DSB instruction is a memory barrier that ensures that memory accesses that occur before the DSB instruction have completed before the completion of the DSB instruction. In doing this, it acts as a stronger barrier than a DMB and all ordering that is created by a DMB with specific options is also generated by a DSB with the same options.

Execution of a DSB at EL2 ensures that any memory accesses caused by speculative translation table walks from the Non-secure PL1&0 translation regime have been observed.

For more information, see Use of out-of-context translation regimes.

A DSB executed by a PE, PEe, completes when all of the following apply:

- All explicit memory effects of the required access types appearing in program order before the DSB are complete for the set of observers in the required shareability domain.
- If the required access types of the DSB is reads and writes, the following instructions issued by PEe before the DSB are complete for the required shareability domain:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

  - All cache maintenance instructions.
  - All AArch32 TLB maintenance instructions.
  - All PSB instructions.
- If the required access types of the DSB is reads and writes, completion of a DSB instruction executed by PEe ensures that:
  - All previous TLB maintenance operations generated by AArch32 TLB maintenance instructions executed at EL1 by PEe when HCRX_EL2.FnXS is 1 are finished for all PEs in the shareability domain of the DSB instruction.
  - All previous TLB maintenance operations generated by AArch32 TLB maintenance instructions are finished for all PEs in the shareability domain of the DSB instruction.

In addition, no instruction that appears in program order after the DSB instruction can alter any state of the system or perform any part of its functionality until the DSB completes, other than:

- Being fetched from memory and decoded, or any change to architectural or microarchitectural state resulting from being fetched or decoded.
- Indirectly or directly reading any of the following, so long as the reading of these resources does not cause a change in architectural state:
  - General-purpose registers
  - SIMD&FP registers
  - The Stack Pointer register
  - The Program Counter
  - Special-purpose registers
- If FEAT_ETS2 is not implemented, having any virtual addresses of loads and stores translated.

The pseudocode function for the operation of a DSB is `DataSynchronizationBarrier()`.

See also Memory barrier instructions and Memory barriers.

E2.3.5.4 Speculation Barrier

An SB instruction is a memory barrier that prevents speculative execution of instructions until after the barrier has completed.

Until the barrier completes, the speculative execution of any instruction appearing later in the program order than the barrier cannot be performed to the extent that such speculation can be observed through side-channels as a result of control flow speculation or data value speculation. An example is speculative allocation into any caching structure where the allocation of that entry could indicate data value present in memory or in the registers.

The speculative execution of an SB instruction cannot be as a result of any of the following:

- Control flow speculation.
- Data value speculation.

An SB instruction can complete when all of the following apply:

- It is known that it is not speculative, or it is speculative only as a result of either:
  - Speculating that an instruction that could generate an exception does not generate an exception.
  - Speculating past the point in the Execution stream where a precise asynchronous exception is taken.
- All the predicted data values generated by instructions appearing in program order before the SB instruction are architecturally resolved, and so are not speculative.

Note

The SB instruction has no effect on the use of prediction resources to predict the instruction stream that is being fetched, so long as the prediction of the instruction stream is not informed by data taken from the register outputs of the speculative execution of instructions appearing in program order after an uncompleted SB instruction.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E2.3.5.5 Consumption of Speculative Data Barrier

The `CSDB` instruction is a memory barrier instruction that controls speculative execution arising from data value prediction.

Any instruction, other than a branch instruction, that appears in program order after the CSDB cannot be speculatively executed using the results of any of the following predictions if those predictions come from instructions that appear in program order before the CSDB and have not been architecturally resolved:

- Data value predictions of any instructions.
- PSTATE.{N,Z,C,V} predictions of any instructions other than conditional branch instructions or conditional instructions that write to the PC.

Note

For purposes of the definition of CSDB, PSTATE.{N,Z,C,V} are not considered data values. This definition permits:

- Control flow speculation before and after the CSDB instruction.
- Speculative execution of conditional data processing instructions after the CSDB instruction, unless they use the results of data value or PSTATE.{N,Z,C,V} predictions of instructions appearing in program order before the CSDB instruction that have not been architecturally resolved.

E2.3.5.6 Speculative Store Bypass Barrier

The `SSBB` instruction is a memory barrier that prevents speculative loads from bypassing earlier stores to the same virtual address under certain conditions.

The semantics of the Speculative Store Bypass Barrier are:

- When a load to a location appears in program order after the `SSBB`, then the load does not speculatively read an entry earlier in the coherence order for that location than the entry generated by the latest store satisfying all of the following conditions:
  - The store is to the same location as the load.
  - The store uses the same virtual address as the load.
  - The store appears in program order before the `SSBB` instruction.
- When a load to a location appears in program order before the `SSBB`, then the load does not speculatively read data from any store satisfying all of the following conditions:
  - The store is to the same location as the load.
  - The store uses the same virtual address as the load.
  - The store appears in program order after the `SSBB` instruction.

E2.3.5.7 Physical Speculative Store Bypass Barrier

The `PSSBB` instruction is a memory barrier that prevents speculative loads from bypassing earlier stores to the same physical address under certain conditions.

The semantics of the Speculative Store Bypass Barrier are:

- When a load to a location appears in program order after the `PSSBB`, then the load does not speculatively read an entry earlier in the coherence order for that location than the entry generated by the latest store satisfying all of the following conditions:
  - The store is to the same location as the load.
  - The store appears in program order before the `PSSBB` instruction.
- When a load to a location appears in program order before the `PSSBB`, then the load does not speculatively read data from any store satisfying all of the following conditions:
  - The store is to the same location as the load.
  - The store appears in program order after the `PSSBB` instruction.

Note

The effect of this barrier applies to accesses to the same location even if they are accessed with different virtual addresses and from different Exception levels.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E2.3.5.8 Trace Synchronization Barrier

The `TSB` instruction is a barrier instruction that preserves the relative order of accesses to System registers due to trace operations and other accesses to the same registers.

A trace operation is an operation of the trace unit generating trace for an instruction when FEAT_TRF is implemented and enabled.

A `TSB` is not required to execute in program order with respect to other instructions. This includes being reordered with respect to other trace instructions. One or more Context synchronization events are required to ensure that `TSB` is executed in the necessary order.

If trace is generated between a Context synchronization event and a `TSB` operation, these trace operations may be reordered with respect to the `TSB` operation, and therefore may not be synchronized.

The following situations are synchronized using a `TSB`:

- A direct write B to a System register is ordered after an indirect read or indirect write of the same register by a trace operation of a traced instruction A, if all of the following are true:
  - A is executed in program order before a Context synchronization event C.
  - C is in program order before a `TSB` operation T.
  - B is executed in program order after T.
- A direct read B of a System register is ordered after an indirect write to the same register by a trace operation of a traced instruction A if all the following are true:
  - A is executed in program order before a Context synchronization event C1.
  - C1 is in program order before `TSB` operation T.
  - T is executed in program order before a second Context synchronization event C2.
  - B is executed in program order after C2.

A `TSB` is not needed when a direct write B to a System register is ordered before an indirect read or indirect write of the same register by a trace operation of a traced instruction A, if all the following are true:

- A is executed in program order after a Context synchronization event C.
- B is executed in program order before C.

The pseudocode function for the operation of a `TSB` is `TraceSynchronizationBarrier()`.

E2.3.5.9 Shareability and access limitations on the data barrier operations

The `DMB` and `DSB` instructions can each take an optional limitation argument that specifies:

- The shareability domain over which the instruction must operate. This is one of:
  - Full system.
  - Outer Shareable.
  - Inner Shareable.
  - Non-shareable.

  Full system applies to all the observers in the system and, as such, encompasses the Inner and Outer Shareable domains of the processor.

  Note

  The distinction between Full system and Outer Shareable is applicable only for Normal Non-cacheable memory accesses and Device memory accesses.

- The accesses for which the instruction operates. This is one of:
  - Read and write accesses, both before and after the barrier instruction.
  - Write accesses only, before and after the barrier instruction.
  - Read accesses before the barrier instruction, and read and write accesses after the barrier instruction.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Note

This form of a DMB or DSB instruction can be described as a load-load/store barrier.

For more information on whether an access is before or after a barrier instruction, see Data Memory Barrier or Data Synchronization Barrier.

Table E2-1 shows how these options are encoded in the <option> field of the instruction.

Table E2-1 Encoding of the DMB and DSB <option> parameter

Accesses		Shareability domain			
Before the barrier	**After the barrier**	**Full system**	**Outer Shareable**	**Inner Shareable**	**Non-shareable**
Reads and writes	Reads and writes	SY	OSH	ISH	NSH
Writes	Writes	ST	OSHST	ISHST	NSHST
Reads	Reads and writes	LD	OSHLD	ISHLD	NSHLD

If no <option> is specified then the instruction operates for read and write accesses, over the full system, meaning the operation is the same as for the SY option. See the instruction descriptions for more information:

- DMB.
- DSB.

Note

ISB also supports an optional limitation argument that can contain only one value that corresponds to full system operation, see ISB.

E2.3.5.10 Load-Acquire, Store-Release

The architecture provides a set of instructions with Acquire semantics for loads, and Release semantics for stores.

The full definition of the Load-Acquire instruction is covered formally in the Definition of the memory model and this introduction to the Load-Acquire instruction is not intended to contradict that section.

The basic principle of a Load-Acquire instruction is to introduce order between the memory access generated by the Load-Acquire instruction and the memory accesses appearing in program order after the Load-Acquire instruction, such that the memory access generated by the Load-Acquire instruction is Observed-by each PE, to the extent that PE is required to observe the access coherently, before any of the memory accesses appearing in program order after the Load-Acquire instruction are Observed-by that PE, to the extent that the PE is required to observe the accesses coherently.

The use of a Load-Acquire instruction creates order between the Memory effects of instructions as described in the definition of Barrier-ordered-before.

The full definition of the Store-Release instruction is covered formally in the Definition of the memory model and this introduction to the Store-Release instruction is not intended to contradict that section.

The basic principle of a Store-Release instruction is to introduce order between the memory accesses generated by the PEe executing the Store-Release instruction, together with those which originate from a different PE, to the extent that the PEe is required to observe them coherently, Observed-by the PEe before executing the Store-release.

The use of a Store-Release instruction creates order between the Memory effects of instructions as described in the definition of Barrier-ordered-before.

In addition, the use of a Load-Acquire or a Store-Release instruction on accesses to a Memory-mapped peripheral introduces order between the Memory effects of the instructions that access that peripheral, as described in the definition of Peripheral arrival order.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Load-Acquire and Store-Release, other than `LDAEXD` and `STLEXD`, access only a single data element. This access is single-copy atomic. The address of the data object must be aligned to the size of the data element being accessed, otherwise the access generates an Alignment fault.

`LDAEXD` and `STLEXD` access two data elements. The address supplied to the instructions must be doubleword-aligned, otherwise the access generates an Alignment fault.

A Store-Release Exclusive instruction has the release semantics only if the store is successful.

Note

- Each Load-Acquire Exclusive and Store-Release Exclusive instruction is essentially a variant of the equivalent Load-Exclusive or Store-Exclusive instruction. All usage restrictions and single-copy atomicity properties:
  - — That apply to the Load-Exclusive instructions also apply to the Load-Acquire Exclusive instructions.
  - — That apply to the Store-Exclusive instructions also apply to the Store-Release Exclusive instructions.
- The Load-Acquire/Store-Release instructions can remove the requirement to use the explicit `DMB` memory barrier instruction.

Table E2-2 summarizes the Load-Acquire/Store-release instructions.

Table E2-2 Load-Acquire/Store-Release instructions

Data type	Load-Acquire	Store-Release	Load-Acquire Exclusive	Store-Release Exclusive
32-bit word	`LDA`	`STL`	`LDAEX`	`STLEX`
16-bit halfword	`LDAH`	`STLH`	`LDAEXH`	`STLEXH`
8-bit byte	`LDAB`	`STLB`	`LDAEXB`	`STLEXB`
64-bit doubleword	-	-	`LDAEXD`	`STLEXD`

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E2.4 Ordering of translation table walks

If FEAT_ETS2 is implemented, then an Effect E_1 is Ordered-before an Effect E_2 when all of the following apply:

- E_1 is not generated by an Advanced SIMD or floating-point instruction.
- E_2 is not generated for an instruction fetch.
- E_1 is an Explicit Memory Effect.
- E_1 appears in program order before E_3.
- E_3 is a TLBUncacheable Fault Effect.
- E_2 is Translation-intrinsically-before E_3.
- E_2 is an Implicit TTD Memory Read Effect.

The full definition of the ordering implications of FEAT_ETS2 is covered formally in Ordering requirements defined by the formal concurrency model, and the rules in this section are not intended to contradict that section.

For more information about ordering requirements to FEAT_ETS2, see DSB-ordered-before and ETS-ordered-before.

If FEAT_ETS3 is implemented, then an Effect E_1 is Ordered-before an Effect E_2 when all of the following apply:

- E_1 is not generated by an Advanced SIMD or floating-point instruction.
- E_2 is not generated for an instruction fetch.
- E_1 is an Explicit Memory Effect.
- E_1 appears in program order before E_3.
- E_3 is an MMU Fault Effect.
- E_2 is Translation-intrinsically-before E_3.
- E_2 is an Implicit TTD Memory Read Effect.

The full definition of the ordering implications of FEAT_ETS3 is covered formally in Ordering requirements defined by the formal concurrency model, and the rules in this section are not intended to contradict that section.

For more information about ordering requirements to FEAT_ETS3, see DSB-ordered-before and ETS-ordered-before.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E2.5 Caches and memory hierarchy

The implementation of a memory system depends heavily on the microarchitecture and therefore many details of the memory system are IMPLEMENTATION DEFINED. The architecture defines the application level interface to the memory system, including a hierarchical memory system with multiple levels of cache. This section describes an application level view of this system. It contains the subsections:

- Introduction to caches.
- Memory hierarchy.
- Implication of caches for the application programmer.
- Prefetching into cache.

E2.5.1 Introduction to caches

A cache is a block of high-speed memory that contains a number of entries, each consisting of:

- Main memory address information, commonly known as a *tag*.
- The associated data.

Caches increase the average speed of a memory access and take account of two principles of locality:

Spatial locality

An access to one location is likely to be followed by accesses to adjacent locations. Examples of this principle are:

- Sequential instruction execution.
- Accessing a data structure.

Temporal locality

An access to an area of memory is likely to be repeated in a short time period. An example of this principle is the execution of a software loop.

To minimize the quantity of control information stored, the spatial locality property groups several locations together under the same tag. This logical block is commonly known as a *cache line*. When data is loaded into a cache, access times for subsequent loads and stores are reduced, resulting in overall performance benefits. An access to information already in a cache is known as a *cache hit*, and other accesses are called *cache misses*.

Normally, caches are self-managing, with the updates occurring automatically. Whenever the PE accesses a cacheable memory location, the cache is checked. If the access is a cache hit, the access occurs in the cache. Otherwise, the access is made to memory. Typically, when making this access, a cache location is allocated and the cache line loaded from memory. The architecture permits different cache topologies and access policies, provided they comply with the memory coherency model described in this manual.

Caches introduce a number of potential problems, mainly because:

- Memory accesses can occur at times other than when the programmer would expect them.
- A data item can be held in multiple physical locations.

E2.5.2 Memory hierarchy

Typically memory close to a PE has very low latency, but is limited in size and expensive to implement. Further from the PE it is common to implement larger blocks of memory but these have increased latency. To optimize overall performance, a memory system can include multiple levels of cache in a hierarchical memory system that exploits this trade-off between size and latency. Figure E2-1 shows an example of such a system in a system that supports virtual addressing.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Figure E2-1 Multiple levels of cache in a memory hierarchy

Note

In this manual, in a hierarchical memory system, Level 1 refers to the level closest to the PE, as shown in Figure E2-1.

Instructions and data can be held in separate caches or in a unified cache. A cache hierarchy can have one or more levels of separate instruction and data caches, with one or more unified caches located at the levels closest to the main memory. Memory coherency for cache topologies can be defined using the conceptual points Point of Unification (PoU) and Point of Coherency (PoC). For more information, including the definitions of PoU and PoC, see About cache maintenance in AArch32 state.

Note

FEAT_DPB adds architectural support for an additional conceptual point, Point of Persistence, but this support is provided only in AArch64 state. For more information, see About cache maintenance in AArch64 state.

E2.5.2.1 The Cacheability and Shareability memory attributes

Cacheability and Shareability are two attributes that describe the memory hierarchy in a multiprocessing system:

Cacheability This term defines whether memory locations are allowed to be allocated into a cache or not. Cacheability is defined independently for Inner and Outer Cacheability locations.

Shareability This term defines whether memory locations are shareable between different agents in a system. Marking a memory location as shareable for a particular domain requires hardware to ensure that the location is coherent for all agents in that domain. Shareability is defined independently for Inner and Outer Shareability domains.

For more information about the Cacheability and Shareability attributes, see Memory types and attributes.

E2.5.3 Implication of caches for the application programmer

In normal operation, the caches are largely invisible to the application programmer. However they can become visible when there is a breakdown in the coherency of the caches. Such a breakdown can occur:

- When memory locations are updated by other agents in the system that do not use hardware management of coherency.
- When memory updates made from the application software must be made visible to other agents in the system, without the use of hardware management of coherency.

For example:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- In the absence of hardware management of coherency of DMA accesses, in a system with a DMA controller that reads memory locations that are held in the data cache of a PE, a breakdown of coherency occurs when the PE has written new data in the data cache, but the DMA controller reads the old data held in memory.
- In a Harvard cache implementation, where there are separate instruction and data caches, a breakdown of coherency occurs when new instruction data has been written into the data cache, but the instruction cache still contains the old instruction data.

E2.5.3.1 Data coherency issues

Software can ensure the data coherency of caches in the following ways:

- By not using the caches in situations where coherency issues can arise. This can be achieved by:
  - Using Non-cacheable or, in some cases, Write-Through Cacheable memory.
  - Not enabling caches in the system.
- By using system calls to functions using cache maintenance instructions that execute at a higher Exception level.
- By using hardware coherency mechanisms to ensure the coherency of data accesses to memory for cacheable locations by observers within the different shareability domains, see Non-shareable Normal memory and Shareable, Inner Shareable, and Outer Shareable Normal memory.

Note

The performance of these hardware coherency mechanisms is highly IMPLEMENTATION SPECIFIC. In some implementations the mechanism suppresses the ability to cache shareable locations. In other implementations, cache coherency hardware can hold data in caches while managing coherency between observers within the shareability domains.

E2.5.3.2 Synchronization and coherency issues between data and instruction accesses

How far ahead of the current point of execution instructions are fetched from is IMPLEMENTATION DEFINED. Such prefetching can be either a fixed or a dynamically varying number of instructions, and can follow any or all possible future execution paths. For all types of memory:

- The PE might have fetched the instructions from memory at any time since the last Context Synchronization event on that PE.
- Any instructions fetched in this way might be executed multiple times, if this is required by the execution of the program, without being re-fetched from memory.

The Arm architecture does not require the hardware to ensure coherency between instruction caches and memory, even for locations of shared memory.

If software requires coherency between instruction execution and memory, it must manage this coherency using Context Synchronization events and cache maintenance instructions. These can be accessed only from an Exception level that is higher than EL0, and therefore require a system call, see Exception-generating and exception-handling instructions. The following code sequence can be used for this purpose:

```
; Coherency example for data and instruction accesses within the same Inner Shareable domain.
; Enter this code with <Rt> containing a new 32-bit instruction,
; to be held in Cacheable space at a location pointed to by Rn. Use STRH in the first line
; instead of STR for a 16-bit instruction.
        STR Rt, [Rn]
        DCCMVAU Rn          ; Clean data cache by MVA to point of unification (PoU)
        DSB                 ; Ensure visibility of the data cleaned from cache
        ICIMVAU Rn          ; Invalidate instruction cache by MVA to PoU
        BPIMVA  Rn          ; Invalidate branch predictor by MVA to PoU
        DSB                 ; Ensure completion of the invalidations
        ISB                 ; Synchronize the fetched instruction stream
```

A write has been made coherent with an instruction fetch of a shareability domain when:

CTR.{DIC, IDC} == {0, 0} The location written to has been cleaned to the Point of unification (PoU) from the data cache, and that clean is complete for the shareability domain. Subsequently the location has been invalidated to the Point of unification (PoU) from the instruction cache, and that invalidation is complete for the shareability domain.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CTR.{DIC, IDC} == {0, 1} The write is complete for the shareability domain. Subsequently the location has been invalidated to the Point of unification (PoU) from the instruction cache, and that invalidation is complete for the shareability domain.

CTR.{DIC, IDC} == {1, 1} The write is complete for the shareability domain.

Note

- For accesses that are Non-cacheable or Write-Through, the clean data cache instruction is not required. For accesses that are Non-cacheable, the invalidate instruction cache is not required, because in AArch32 state these accesses are not permitted to be held in an instruction cache.
- This code can be used when the thread of execution modifying the code is the same thread of execution that is executing the code. The architecture limits the set of instructions that can be executed by one thread of execution as they are being modified by another thread of execution without requiring explicit synchronization. See Concurrent modification and execution of instructions.

E2.5.4 Prefetching into cache

The Arm architecture provides the memory system hints `PLD` (Preload Data), `PLDW`(Preload Data With Intent To Write) and `PLI` (Preload Instruction) that software can use to communicate the expected use of memory locations to the hardware. The memory system can respond by taking actions that are expected to speed up the memory accesses if they occur. The effect of these memory system hints is IMPLEMENTATION DEFINED. Typically, implementations use this information to bring data or instruction locations into caches.

The prefetch instructions are hints, and so implementations can treat them as `NOP`s without affecting the functional behavior of the device. The instructions cannot generate synchronous Data Abort exceptions, but the resulting memory system operations might, under exceptional circumstances, generate an asynchronous External abort, which is reported using an SError exception and taken using an asynchronous Data Abort exception. For more information, see Data Abort exception.

A `PLD`, `PLDW`, or `PLI` instruction can only cause allocation to software-visible caching structures such caches or TLBs for memory locations that can be accessed, according to the permissions defined by the current translation regime or a translation regime for a higher Exception level in the current Security state, by any of:

- Reads.
- Writes.
- Instruction fetches.

A `PLD`, `PLDW`, or `PLI` instruction can access any memory location in Normal memory that can be accessed, according to the permissions defined by the current translation regime or a translation regime for a higher Exception level in the current Security state, by any of:

- Reads.
- Writes.
- Instruction fetches.

Note

In each case, the entire list applies to each of `PLD`, `PLDW`, and `PLI`.

A `PLD`, `PLDW`, or `PLI` instruction is guaranteed not to access any type of Device memory.

A `PLI` instruction must not perform any access that cannot be performed by a speculative instruction fetch by the processor. Therefore in a VMSA implementation, if all associated MMUs are disabled, a `PLI` instruction cannot access any memory location that cannot be accessed by instruction fetches.

The pseudocode enumeration `PrefetchHint` defines the prefetch hint types.

The `Hint_Prefetch()` pseudocode function signals to the memory system that memory accesses of the type `hint` to or from the specified `address` are likely to occur in the near future. The memory system might take some action to speed up the memory accesses when they do occur, such as prefetching the specified address into one or more caches as indicated by the innermost cache level `target` and non-temporal hint `stream`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

For more information on `PLD`, `PLI`, and `PLDW`, see:

- PLD, PLDW (immediate).
- PLD (literal).
- PLD, PLDW (register).
- PLI (immediate, literal).
- PLI (register).

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E2.6 Alignment support

This section describes alignment support. It contains the following subsections:

- Instruction alignment.
- Unaligned data access.
- Cases where unaligned accesses are CONSTRAINED UNPREDICTABLE.
- Unaligned data access restrictions.
- Generation of Alignment faults by load/store multiple accesses to Device memory.

For more information about Alignment faults, see Alignment faults.

E2.6.1 Instruction alignment

A32 instructions are word-aligned.

T32 instructions are halfword-aligned.

E2.6.2 Unaligned data access

An A-profile implementation must support unaligned data accesses to Normal memory by some load and store instructions. As Table E2-3 shows, software can control whether a misaligned access to Normal memory by one of these instructions causes an Alignment fault Data Abort exception:

- By setting SCTLR.A, for unaligned accesses from any mode other than Hyp mode.
- By setting HSCTLR.A, for unaligned accesses from Hyp mode.

Table E2-3 Alignment requirements of load/store instructions

Instructions	Alignment check	Result if check fails when: SCTLR.A or HSCTLR.A is 0	SCTLR.A or HSCTLR.A is 1
`LDRB`, `LDREXB`, `LDRBT`, `LDRSB`, `LDRSBT`, `STRB`, `STREXB`, `STRBT`, `TBB`	None	-	-
`LDRH`, `LDRHT`, `LDRSH`, `LDRSHT`, `STRH`, `STRHT`, `TBH`	Halfword	Unaligned access	Alignment fault
`LDREXH`, `STREXH`, `LDAH`, `STLH`, `LDAEXH`, `STLEXH`	Halfword	Alignment fault	Alignment fault
`LDR`, `LDRT`, `STR`, `STRT` `PUSH`, encodings T3 and A2 only `POP`, encodings T3 and A2 only	Word	Unaligned access	Alignment fault
`LDREX`, `STREX`, `LDA`, `STL`, `LDAEX`, `STLEX`	Word	Alignment fault	Alignment fault
`LDREXD`, `STREXD`, `LDAEXD`, `STLEXD`	Doubleword	Alignment fault	Alignment fault
All forms of `LDM` and `STM`, `LDRD`, `RFE`, `SRS`, `STRD`	Word	Alignment fault	Alignment fault
`LDC`, `STC`	Word	Alignment fault	Alignment fault
`VLDM`, `VPOP`, `VPUSH`, `VSTM`	Word	Alignment fault	Alignment fault
`VLDR`, `VSTR` - single-precision scalar and double-precision scalar	Word	Alignment fault	Alignment fault
`VLDR`, `VSTR` - half-precision scalar	Halfword	Alignment fault	Alignment fault
`VLD1`, `VLD2`, `VLD3`, `VLD4`, `VST1`, `VST2`, `VST3`, `VST4`, all with standard alignment	Element size	Unaligned access	Alignment fault

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Instructions	Alignment check	Result if check fails when: SCTLR.A or HSCTLR.A is 0	SCTLR.A or HSCTLR.A is 1
VLD1, VLD2, VLD3, VLD4, VST1, VST2, VST3, VST4, all with :<align> specified[a]	As specified by :<align>	Alignment fault	Alignment fault

a. Previous versions of this manual used @<align> to specify alignment. Both forms are supported, see T32 and A32 Advanced SIMD and Floating-point Instruction Descriptionsfor more information.

Note

Any unaligned access to any type of Device memory generates an Alignment fault, see Alignment faults.

E2.6.3 Cases where unaligned accesses are CONSTRAINED UNPREDICTABLE

Any load instruction that is not faulted by the alignment restrictions shown in Table E2-3 and that loads the PC has CONSTRAINED UNPREDICTABLE behavior if the address it loads from is not word-aligned, see Loads and Stores to unaligned locations. This overrules any permitted load/store behavior shown in Table E2-3.

E2.6.4 Unaligned data access restrictions

The following points apply to unaligned data accesses:

- Accesses are not guaranteed to be single-copy atomic except at the byte access level, see Atomicity in the Arm architecture.
- Unaligned accesses typically take a number of additional cycles to complete compared to a naturally-aligned access.
- An operation that performs an unaligned access can abort on any memory access that it makes, and can abort on more than one access. This means that an unaligned access that occurs across a page boundary can generate an abort on either side of the boundary.

E2.6.5 Generation of Alignment faults by load/store multiple accesses to Device memory

When FEAT_LSMAOC is implemented and the value of the applicable nTLSMD field is 0, any memory access by an AArch32 Load Multiple or Store Multiple instruction to an address that the stage 1 translation assigns as Device-nGRE, Device-nGnRE, or Device-nGnRnE generates an Alignment fault.

The applicable nTLSMD field is the field in the SCTLR_EL1, SCTLR_EL2, HSCTLR, or SCTLR register that applies to the Exception level and Security state at which the LDM or STM instruction is executed.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E2.7 Endian support

General description of endianness in the Arm architecture describes the relationship between endianness and memory addressing in the Arm architecture.

The following subsections then describe the endianness schemes supported by the architecture:

- Instruction endianness.
- Data endianness.
- Endianness of memory-mapped peripherals.

E2.7.1 General description of endianness in the Arm architecture

This section only describes memory addressing and the effects of endianness for data elements up to doubleword of 64 bits. However, this description can be extended to apply to larger data elements.

For an address A, Figure E2-2 shows, for big-endian and little-endian memory systems, the relationship between:

- The doubleword at address A.
- The words at addresses A and A+4.
- The halfwords at addresses A, A+2, A+4, and A+6.
- The bytes at addresses A, A+1, A+2, A+3, A+4, A+5, A+6, and A+7.

The terms in Figure E2-2 have the following definitions:

MSByte Most significant byte.

LSByte Least significant byte.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

In this figure, *Byte, A+1* is an abbreviation for *Byte at address A+1*

Figure E2-2 Endianness relationships in AArch32 state

E2.7.2 Instruction endianness

The mapping of instruction memory is always little-endian.

E2.7.3 Data endianness

The size of the data value that is loaded or stored is the size that is used for the purpose of endian conversion for floating-point, Advanced SIMD, and general-purpose register loads and stores.

Table E2-4 shows the element sizes of all the load/store instructions, for all instruction sets.

Table E2-4 Element size of load/store instructions

Instructions	Element size
`LDRB`, `LDREXB`, `LDRBT`, `LDRSB`, `LDRSBT`, `STRB`, `STREXB`, `STRBT`, `TBB`	Byte
`LDRH`, `LDREXH`, `LDRHT`, `LDRSH`, `LDRSHT`, `STRH`, `STREXH`, `STRHT`, `TBH`	Halfword
`LDR`, `LDRT`, `LDREX`, `STR`, `STRT`, `STREX`	Word
`LDRD`, `LDREXD`, `STRD`, `STREXD`	Word
All forms of `LDM`, `PUSH`, `POP`, `RFE`, `SRS`, all forms of `STM`,	Word
`LDC`, `STC`	Word

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Instructions	Element size
Forms of `VLDM`, `VLDR`, `VPOP`, `VPUSH`, `VSTM`, `VSTR` that transfer 32-bit Si registers	Word
Forms of `VLDM`, `VLDR`, `VPOP`, `VPUSH`, `VSTM`, `VSTR` that transfer 64-bit Di registers	Doubleword
`VLD1`, `VLD2`, `VLD3`, `VLD4`, `VST1`, `VST2`, `VST3`, `VST4`	Element size of the Advanced SIMD access

CPSR.E determines the data endianness.

The data size used for endianness conversions:

- Is the size of the data value that is loaded or stored for Advanced SIMD and floating-point register and general-purpose register loads and stores.
- Is the size of the data element that is loaded or stored for Advanced SIMD element and data structure loads and stores. For more information, see Endianness in Advanced SIMD.

E2.7.3.1 Instructions to reverse bytes in registers

An application or device driver might have to interface to memory-mapped peripheral registers or shared memory structures that are not the same endianness as the internal data structures. Similarly, the endianness of the operating system might not match that of the peripheral registers or shared memory. In these cases, the PE requires an efficient method to transform explicitly the endianness of the data.

Table E2-5 shows the instructions that provide this functionality in the T32 and A32 instruction sets.

Table E2-5 Byte reversal instructions

Function	T32/A32 instruction	Notes
Reverse bytes in whole register	`REV`	For use with general purpose registers
Reverse bytes in 16-bit halfwords	`REV16`	For use with general purpose registers
Reverse bytes in halfword and sign-extend	`REVSH`	For use with general purpose registers
Reverse elements in doublewords, vector	`VREV64`	For use with registers in the SIMD and floating-point register file
Reverse elements in words, vector	`VREV32`	For use with registers in the SIMD and floating-point register file
Reverse elements in halfwords, vector	`VREV16`	For use with registers in the SIMD and floating-point register file

E2.7.3.2 Endianness in Advanced SIMD

Advanced SIMD element load/store instructions transfer vectors of elements between memory and the SIMD and floating-point register file. An instruction specifies both the length of the transfer and the size of the data elements being transferred. This information is used by the PE to load and store data correctly in both big-endian and little-endian systems.

Consider, for example, the A32 or T32 instruction:

```
VLD1.16 {D0}, [R1]
```

This loads a 64-bit register with four 16-bit values. The four elements appear in the register in array order, with the lowest indexed element fetched from the lowest address. The order of bytes in the elements depends on the endianness configuration, as shown in Figure E2-3. Therefore, the order of the elements in the registers is the same regardless of the endianness configuration.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Figure E2-3 Advanced SIMD byte order example for AArch32 state

For information about the alignment of Advanced SIMD instructions, see Alignment support.

The `BigEndian()` pseudocode function determines the current endianness of the data.

The `BigEndianReverse()` pseudocode function reverses the endianness of a bitstring.

The `BigEndian()` and `BigEndianReverse()` functions are defined in A-profile Architecture Pseudocode.

E2.7.4 Endianness of memory-mapped peripherals

All memory-mapped peripherals defined in the Arm architecture must be little-endian.

Peripherals to which this requirement applies include:

- Memory-mapped register interfaces to a debugger, or to a cross-trigger interface, see About the External Debug Registers.
- The memory-mapped register interface to the system level implementation of the Generic Timer, see System Level Implementation of the Generic Timer.
- A memory-mapped register interface to the Performance Monitors, see Recommended External Interface to the Performance Monitors.
- A memory-mapped register interface to the Activity Monitors, see Recommended External Interface to the Activity Monitors.
- Memory-mapped register interfaces to an Arm Generic Interface Controller, see the *ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3 and version 4.*
- The memory-mapped register interface to an Arm trace component. See, for example, the *Arm® Embedded Trace Macrocell Architecture Specification, ETMv4.*

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E2.8 Memory types and attributes

The ordering of accesses for addresses in memory, referred to as the memory order model, is defined by the memory attributes. The following sections describe this model:

- Normal memory.
- Device memory.
- Memory access restrictions.

E2.8.1 Normal memory

The Normal memory type attribute applies to most memory in a system. It indicates that the hardware is permitted by the architecture to perform Speculative data read accesses to these locations, regardless of the access permissions for these locations.

The Normal memory type has the following properties:

- An Explicit Memory Write Effect to a memory location with the Normal attribute completes in finite time.
- Explicit Memory Write Effects to a memory location with the Normal memory type that is either Non-cacheable or Write-Through cacheable for both the Inner and Outer cacheability must reach the endpoint for that location in the memory system in finite time. Two writes to the same location, where at least one is using the Normal memory type, might be merged before they reach the endpoint unless there is an ordered-before relationship between the two writes. For the purposes of this requirement, the endpoint for a location in Conventional memory is the PoC.
- Unaligned memory accesses can access Normal memory if the system is configured to generate such accesses.
- There is no requirement for the memory system beyond the PE to be able to identify the elements accessed by multi-register load/store instructions. See Multi-register loads and stores that access Normal memory.

Note

- The Normal memory attribute is appropriate for locations of memory that are idempotent, meaning that they exhibit all of the following properties:
  - Read accesses can be repeated with no side-effects.
  - Repeated read accesses return the last value written to the resource being read.
  - Read accesses can fetch additional memory locations with no side-effects.
  - Write accesses can be repeated with no side-effects if the contents of the location accessed are unchanged between the repeated writes or as the result of an exception, as described in this section.
  - Unaligned accesses can be supported.
  - Accesses can be merged before accessing the target memory system.
- Normal memory allows speculative reads and may be affected by intermediate buffering and forwarding of data. If non-idempotent memory locations are mapped as Normal memory, the following may occur:
  - Memory accesses return UNKNOWN values.
  - UNPREDICTABLE effects on memory-mapped peripherals.
- An instruction that generates a sequence of accesses as described in Atomicity in the Arm architecture might be abandoned as a result of an exception being taken during the sequence of accesses. On return from the exception the instruction is restarted, and therefore one or more of the memory locations might be accessed multiple times. This can result in repeated write accesses to a location that has been changed between the write accesses.

The following sections describe the other attributes for Normal memory:

- Shareable Normal memory.
- Non-shareable Normal memory.
- Cacheability attributes for Normal memory.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

See also:

- Multi-register loads and stores that access Normal memory.
- Atomicity in the Arm architecture.
- Memory barriers. For accesses to Normal memory, a `DMB` instruction is required to ensure the required ordering.
- Concurrent modification and execution of instructions.

E2.8.1.1 Shareable Normal memory

A Normal memory location has a Shareability attribute that is defined as one of:

- Inner Shareable.
- Outer Shareable.
- Non-shareable.

The shareability attributes define the data coherency requirements of the location, which hardware must enforce. They do not affect the coherency requirements of instruction fetches, see Synchronization and coherency issues between data and instruction accesses.

Note

- System designers can use the Shareability attribute to specify the locations in Normal memory for which coherency must be maintained. However, software developers must not assume that specifying a memory location as Non-shareable permits software to make assumptions about the incoherency of the location between different PEs in a shared memory system. Such assumptions are not portable between different multiprocessing implementations that might use the Shareability attribute. Any multiprocessing implementation might implement caches that are shared, inherently, between different PEs.
- This architecture assumes that all PEs that use the same operating system or hypervisor are in the same Inner Shareable shareability domain.

E2.8.1.1.1 Shareable, Inner Shareable, and Outer Shareable Normal memory

The Arm architecture abstracts the system as a series of Inner and Outer Shareability domains.

Each Inner Shareability domain contains a set of observers that are data coherent for each member of that set for data accesses with the Inner Shareable attribute made by any member of that set.

Each Outer Shareability domain contains a set of observers that are data coherent for each member of that set for data accesses with the Outer Shareable attribute made by any member of that set.

The following properties also hold:

- Each observer is only a member of a single Inner Shareability domain.
- Each observer is only a member of a single Outer Shareability domain.
- All observers in an Inner Shareability domain are always members of the same Outer Shareability domain. This means that an Inner Shareability domain is a subset of an Outer Shareability domain, although it is not required to be a proper subset.

Note

- Because all data accesses to Non-cacheable locations are data coherent to all observers, Non-cacheable locations are always treated as Outer Shareable.
- The Inner Shareable domain is expected to be the set of PEs controlled by a single hypervisor or operating system.

The details of the use of the Shareability attributes are system-specific. Example E2-1 shows how they might be used.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Example E2-1 Use of shareability attributes

In an implementation, a particular subsystem with two clusters of PEs has the requirement that:

- In each cluster, the data caches or unified caches of the PEs in the cluster are transparent for all data accesses to memory locations with the Inner Shareable attribute.
- However, between the two clusters, the caches:
  - Are not required to be coherent for data accesses that have only the Inner Shareable attribute.
  - Are coherent for data accesses that have the Outer Shareable attribute.

In this system, each cluster is in a different Shareability domain for the Inner Shareable attribute, but all components of the subsystem are in the same Shareability domain for the Outer Shareable attribute.

A system might implement two such subsystems. If the data caches or unified caches of one subsystem are not transparent to the accesses from the other subsystem, this system has two Outer Shareable Shareability domains.

Having two levels of shareability means system designers can reduce the performance and power overhead for shared memory locations that do not need to be part of the Outer Shareable Shareability domain.

For Shareable Normal memory, the Load-Exclusive and Store-Exclusive synchronization primitives take account of the possibility of accesses by more than one observer in the same Shareability domain.

E2.8.1.2 Non-shareable Normal memory

For Normal memory locations, the Non-shareable attribute identifies Normal memory that is likely to be accessed only by a single PE.

A location in Normal memory with the Non-shareable attribute does not require the hardware to make data accesses by different observers coherent, unless the memory is Non-cacheable. For a Non-shareable location, if other observers share the memory system, software must use cache maintenance instructions, if the presence of caches might lead to coherency issues when communicating between the observers. This cache maintenance requirement is in addition to the barrier operations that are required to ensure memory ordering.

For Non-shareable Normal memory, it is IMPLEMENTATION DEFINED whether the Load-Exclusive and Store-Exclusive synchronization primitives take account of the possibility of accesses by more than one observer.

E2.8.1.3 Cacheability attributes for Normal memory

In addition to being Outer Shareable, Inner Shareable or Non-shareable, each region of Normal memory is assigned a Cacheability attribute that is one of:

- Write-Through Cacheable.
- Write-Back Cacheable.
- Non-cacheable.

Also, for Write-Through Cacheable and Write-Back Cacheable Normal memory regions:

- A region might be assigned cache allocation hints for read and write accesses.
- It is IMPLEMENTATION DEFINED whether the cache allocation hints can have an additional attribute of Transient or Non-transient.

For more information, see Cacheability, cache allocation hints, and cache transient hints.

A memory location can be marked as having different cacheability attributes, for example when using aliases in a VA to PA mapping:

- If the attributes differ only in the cache allocation hint this does not affect the behavior of accesses to that location.
- For other cases, see Mismatched memory attributes.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The cacheability attributes provide a mechanism of coherency control with observers that lie outside the Shareability domain of a region of memory. In some cases, the use of Write-Through Cacheable or Non-cacheable regions of memory might provide a better mechanism for controlling coherency than the use of hardware coherency mechanisms or the use of cache maintenance routines. For this reason, the architecture requires the following properties for Non-cacheable or Write-Through Cacheable memory:

- A completed write to a memory location that is Non-cacheable or Write-Through Cacheable for a level of cache made by an observer accessing the memory system inside the level of cache is visible to all observers accessing the memory system outside the level of cache without the need of explicit cache maintenance.
- A completed write to a memory location that is Non-cacheable for a level of cache made by an observer accessing the memory system outside the level of cache is visible to all observers accessing the memory system inside the level of cache without the need of explicit cache maintenance.
- For accesses to Normal memory that is Non-cacheable, a DMB instruction introduces a Barrier-ordered-before relation on all accesses to a single peripheral or block of memory that is of IMPLEMENTATION DEFINED size. For more information, see Ordering relations and constraints.

Note

Implementations can use the cache allocation hints to indicate a probable performance benefit of caching. For example, a programmer might know that a piece of memory is not going to be accessed again and would be better treated as Non-cacheable. The distinction between memory regions with attributes that differ only in the cache allocation hints exists only as a hint for performance.

For Normal memory, the Arm architecture provides cacheability attributes that are defined independently for each of two conceptual levels of cache, the *inner* and the *outer* cache. The relationship between these conceptual levels of cache and the implemented physical levels of cache is IMPLEMENTATION DEFINED, and can differ from the boundaries between the Inner and Outer Shareability domains. However:

- Inner refers to the innermost caches, meaning the caches that are closest to the PE, and always includes the lowest level of cache.
- No cache that is controlled by the Inner cacheability attributes can lie outside a cache that is controlled by the Outer cacheability attributes.
- An implementation might not have any outer cache.

Example E2-2, Example E2-3, and Example E2-4 describe the possible ways of implementing a system with three levels of cache, *level 1* (L1) to *level 3* (L3).

Note

- L1 cache is the level closest to the PE, see Memory hierarchy.
- When managing coherency, system designs must consider both the inner and outer cacheability attributes, as well as the Shareability attributes. This is because hardware might have to manage the coherency of caches at one conceptual level, even when another conceptual level has the Non-cacheable attribute.

Example E2-2 Implementation with two inner and one outer cache levels

Implement the three levels of cache in the system, L1 to L3, with:

- The Inner cacheability attribute applied to L1 and L2 cache.
- The Outer cacheability attribute applied to L3 cache.

Example E2-3 Implementation with three inner and no outer cache levels

Implement the three levels of cache in the system, L1 to L3, with the Inner cacheability attribute applied to L1, L2, and L3 cache. Do not use the Outer cacheability attribute.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Example E2-4 Implementation with one inner and two outer cache levels

Implement the three levels of cache in the system, L1 to L3, with:

- The Inner cacheability attribute applied to L1 cache.
- The Outer cacheability attribute applied to L2 and L3 cache.

E2.8.1.4 Multi-register loads and stores that access Normal memory

For all instructions that load or store more than one general-purpose register from an Exception level there is no requirement for the memory system beyond the PE to be able to identify the size of the elements accessed by these load or store instructions.

For all instructions that load or store more than one general-purpose register from an Exception level the order in which the registers are accessed is not defined by the architecture.

For all instructions that load or store one or more registers from the SIMD and floating-point register file from an Exception level there is no requirement for the memory system beyond the PE to be able to identify the size of the element accessed by these load or store instructions.

E2.8.2 Device memory

The Device memory type attributes define memory locations where an access to the location can cause side-effects, or where the value returned for a load can vary depending on the number of loads performed. Typically, the Device memory attributes are used for memory-mapped peripherals and similar locations.

The attributes for Device memory are:

Gathering Identified as G or nG, see Gathering.

Reordering Identified as R or nR, see Reordering.

Early Write Acknowledgement

Identified as E or nE, see Early Write Acknowledgement.

The Device memory types are:

Device-nGnRnE Device non-Gathering, non-Reordering, No Early Write Acknowledgement.

Equivalent to the Strongly-ordered memory type in earlier versions of the architecture.

Device-nGnRE Device non-Gathering, non-Reordering, Early Write Acknowledgement.

Equivalent to the Device memory type in earlier versions of the architecture.

Device-nGRE Device non-Gathering, Reordering, Early Write Acknowledgement.

Armv8 added this memory type to the translation table formats found in earlier versions of the architecture. The use of barriers is required to order accesses to Device-nGRE memory.

The Device-nGRE memory type is introduced into the AArch32 translation table formats when the PE is using the Long Descriptor Translation Table format.

Device-GRE Device Gathering, Reordering, Early Write Acknowledgement.

Armv8 added this memory type to the translation table formats found in earlier versions of the architecture. Device-GRE memory has the fewest constraints. It behaves similar to Normal memory, with the restriction that speculative accesses to Device-GRE memory is forbidden.

The Device-GRE memory type is introduced into the AArch32 translation table formats when the PE is using the Long Descriptor Translation Table format.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Collectively these are referred to as *any Device memory type*. Going down the list, the memory types are described as getting *weaker*; conversely the going up the list the memory types are described as getting *stronger*.

Note

- As the list of types shows, these additional attributes are hierarchical. For example, a memory location that permits Gathering must also permit Reordering and Early Write Acknowledgement.
- The architecture does not require an implementation to distinguish between each of these memory types and Arm recognizes that not all implementations will do so. The subsection that describes each of the attributes, describes the implementation rules for the attribute.
- Earlier versions of the Arm architecture defined the following memory types:
  - Strongly-ordered memory. This is the equivalent of the Device-nGnRnE memory type.
  - Device memory. This is the equivalent of the Device-nGnRE memory type.

All of these memory types have the following properties:

- Speculative data accesses are not permitted to any memory location with any Device memory attribute. This means that each memory access to any Device memory type must be one that would be generated by a simple sequential execution of the program.

  An exception to this applies:
  - Reads generated by the Advanced SIMD and floating-point instructions can access bytes that are not explicitly accessed by the instruction if the bytes accessed are in a 16-byte window, aligned to 16-bytes, that contains at least one byte that is explicitly accessed by the instruction.

  Note

  - An instruction that generates a sequence of accesses as described in Atomicity in the Arm architecture might be abandoned as a result of an exception being taken during the sequence of accesses. On return from the exception the instruction is restarted, and therefore one or more of the memory locations might be accessed multiple times. This can result in repeated accesses to a location where the program defines only a single access. For this reason, Arm strongly recommends that no accesses to Device memory are performed from a single instruction that spans the boundary of a translation granule or which in some other way could lead to some of the accesses being aborted.
  - Write speculation that is visible to other observers is prohibited for all memory types.
- An Explicit Memory Write Effect to a memory location with any Device memory type completes in finite time.
- If a value that would be returned from a read of a memory location with the Device memory type changes without an explicit memory write effect by an observer, this change must also be globally observed for all observers in the system in finite time. Such a change might occur in a peripheral location that holds status information.
- Data accesses to memory locations are coherent for all observers in the system, and correspondingly are treated as being Outer Shareable.
- A memory location with any Device memory attribute cannot be allocated into a cache.
- Explicit Memory Write Effects to a memory location with any Device memory attribute must reach the endpoint for that address in the memory system in finite time. Two Explicit Memory Write Effects of Device memory type to the same location might be merged before they reach the endpoint, unless both Explicit Memory Write Effects have the non-Gathering attribute or there is an ordered-before relationship between the two Explicit Memory Write Effects.
- If a memory location is not capable of supporting unaligned memory accesses, then an unaligned access to that memory location generates an Alignment fault at the first stage of translation that defined the location as being Device.
- If a memory location is capable of supporting unaligned memory accesses, and such a memory location is marked as Device, then it is IMPLEMENTATION DEFINED whether an unaligned access to that memory location generates an Alignment fault at the first stage of translation that defined the location as being Device.
- Hardware does not prevent speculative instruction fetches from a memory location with any of the Device memory attributes unless the memory location is also marked as execute-never for all Exception levels.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Note

This means that to prevent speculative instruction fetches from memory locations with Device memory attributes, any location that is assigned any Device memory type must also be marked as execute-never for all Exception levels. Failure to mark a memory location with any Device memory attribute as execute-never for all Exception levels is a programming error.

Note

In the Non-secure PL1&0 translation regime in systems where HCR.TGE == 1 and HCR.DC == 0, any Alignment fault that results from the fact that all locations are treated as Device is a fault at the first stage of translation. This causes the value of HSR.ISS.[24] to be 0.

See also Memory access restrictions.

The memory types for translation table walks cannot be defined as any Device memory type within the TCR. For the Non-secure EL1&0 translation regime, the memory accesses made during a stage 1 translation table walk are subject to a stage 2 translation, and as a result of this second stage of translation, the accesses from the first stage translation table walk might be made to memory locations with any Device memory type. These accesses might be made speculatively. When the value of the HCR.PTW bit is 1, a stage 2 Permission fault is generated if a first stage translation table walk is made to any Device memory type.

For an instruction fetch from a memory location with the Device attribute that is not marked as execute-never for the current Exception level, an implementation can either:

- Treat the instruction fetch as if it were to a memory location with the Normal Non-cacheable attribute.
- Take a Permission fault.

E2.8.2.1 Gathering

In the Device memory attribute:

G Indicates that the location has the Gathering attribute.

nG Indicates that the location does not have the Gathering attribute, meaning it is non-Gathering.

The Gathering attribute determines whether it is permissible for either:

- Multiple memory accesses of the same type, read or write, to the same memory location to be merged into a single transaction.
- Multiple memory accesses of the same type, read or write, to different memory locations to be merged into a single memory transaction on an interconnect.

For memory types with the Gathering attribute, either of these behaviors is permitted, provided that the ordering and coherency rules of the memory location are followed.

For memory types with the non-Gathering attribute, neither of these behaviors is permitted. As a result:

- The number of memory accesses that are made corresponds to the number that would be generated by a simple sequential execution of the program.
- All access occur at their programmed size, except that there is no requirement for the memory system beyond the PE to be able to identify the elements accessed by multi-register load/store instructions. See Multi-register loads and stores that access Device memory.

Gathering between memory accesses separated by a memory barrier that affects those memory accesses is not permitted.

Gathering between two memory accesses generated by a Load-Acquire/Store-Release is not permitted.

A read from a memory location with the non-Gathering attribute cannot come from a cache or a buffer, but must come from the endpoint for that address in the memory system. Typically this is a peripheral or physical memory.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Note

- A read from a memory location with the Gathering attribute can come from intermediate buffering of a previous write, provided that:
  - The accesses are not separated by a DMB or DSB barrier that affects both of the accesses.
  - The accesses are not separated by other ordering constructions that require that the accesses are in order. Such a construction might be a combination of Load-Acquire and Store-Release.
  - The accesses are not generated by a Store-Release instruction.
- The Arm architecture defines only programmer visible behavior. Therefore, gathering can be performed if a programmer cannot tell whether gathering has occurred.

An implementation is permitted to perform an access with the Gathering attribute in a manner consistent with the requirements specified by the non-Gathering attribute.

An implementation is not permitted to perform an access with the non-Gathering attribute in a manner consistent with the relaxations allowed by the Gathering attribute.

E2.8.2.2 Reordering

In the Device memory attribute:

R Indicates that the location has the Reordering attribute.

nR Indicates that the location does not have the Reordering attribute, meaning it is non-Reordering.

For all memory types with the non-Reordering attribute, the order of memory accesses arriving at a single peripheral of IMPLEMENTATION DEFINED size, as defined by the peripheral, must be the same order that occurs in a simple sequential execution of the program. That is, the accesses appear in program order. This ordering applies to all accesses using any of the memory types with the non-Reordering attribute. As a result, if there is a mixture of Device-nGnRE and Device-nGnRnE accesses to the same peripheral, these occur in program order. If the memory accesses are not to a peripheral, then this attribute imposes no restrictions.

Note

- The IMPLEMENTATION DEFINED size of the single peripheral is the same as applies for the ordering guarantee provided by the DMB instruction.
- The Arm architecture defines only programmer visible behavior. Therefore, reordering can be performed if a programmer cannot tell whether reordering has occurred.

An implementation is permitted to perform an access with the Reordering attribute in a manner consistent with the requirements specified by the non-Reordering attribute.

An additional relaxation is that an implementation is not permitted to perform an access with the non-Reordering attribute in a manner consistent with the relaxations allowed by the Reordering attribute.

The non-Reordering attribute does not require any additional ordering, other than that which applies to Normal memory, between:

- Accesses to one physical address with the non-Reordering attribute and accesses to a different physical address with the Reordering attribute.
- Access to one physical address with the non-Reordering attribute and access to a different physical address to Normal memory.
- Accesses with the non-Reordering attribute and accesses to different peripherals of IMPLEMENTATION DEFINED size.

E2.8.2.3 Early Write Acknowledgement

In the Device memory attribute:

E Indicates that the location has the Early Write Acknowledgement attribute.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

nE Indicates that the location has the No Early Write Acknowledgement attribute.

If the No Early Write Acknowledgement attribute is assigned for a Device memory location:

- For memory system endpoints where the system architecture in which the PE is operating requires that acknowledgement of a write comes from the endpoint, it is guaranteed that:
  - Only the endpoint of the write access returns a write acknowledgement of the access.
  - No earlier point in the memory system returns a write acknowledgement.
- For memory system endpoints where the system architecture in which the PE is operating does not require that acknowledgement of a write comes from the endpoint, the acknowledgement of a write is not required to come from the endpoint.

Note

A write with the No Early Write Acknowledgement attribute assigned for a Device memory location is not expected to generate an abort in any situation where the equivalent write to the same location without the No Early Write Acknowledgement attribute assigned does not generate an abort.

This means that a `DSB` barrier instruction, executed by the PE that performed the write to the No Early Write Acknowledgement Location, completes only after the write has reached its endpoint in the memory system if that is required by the system architecture.

Peripherals are an example of system endpoints that require that the acknowledgment of a write comes from the endpoint.

Note

- The Early Write Acknowledgement attribute affects only where the endpoint acknowledgment is returned from, and does not affect the ordering of arrival at the endpoint between accesses, which is determined by either the Device Reordering attribute, or the use of barriers to create order.
- The areas of the physical memory map for which write acknowledgment from the endpoint is required is outside the scope of the Arm Architecture definition and must be defined as part of the system architecture in which the PE is operating. In particular, regions of memory handled as PCIe configuration writes are expected to support write acknowledgment from the endpoint.
- Arm recognizes that not all areas of a physical memory map will be capable of supporting write acknowledgment from the endpoint. In particular, Arm expects that regions of memory handled as posted writes under PCIe will not support write acknowledgment from the endpoint.
- For maximum software compatibility, Arm strongly recommends that all peripherals for which standard software drivers expect that the use of a `DSB` instruction will determine that a write has reached its endpoint are placed in areas of the physical memory map that support write acknowledgment from the endpoint.

E2.8.2.4 Multi-register loads and stores that access Device memory

For all instructions that load or store more than one general-purpose register there is no requirement for the memory system beyond the PE to be able to identify the size of the elements accessed by these load and store instructions.

For all instructions that load or store one or more registers from the SIMD and floating-point register file there is no requirement for the memory system beyond the PE to be able to identify the size of the element accessed by these load and store instructions.

The architecture permits that the non-speculative execution of an instruction that loads or stores more than one general-purpose or SIMD and floating-point register might result in repeated accesses to the same address, even if the resulting accesses are to any type of Device memory.

For an `LDRD`, `STRD`, or `LDM` instruction with a register list that includes the PC, or an `STM` instruction with a register list that includes the PC, the order in which the registers are accessed is not defined by the architecture.

For a load or store of an Advanced SIMD element or structure, the order in which the registers are accessed is not defined by the architecture.

For a `VLDM` and `VSTM` instruction with a register list that does not include the PC, all registers are accessed in ascending address order for accesses to Device memory with the non-Reordering attribute.

For a `LDM` or `STM` instruction with a register list that does not include the PC:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- When FEAT_LSMAOC is not implemented, and when FEAT_LSMAOC is implemented and the value of the applicable LSMAOE field is 1, all registers are accessed in ascending address order for accesses to Device memory with the non-Reordering attribute.
- When FEAT_LSMAOC is implemented and the value of the applicable LSMAOE field is 0, no memory accesses are required to be ordered.
- When FEAT_LSMAOC is implemented and the value of the applicable nTLSMD field is 0, any memory access to an address that the stage 1 translation assigns as Device-nGRE, Device-nGnRE, or Device-nGnRnE generates an Alignment fault.

The applicable LSMAOE or nTLSMD field is the field in the SCTLR_EL1, SCTLR_EL2, HSCTLR, or SCTLR register that applies to the Exception level and Security state at which the `LDM` or `STM` instruction is executed.

Armv8.2 deprecates software relying on accesses to Device memory made by a single `LDM` or `STM` instruction not being reordered.

E2.8.3 Memory access restrictions

The following restrictions apply to memory accesses:

- For two explicit memory read effects to any two adjacent bytes in memory, *p* and *p+1*, generated by the same instruction, and for two explicit memory write effects to any two adjacent bytes in memory, *p* and *p+1*, that are generated by the same instruction:
  - The bytes *p* and *p+1* must have the same memory type and Shareability attributes. otherwise the results are CONSTRAINED UNPREDICTABLE. For example, an `LDC`, `LDM`, `LDRD STC`, `STM or STRD` instruction, or an unaligned load or store that spans the boundary between Normal memory and Device memory is CONSTRAINED UNPREDICTABLE.
  - Except for possible differences in the cache allocation hints, Arm deprecates having different cacheability attributes for bytes *p* and *p+1*.

  For the permitted CONSTRAINED UNPREDICTABLE behavior, see Crossing a page boundary with different memory types or Shareability attributes.
- If the accesses of an instruction that causes multiple accesses to any type of Device memory cross a 4KB address boundary then behavior is CONSTRAINED UNPREDICTABLE and Crossing a 4KB boundary with a Device access describes the permitted behaviors.

  Note

  — The boundary referred to is between two Device memory regions that are both of 4KB and aligned to 4KB.

  — This restriction means it is important that an access to a volatile memory device is not made using a single instruction that crosses a 4KB address boundary.

  — Arm expects this restriction to constrain the placing of volatile memory devices in the system memory map, rather than expecting a compiler to be aware of the alignment of memory accesses.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E2.9 Mismatched memory attributes

Mismatched memory attributes are controlled by privileged software. For more information, see The AArch32 Virtual Memory System Architecture.

Physical memory Locations are accessed with *mismatched attributes* if all accesses to the Location do not use a common definition of all of the following attributes of that Location:

- Memory type: Device-nGnRnE, Device-nGnRE, Device-nGRE, Device-GRE or Normal.
- Shareability.
- Cacheability, for the same level of the inner or outer cache, but excluding any cache allocation hints.

Collectively these are referred to as memory attributes.

Note

In this document, the terms *location* and *memory location* refer to any byte within the current coherency granule and are used interchangeably.

When a memory Location is accessed with mismatched attributes the only software visible effects are one or more of the following:

- Uniprocessor semantics for reads and writes to that memory Location might be lost. This means:
  - A read of the memory Location by one agent might not return the value most recently written to that memory Location by the same agent.
  - Multiple writes to the memory Location by one agent with different memory attributes might not be ordered in program order.
- There might be a loss of coherency when multiple agents attempt to access a memory Location.
- There might be a loss of properties derived from the memory type, as described in later bullets in this section.
- If all Load-Exclusive/Store-Exclusive instructions executed across all threads to access a given memory Location do not use consistent memory attributes, the Exclusives monitor state becomes UNKNOWN.
- Bytes written without the Write-Back cacheable attribute within the same Write-Back granule as bytes written with the Write-Back cacheable attribute might have their values reverted to the old values as a result of cache Write-Back.

The loss of properties associated with mismatched memory type attributes refers only to the following properties of Device memory that are additional to the properties of Normal memory:

- Prohibition of speculative read accesses.
- Prohibition on Gathering.
- Prohibition on Reordering.

For the following situations, when a physical memory Location is accessed with mismatched attributes, a more restrictive set of behaviors applies. The description of each situation also describes the behaviors that apply:

1. Any agent that reads that memory Location using the same common definition of the Memory type, Shareability and Cacheability attributes is guaranteed to access it coherently, to the extent required by that common definition of the memory attributes, only if all the following conditions are met:
   - All writes are performed to an alias of the memory Location that uses the same definition of the Memory type, Shareability and Cacheability attributes.
   - Either:
     - All writes are performed to an alias of the memory Location that has Inner Cacheability and Outer Cacheability attributes both as Non-cacheable.
     - All aliases to a memory Location use a definition of the Shareability attributes that encompasses all the agents with permission to access the Location.
2. The possible software-visible effects caused by mismatched attributes for a memory Location are defined more precisely if all of the mismatched attributes define the memory Location as one of:
   - Any Device memory type.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- Normal Inner Non-cacheable, Outer Non-cacheable memory.

In these cases, the only permitted software-visible effects of the mismatched attributes are one or more of the following:

- Possible loss of properties derived from the memory type when multiple agents attempt to access the memory Location.
- Possible reordering of memory transactions to the same memory Location with different memory attributes, potentially leading to a loss of coherency or uniprocessor semantics. Any possible loss of coherency or uniprocessor semantics can be avoided by inserting `DMB` barrier instructions between accesses to the same memory Location that might use different attributes.

Where there is a loss of the uniprocessor semantics, ordering, or coherency, the following approaches can be used:

1. If the mismatched attributes for a memory Location all assign the same Shareability attribute to a Location that has a cacheable attribute, any loss of uniprocessor semantics, ordering, or coherency within a Shareability domain can be avoided by use of software cache management. To do so, software must use the techniques that are required for the software management of the ordering or coherency of cacheable Locations between agents in different shareability domains. This means:
   - Before writing to a cacheable Location not using the Write-Back attribute, software must invalidate, or clean, a Location from the caches if any agent might have written to the Location with the Write-Back attribute. This avoids the possibility of overwriting the Location with stale data.
   - After writing to a cacheable Location with the Write-Back attribute, software must clean the Location from the caches, to make the write visible to external memory.
   - Before reading the Location with a cacheable attribute, software must invalidate, or clean and invalidate, the Location from the caches, to ensure that any value held in the caches reflects the last value made visible in external memory.
   - Executing a `DMB` barrier instruction, with scope that applies to the common Shareability of the accesses, between any accesses to the same cacheable Location that use different attributes.

   Note

   In AArch32 state, cache maintenance instructions can be accessed only from an Exception level that is higher than EL0, and therefore require a system call. For information on system calls, see Exception-generating and exception-handling instructions. For information about the AArch32 cache maintenance instructions, see AArch32 cache and branch predictor support.

   In all cases:

   - Location refers to any byte within the current coherency granule.
   - A clean and invalidate instruction can be used instead of a clean instruction, or instead of an invalidate instruction.
   - In the sequences outlined in this section, all cache maintenance instructions and memory transactions must be completed, or ordered by the use of barrier operations, if they are not naturally ordered by the use of a common address, see Ordering of cache and branch predictor maintenance instructions.

   Note

   With software management of coherency, race conditions can cause loss of data. A race condition occurs when different agents write simultaneously to bytes that are in the same Location, and the invalidate, write, clean sequence of one agent overlaps with the equivalent sequence of another agent. A race condition also occurs if the first operation of either sequence is a clean, rather than an invalidate.

2. If the mismatched attributes for a Location mean that multiple cacheable accesses to the Location might be made with different Shareability attributes, then ordering and coherency are guaranteed only if:
   - Software running on a PE cleans and invalidates a Location from cache before and after each read or write to that Location by that PE.
   - A `DMB` barrier with scope that covers the full Shareability of the accesses is placed between any accesses to the same memory Location that use different attributes.

   Note

   The Note in rule 1 of this list, about possible race conditions, also applies to this rule.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

In addition, if multiple agents attempt to use Load-Exclusive or Store-Exclusive instructions to access a Location, and the accesses from the different agents have different memory attributes associated with the Location, the Exclusives monitor state becomes UNKNOWN.

Arm strongly recommends that software does not use mismatched attributes for aliases of the same Location. An implementation might not optimize the performance of a system that uses mismatched aliases.

Note

As described in Non-cacheable accesses and instruction caches, a non-cacheable access is permitted to be cached in an instruction cache, despite the fact that a non-cacheable access is not permitted to be cached in a unified cache. Despite this, when cacheable and non-cacheable aliases exist for memory which is executable, these must be treated as mismatched aliases to avoid coherency issues from the data or unified caches that might hold entries that will be brought into the instruction caches.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E2.10 Synchronization and semaphores

The architecture provides non-blocking synchronization of shared memory, using *synchronization primitives*. The information in this section about memory accesses by synchronization primitives applies to accesses to both Normal and Device memory.

Note

Use of the synchronization primitives scales for multiprocessing system designs.

Table E2-6 shows the synchronization primitives and the associated `CLREX` instruction.

Table E2-6 Synchronization primitives and associated instruction, T32 and A32 instruction sets

Transaction size	Additional semantics	Load-Exclusive[a]	Store-Exclusive[a]	Other[a]
Byte	-	`LDREXB`	`STREXB`	-
	Load-Acquire/Store-Release	`LDAEXB`	`STLEXB`	-
Halfword	-	`LDREXH`	`STREXH`	-
	Load-Acquire/Store-Release	`LDAEXH`	`STLEXH`	-
Word	-	`LDREX`	`STREX`	-
	Load-Acquire/Store-Release	`LDAEX`	`STLEX`	-
Doubleword	-	`LDREXD`	`STREXD`	-
	Load-Acquire/Store-Release	`LDAEXD`	`STLEXD`	-
None	Clear-Exclusive	-	-	`CLREX`

a. Instruction in the T32 and A32 instruction sets.

Except for the row showing the `CLREX` instruction, the two instructions in a single row are a Load-Exclusive/Store-Exclusive instruction pair. The model for the use of a Load-Exclusive/Store-Exclusive instruction pair accessing a non-aborting memory address *x* is:

- The Load-Exclusive instruction reads a value from memory address *x*.
- The corresponding Store-Exclusive instruction succeeds in writing back to memory address *x* only if no other observer, process, or thread has performed a more recent store to address *x*. The Store-Exclusive instruction returns a status bit that indicates whether the memory write succeeded.

A Load-Exclusive instruction marks a small block of memory for exclusive access. The size of the marked block is IMPLEMENTATION DEFINED, see Marking and the size of the marked memory block. A Store-Exclusive instruction to any address in the marked block clears the marking.

Note

In this section, the term PE includes any observer that can generate a Load-Exclusive or a Store-Exclusive instruction.

The following sections give more information:

- Exclusive access instructions and Non-shareable memory locations.
- Exclusive access instructions and shareable memory locations.
- Marking and the size of the marked memory block.
- Context switch support.
- Load-Exclusive and Store-Exclusive instruction usage restrictions.
- Use of WFE and SEV instructions by spin-locks.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

E2.10.1 Exclusive access instructions and Non-shareable memory locations

For memory locations for which the Shareability attribute is Non-shareable, the exclusive access instructions rely on a *local Exclusives monitor*, or *local monitor*, that marks any address from which the PE executes a Load-Exclusive instruction. Any non-aborted attempt by the same PE to use a Store-Exclusive instruction to modify any address is guaranteed to clear the marking.

A Load-Exclusive instruction performs a load from memory, and:

- The executing PE marks the physical memory address for exclusive access.
- The local monitor of the executing PE transitions to the Exclusive Access state.

A Store-Exclusive instruction performs a conditional store to memory that depends on the state of the local monitor:

If the local monitor is in the Exclusive Access state

- If the address of the Store-Exclusive instruction is the same as the address that has been marked in the monitor by an earlier Load-Exclusive instruction, then the store occurs. Otherwise, it is IMPLEMENTATION DEFINED whether the store occurs.
- A status value is returned to a register:
  - If the store took place the status value is 0.
  - Otherwise, the status value is 1.
- The local monitor of the executing PE transitions to the Open Access state.

When an Exclusives monitor is in the Exclusive Access state the monitor is *set*.

If the local monitor is in the Open Access state

- No store takes place.
- A status value of 1 is returned to a register.
- The local monitor remains in the Open Access state.

When an Exclusives monitor is in the Exclusive Access state the monitor is *clear*.

The Store-Exclusive instruction defines the register to which the status value is returned.

When a PE writes using any instruction other than a Store-Exclusive instruction:

- If the write is to a PA that is not marked as Exclusive Access by its local monitor and that local monitor is in the Exclusive Access state it is IMPLEMENTATION DEFINED whether the write affects the state of the local monitor.
- If the write is to a PA that is marked as Exclusive Access by its local monitor it is IMPLEMENTATION DEFINED whether the write affects the state of the local monitor.

It is IMPLEMENTATION DEFINED whether a store to a marked PA causes a mark in the local monitor to be cleared if that store is by an observer other than the one that caused the PA to be marked.

Figure E2-4 shows the state machine for the local monitor and the effect of each of the operations shown in the figure.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Operations marked * are possible alternative IMPLEMENTATION DEFINED options.
In the diagram: `LoadExcl` represents any Load-Exclusive instruction
`StoreExcl` represents any Store-Exclusive instruction
`Store` represents any other store instruction.

Any `LoadExcl` operation updates the marked address to the most significant bits of the address `x` used for the operation.

Figure E2-4 Local monitor state machine diagram

For more information about marking, see Marking and the size of the marked memory block.

Note

For the local monitor state machine, as shown in Figure E2-4:

- The IMPLEMENTATION DEFINED options for the local monitor are consistent with the local monitor being constructed so that it does not hold any PA, but instead treats any access as matching the address of the previous Load-Exclusive instruction.
- A local monitor implementation can be unaware of Load-Exclusive and Store-Exclusive instructions from other PEs.
- The architecture does not require a load instruction, by another PE, that is not a Load-Exclusive instruction, to have any effect on the local monitor.
- It is IMPLEMENTATION DEFINED whether the transition from Exclusive Access to Open Access state occurs when the `Store` or `StoreExcl` is from another observer.

E2.10.1.1 Changes to the local monitor state resulting from speculative execution

The architecture permits a local monitor to transition to the Open Access state as a result of speculation, or from some other cause. This is in addition to the transitions to Open Access state caused by the architectural execution of an operation shown in Figure E2-4.

An implementation must ensure that:

- The local monitor cannot be seen to transition to the Exclusive Access state except as a result of the architectural execution of one of the operations shown in Figure E2-4.
- Any transition of the local monitor to the Open Access state not caused by the architectural execution of an operation shown in Figure E2-4 must not indefinitely delay forward progress of execution.

E2.10.2 Exclusive access instructions and shareable memory locations

In the context of this section, a shareable memory location is a memory location that has, or is treated as if it has, a Shareability attribute of Inner Shareable or Outer Shareable.

For shareable memory locations, exclusive access instructions rely on:

- A *local monitor* for each PE in the system, which marks any address from which the PE executes a Load-Exclusive. The local monitor operates as described in Exclusive access instructions and Non-shareable

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

memory locations, except that for shareable memory any Store-Exclusive is then subject to checking by the global monitor if it is described in that section as doing at least one of the following:

— Updating memory.
— Returning a status value of 0.

The local monitor can ignore accesses from other PEs in the system.

- A *global monitor* that marks a PA as exclusive access for a particular PE. This marking is used later to determine whether a Store-Exclusive to that address that has not been failed by the local monitor can occur. Any successful write to the marked block by any other observer in the Shareability domain of the memory location is guaranteed to clear the marking. For each PE in the system, the global monitor:

  — Can hold at least one marked block.
  — Maintains a state machine for each marked block it can hold.

  Note

  For each PE, the architecture only requires global monitor support for a single marked address. Any situation that might benefit from the use of multiple marked addresses on a single PE is CONSTRAINED UNPREDICTABLE, see Load-Exclusive and Store-Exclusive instruction usage restrictions.

Note

The global monitor can either reside in a block that is part of the hardware on which the PE executes or exist as a secondary monitor at the memory interfaces. The IMPLEMENTATION DEFINED aspects of the monitors mean that the global monitor and local monitor can be combined into a single unit, provided that the unit performs the global monitor and local monitor functions defined in this manual.

For shareable memory locations, in some implementations and for some memory types, the properties of the global monitor require functionality outside the PE. Some system implementations might not implement this functionality for all locations of memory. In particular, this can apply to:

- Any type of memory in the system implementation that does not support hardware cache coherency.
- Non-cacheable memory, or memory treated as Non-cacheable, in an implementation that does support hardware cache coherency.

In such a system, it is defined by the system:

- Whether the global monitor is implemented.
- If the global monitor is implemented, which address ranges or memory types it monitors.

Note

To support the use of the Load-Exclusive/Store-Exclusive mechanism when address translation is disabled, a system might define at least one location of memory, of at least the size of the translation granule, in the system memory map to support the global monitor for all PEs within a common Inner Shareable domain. However, this is not an architectural requirement. Therefore, architecturally-compliant software that requires mutual exclusion must not rely on using the Load-Exclusive/Store-Exclusive mechanism, and must instead use a software algorithm such as Lamport's Bakery algorithm to achieve mutual exclusion.

Because implementations can choose which memory types are treated as Non-cacheable, the only memory types for which it is architecturally guaranteed that a global Exclusives monitor is implemented are:

- Inner Shareable, Inner Write-Back, Outer Write-Back Normal memory.
- Outer Shareable, Inner Write-Back, Outer Write-Back Normal memory.

If the global monitor is not implemented for an address range or memory type, then performing a Load-Exclusive/Store-Exclusive instruction to such a location has one or more of the following effects:

- The instruction generates an External abort.
- The instruction generates an IMPLEMENTATION DEFINED MMU fault. This is reported using the Fault status code of:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

— DFSR.STATUS = `0b110101` when using the Long-descriptor translation table format. The fault can also be reported in the HSR.ISS[5:0] field for exceptions to Hyp mode.

— DFSR.FS = `0b10101` when using the Short-descriptor translation table format.

If the IMPLEMENTATION DEFINED MMU fault is generated for the Non-secure PL1&0 translation regime then:

— If the fault is generated because of the memory type defined in the first stage of translation, or if the second stage of translation is disabled, then this is a first stage fault and the exception is taken to EL1.

— Otherwise, the fault is a second stage fault and the exception is taken to EL2.

The priority of this fault is IMPLEMENTATION DEFINED.

- The instruction is treated as a `NOP`.
- The Load-Exclusive instruction is treated as if it were accessing a Non-shareable location, but the state of the local monitor becomes UNKNOWN.
- The Store-Exclusive instruction is treated as if it were accessing a Non-shareable location, but the state of the local monitor becomes UNKNOWN.
- The value held in the result register of the Store-Exclusive instruction becomes UNKNOWN.

In addition, for write transactions generated by non-PE observers that do not implement exclusive accesses or other atomic access mechanisms, the effect that writes have on the global and local monitors used by an Arm PE is IMPLEMENTATION DEFINED. The writes might not clear the global monitors of other PEs for:

- Some address ranges.
- Some memory types.

E2.10.2.1 Operation of the global Exclusives monitor

A Load-Exclusive instruction from shareable memory performs a load from memory, and causes the PA of the access to be marked as exclusive access for the requesting PE. This access can also cause the exclusive access mark to be removed from any other PA that has been marked by the requesting PE.

Note

The global monitor only supports a single outstanding exclusive access to shareable memory for each PE.

A Load-Exclusive instruction by one PE has no effect on the global monitor state for any other PE.

A Store-Exclusive instruction performs a conditional store to memory:

- The store is guaranteed to succeed only if the PA accessed is marked as exclusive access for the requesting PE and both the local monitor and the global monitor state machines for the requesting PE are in the Exclusive Access state. In this case:
  — A status value of 0 is returned to a register to acknowledge the successful store.
  — The final state of the global monitor state machine for the requesting PE is IMPLEMENTATION DEFINED.
  — If the address accessed is marked for exclusive access in the global monitor state machine for any other PE then that state machine transitions to Open Access state.
- If no address is marked as exclusive access for the requesting PE, the store does not succeed:
  — A status value of 1 is returned to a register to indicate that the store failed.
  — The global monitor is not affected and remains in Open Access state for the requesting PE.
- If a different PA is marked as exclusive access for the requesting PE, it is IMPLEMENTATION DEFINED whether the store succeeds or not:
  — If the store succeeds a status value of 0 is returned to a register, otherwise a value of 1 is returned.
  — If the global monitor state machine for the PE was in the Exclusive Access state before the Store-Exclusive instruction it is IMPLEMENTATION DEFINED whether that state machine transitions to the Open Access state.

The Store-Exclusive instruction defines the register to which the status value is returned.

In a shared memory system, the global monitor implements a separate state machine for each PE in the system. The state machine for accesses to shareable memory by PE(n) can respond to all the shareable memory accesses visible to it. This means it responds to:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- Accesses generated by PE(n).
- Accesses generated by the other observers in the Shareability domain of the memory location. These accesses are identified as (!n).

In a shared memory system, the global monitor implements a separate state machine for each observer that can generate a Load-Exclusive or a Store-Exclusive instruction in the system.

A global monitor:

- In the Exclusive Access state is *set*.
- In the Open Access state is *clear*.

E2.10.2.1.1 Clear global monitor event

Whenever the global monitor state for a PE changes from Exclusive access to Open access, an event is generated and held in the Event register for that PE. This register is used by the Wait for Event mechanism, see Wait For Event and Send Event.

Figure E2-5 shows the state machine for PE(n) in a global monitor.

‡`StoreExcl`(Marked_address,!n) clears the monitor only if the `StoreExcl` updates memory
Operations marked * are possible alternative IMPLEMENTATION DEFINED options.
In the diagram: `LoadExcl` represents any Load-Exclusive instruction
`StoreExcl` represents any Store-Exclusive instruction
`Store` represents any other store instruction.

Any `LoadExcl` operation updates the marked address to the most significant bits of the address x used for the operation.

Figure E2-5 Global monitor state machine diagram for PE(n) in a multiprocessor system

For more information about marking, see Marking and the size of the marked memory block.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Note

For the global monitor state machine, as shown in Figure E2-5:

- The architecture does not require a load instruction by another PE, that is not a Load-Exclusive instruction, to have any effect on the global monitor.
- Whether a Store-Exclusive instruction successfully updates memory or not depends on whether the address accessed matches the marked shareable memory address for the PE issuing the Store-Exclusive instruction, and whether the local and global monitors are in the exclusive state. For this reason, Figure E2-5 shows only how the operations by (!n) cause state transitions of the state machine for PE(n).
- A Load-Exclusive instruction can update only the marked shareable memory address for the PE issuing the Load-Exclusive instruction.
- When the global monitor is in the Exclusive Access state, it is IMPLEMENTATION DEFINED whether a `CLREX` instruction causes the global monitor to transition from Exclusive Access to Open Access state.
- It is IMPLEMENTATION DEFINED:
  - Whether a modification to a Non-shareable memory location can cause a global monitor to transition from Exclusive Access to Open Access state.
  - Whether a Load-Exclusive instruction to a Non-shareable memory location can cause a global monitor to transition from Open Access to Exclusive Access state.

E2.10.3 Marking and the size of the marked memory block

When a Load-Exclusive instruction is executed, the resulting marked block ignores the least significant bits of the 64-bit memory address.

When a Load-Exclusive instruction is executed, a marked block of size 2^a bytes is created by ignoring the least significant bits of the memory address. A marked address is any address within this marked block. The size of the marked memory block is called the *Exclusives reservation granule*. The Exclusives reservation granule is IMPLEMENTATION DEFINED in the range 4 - 512 words.

Note

This definition means that the Exclusives reservation granule is:

- 4 words in an implementation where *a* is 4.
- 512 words in an implementation where *a* is 11.

For example, in an implementation where *a* is 4, a successful `LDREXB` of address `0x341B4` defines a marked block using bits[47:4] of the address. This means that the four words of memory from `0x341B0` to `0x341BF` are marked for exclusive access.

In some implementations the CTR identifies the Exclusives reservation granule, see CTR. Otherwise, software must assume that the maximum Exclusives reservation granule, 512 words, is implemented.

E2.10.4 Context switch support

An exception return clears the local monitor. As a result, performing a `CLREX` instruction as part of a context switch is not required in most situations.

Note

Context switching is not an application level operation. However, this information is included here to complete the description of the exclusive operations.

E2.10.5 Load-Exclusive and Store-Exclusive instruction usage restrictions

The Load-Exclusive and Store-Exclusive instructions are intended to work together as a pair, for example a `LDREX/STREX` pair or a `LDREXB/STREXB` pair, executed as part of a loop that contains only a single Load-Exclusive and

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Store-Exclusive pair. To support different implementations of these functions, software must follow the notes and restrictions given in this subsection.

The following notes describe use of a Load-Exclusive/ Store-Exclusive instruction pair, `LoadExcl`/`StoreExcl`, to indicate the use of any of the Load-Exclusive/Store-Exclusive instruction pairs shown in Table E2-6. In this context, a `LoadExcl`/`StoreExcl` pair comprises two instructions in the same thread of execution:

- The exclusives support a single outstanding exclusive access for each PE thread that is executed. The architecture makes use of this by not requiring an address or size check as part of the `IsExclusiveLocal()` function. If the target VA of a `StoreExcl` is different from the VA of the preceding `LoadExcl` instruction in the same thread of execution, behavior can be CONSTRAINED UNPREDICTABLE with the following behavior:
  - The `StoreExcl` either passes or fails, the status value returned by the `StoreExcl` is UNKNOWN, and the states of the local and global monitors for that PE are UNKNOWN.

    Note

    This means the `StoreExcl` might pass for some instances of a `LoadExcl`/`StoreExcl` pair with mismatched addresses, and fail for other instances of a `LoadExcl`/`StoreExcl` pair with mismatched addresses.

  - The data at the address accessed by the `LoadExcl`, and at the address accessed by the `StoreExcl`, is UNKNOWN.

  This means software can rely on a `LoadExcl`/`StoreExcl` pair to eventually succeed only if the `LoadExcl` and the `StoreExcl` are executed with the same VA.
- An implementation of the Load-Exclusive and Store-Exclusive instructions can require that, in any thread of execution, the transaction size of a `StoreExcl` instruction is the same as the transaction size of the preceding `LoadExcl` instruction executed in that thread. If the transaction size of a `StoreExcl` instruction is different from the preceding `LoadExcl` instruction in the same thread of execution, behavior can be CONSTRAINED UNPREDICTABLE with the following behavior:
  - The `StoreExcl` either passes or fails, and the status value returned by the `StoreExcl` is UNKNOWN.

    Note

    This means the `StoreExcl` might pass for some instances of a `LoadExcl`/`StoreExcl` pair with mismatched transaction sizes, and fail for other instances of a `LoadExcl`/`StoreExcl` pair with mismatched transaction sizes.

  - The block of data of the size of the larger of the transaction sizes used by the `LoadExcl`/`StoreExcl` pair at the address accessed by the `LoadExcl`/`StoreExcl` pair, is UNKNOWN.

  This means software can rely on a `LoadExcl`/`StoreExcl` pair to eventually succeed only if the `LoadExcl` and the `StoreExcl` have the same transaction size.
- If FEAT_LSE is not implemented, `LoadExcl`/`StoreExcl` loops are guaranteed to make forward progress only if, for any `LoadExcl`/`StoreExcl` loop within a single thread of execution, the software meets all of the following conditions. If FEAT_LSE is implemented, software can maximize the likelihood that `LoadExcl`/`StoreExcl` loops make forward progress, for any `LoadExcl`/`StoreExcl` loop within a single thread of execution, by meeting all of the following conditions:

  1 Between the Load-Exclusive and the Store-Exclusive, there are no explicit memory effects, software prefetches, direct or indirect System register writes, address translation instructions, cache or TLB maintenance instructions, exception generating instructions, exception returns, `ISB` barriers, indirect branches, or Branch with Link instructions.

  2 Between the Store-Exclusive returning a failing result and the retry of the corresponding Load-Exclusive:
  - There are no stores or `PLDW` instructions to any address within the Exclusives reservation granule accessed by the Store-Exclusive. This includes to VA aliases to those addresses.
  - There are no loads or prefetches to any address within the Exclusives reservation granule accessed by the Store-Exclusive that use a different VA alias to that address.
  - There are no other Store-Exclusive instructions to any other address.
  - There are no direct or indirect System register writes, other than changes to the flag fields in the CPSR or FPSCR, caused by data processing or comparison instructions.
  - There are no direct or indirect address translation instructions, cache or TLB maintenance instructions, exception generating instructions, exception returns, indirect branches, or Branch with Link instructions.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

— All loads and stores are to a block of contiguous virtual memory of not more than 512 bytes in size.

The Exclusives monitor can be cleared at any time without an application-related cause, provided that such clearing is not systematically repeated so as to prevent the forward progress in finite time of at least one of the threads that is accessing the Exclusives monitor. However, it is permissible for the `LoadExcl`/`StoreExcl` loop not to make forward progress if a different thread is repeatedly doing any of the following in a tight loop:

— Performing stores to a PA covered by the Exclusives monitor.

— Prefetching with intent to write to a PA covered by the Exclusives monitor.

— Executing data cache clean, data cache invalidate, or data cache clean and invalidate instructions to a PA covered by the Exclusives monitor.

— Executing instruction cache invalidate all instructions.

— Executing instruction cache invalidate by VA instructions to a PA covered by the Exclusives monitor.

- Implementations can benefit from keeping the `LoadExcl` and `StoreExcl` operations close together in a single thread of execution. This minimizes the likelihood of the Exclusives monitor state being cleared between the `LoadExcl` instruction and the `StoreExcl` instruction. Therefore, for best performance, Arm strongly recommends a limit of 128 bytes between `LoadExcl` and `StoreExcl` instructions in a single thread of execution.
- The architecture sets an upper limit of 2048 bytes on the Exclusives reservation granule that can be marked as exclusive. For performance reasons, Arm recommends that objects that are accessed by exclusive accesses are separated by the size of the Exclusives reservation granule. This is a performance guideline rather than a functional requirement.
- After taking a Data Abort exception, the state of the Exclusives monitors is UNKNOWN.
- For the memory location accessed by a `LoadExcl`/`StoreExcl` pair, if the memory attributes for a `StoreExcl` instruction are different from the memory attributes for the preceding `LoadExcl` instruction in the same thread of execution, behavior is CONSTRAINED UNPREDICTABLE. Where this occurs because the translation of the accessed address changes between the `LoadExcl` instruction and the `StoreExcl` instruction, the CONSTRAINED UNPREDICTABLE behavior is as follows:

  — The `StoreExcl` either passes or fails, and the status value returned by the `StoreExcl` is UNKNOWN.

  Note

  This means the `StoreExcl` might pass for some instances of a `LoadExcl`/`StoreExcl` pair with changed memory attributes, and fail for other instances of a `LoadExcl`/`StoreExcl` pair with changed memory attributes.

  — The data at the address accessed by the `StoreExcl` is UNKNOWN.

  Note

  Another bullet point in this list covers the case where the memory attributes of a `LoadExcl`/`StoreExcl` pair differ as a result of using different VAs with different attributes that point to the same PA.

- The effect of a data or unified cache invalidate, clean, or clean and invalidate instruction on a local or global Exclusives monitor that is in the Exclusive Access state is CONSTRAINED UNPREDICTABLE, and the instruction might clear the monitor, or it might leave it in the Exclusive Access state. For address-based maintenance instructions, this also applies to the monitors of other PEs in the same Shareability domain as the PE executing the cache maintenance instruction, as determined by the Shareability domain of the address being maintained.

  Note

  Arm strongly recommends that implementations ensure that the use of such maintenance instructions by a PE in the Non-secure state cannot cause a denial of service on a PE in the Secure state.

- If the mapping of the VA to PA is changed between the `LoadExcl` instruction and the `StoreExcl` instruction, and the change is performed using a break-before-make sequence as described in Using break-before-make when updating translation table entries, if the `StoreExcl` is performed after another write to the same PA as the `StoreExcl`, and that other write was performed after the old translation was properly invalidated and that invalidation was properly synchronized, then the `StoreExcl` will not pass its monitor check.

  Note

  Arm expects that, in many implementations, either:

  — The TLB invalidation will clear either the local or global monitor.

  — The PA will be checked between the `LoadExcl` and `StoreExcl`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- The Exclusive Access state for an address accessed by a PE can be lost as a result of a `PLDW` instruction to the same PA executed by another PE. This means that a very high rate of repeated `PLDW` accesses to a memory location might impede the forward progress of another PE.

Note

In the event of repeatedly-contending `LoadExcl`/`StoreExcl` instruction sequences from multiple PEs, an implementation must ensure that forward progress is made by at least one PE.

E2.10.6 Use of WFE and SEV instructions by spin-locks

The architecture provides Wait For Event, Send Event, and Send Event Local instructions, `WFE`, `SEV`, `SEVL`, that can assist with reducing power consumption and bus contention caused by PEs repeatedly attempting to obtain a spin-lock. These instructions can be used at the application level, but a complete understanding of what they do depends on a system level understanding of exceptions. They are described in Wait For Event and Send Event. However, when the global monitor for a PE changes from Exclusive Access state to Open Access state, an event is generated.

Note

This is equivalent to issuing an `SEVL` instruction on the PE for which the monitor state has changed. It removes the need for spinlock code to include an `SEV` instruction after clearing a spinlock.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Part F

The AArch32 Instruction Sets

Chapter F1
About the T32 and A32 Instruction Descriptions

This chapter describes each instruction. It contains the following sections:

- Format of instruction descriptions.
- Standard assembler syntax fields.
- Conditional execution.
- Shifts applied to a register.
- Memory accesses.
- Encoding of lists of general-purpose registers and the PC.
- General information about the T32 and A32 instruction descriptions.
- Additional pseudocode support for instruction descriptions.
- Additional information about Advanced SIMD and floating-point instructions.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F1.1 Format of instruction descriptions

The instruction descriptions in T32 and A32 Base Instruction Set Instruction Descriptions and T32 and A32 Advanced SIMD and Floating-point Instruction Descriptions normally use the following format:

- Instruction section title.
- Introduction to the instruction.
- A description of each encoding of the instruction.
- Assembler syntax.
- Pseudocode describing how the instruction operates.
- Notes, if applicable.

Each of these items is described in more detail in the following subsections.

F1.1.1 Instruction section title

The instruction section title gives the base mnemonic for the instruction or instructions described in the section. When one mnemonic has multiple forms described in separate instruction sections, this is followed by a short description of the form in parentheses. The most common use of this is to distinguish between forms of an instruction in which one of the operands is an immediate value and forms in which it is a register.

F1.1.2 Introduction to the instruction

The introduction to the instruction briefly describes the main features of the instruction. This description is not necessarily complete and is not definitive. If there is any conflict between it and the more detailed information that follows, the latter takes priority.

F1.1.3 Instruction encodings

This is a list of one or more instruction encodings. Each instruction encoding is labeled as:

- A1, A2, A3 . . . for the first, second, third, and any additional A32 encodings.
- T1, T2, T3 . . . for the first, second, third, and any additional T32 encodings.

Each instruction encoding description consists of:

- An assembly syntax that ensures that the assembler selects the encoding in preference to any other encoding. Sometimes, multiple syntax variants are given. These are written in a `typewriter` font using the conventions described in Assembler syntax prototype line conventions. The correct one to use can be indicated by:
  - — A subheading that identifies the encodings that correspond to the syntax. See, for example, the subheading *Flag setting, rotate right with extend variant* in the description of the A1 encoding of the `ADC`, `ADCS` (register) instructions in A1.
  - — An annotation to the syntax, such as *Inside IT block* or *Outside IT block*. See, for example, the syntax descriptions of the T1 encoding of the `ADC`, `ADCS` (register) instructions in T1.

  In other cases, the correct one to use can be determined by looking at the assembler syntax description and using it to determine which syntax corresponds to the instruction being disassembled.

  There is usually more than one syntax variant that ensures re-assembly to any particular encoding, and the exact set of syntaxes that do so usually depends on the register numbers, immediate constants, and other operands to the instruction. For example, when assembling to the T32 instruction set, the syntax `AND R0, R0, R8` ensures selection of a 32-bit encoding but `AND R0, R0, R1` selects a 16-bit encoding.

  For each instruction encoding belonging to a target instruction set, an assembler can use this information to determine whether it can use that encoding to encode the instruction requested by the UAL source. If multiple encodings can encode the instruction, then:

  - — If both a 16-bit encoding and a 32-bit encoding can encode the instruction, the architecture *prefers* the 16-bit encoding. This means the assembler must use the 16-bit encoding rather than the 32-bit encoding. Software can use the `.W` and `.N` qualifiers to specify the required encoding width, see Standard assembler syntax fields.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

— If multiple encodings of the same length can encode the instruction, the *Assembler syntax* subsection says which encoding is preferred, and how software can, instead, select the other encodings.
Each encoding also documents UAL syntax that selects it in preference to any other encoding.

If no encodings of the target instruction set can encode the instruction requested by the UAL source, normally the assembler generates an error saying that the instruction is not available in that instruction set.

Note

In some cases, an instruction is available in one instruction set but not in another. The *Assembler syntax* subsection identifies many of these cases. For example, the A32 instructions with bits<31:28> == `0b1111` described in Branch, branch with link, and block data transfer, System register access, Advanced SIMD, floating-point, and Supervisor call, and Unconditional instructions cannot have a Condition code, but the equivalent T32 instructions often can, and this usually appears in the *Assembler syntax* subsection as a statement that the A32 instruction cannot be conditional.
However, some such cases are too complex to describe in the available space, so the definitive test of whether an instruction is available in a given instruction set is whether there is an available encoding for it in that instruction set.

The assembly syntax given for an encoding is therefore a suitable one for a disassembler to disassemble that encoding to. However, disassemblers might use simpler syntaxes when they are suitable for the operand combination, to produce more readable disassembled code.

- An encoding diagram, where:
  — For a 32-bit A32 encoding diagram, the bits are numbered from 31-0.
  — For a 16-bit T32 encoding diagram, the bits are numbered from 15-0.
  This halfword can be described as `hw1` of the instruction.
  — For a 32-bit T32 encoding diagram, the bits are numbered from 15-0 for each halfword, as a reminder that a 32-bit T32 instruction consists of two consecutive halfwords rather than a word.
  In this case, the left-hand halfword in the diagram is identified as `hw1`, and the right-hand halfword is identified as `hw2`.

  Where instructions are stored using the standard little-endian instruction endianness:
  — The encoding diagram for an A32 instruction at address A shows, from left to right, the bytes at addresses A+3, A+2, A+1, A.
  — The encoding diagram for a 32-bit T32 instruction shows bytes in the order A+1, A for `hw1`, followed by bytes A+3, A+2 for `hw2`.
- Encoding-specific pseudocode. This is pseudocode that translates the encoding-specific instruction fields into inputs to the encoding-independent pseudocode in the *Operation* subsection, and that picks out any special cases in the encoding. For a detailed description of the pseudocode used and of the relationship between the encoding diagram, the encoding-specific pseudocode and the encoding-independent pseudocode, see Arm Pseudocode Definition.

F1.1.4 Assembler symbols

The *Assembly symbols* describe the standard UAL syntax for the instruction.

Each syntax description consists of the following elements:

- Descriptions of all variable or optional fields of the syntax.
  Some syntax fields are standardized across all or most instructions. Standard assembler syntax fields describes these fields.
  By default, syntax fields that specify registers, such as `<Rd>`, `<Rn>`, or `<Rt>`, can be any of R0-R12 or LR in T32 instructions, and any of R0-R12, SP, or LR in A32 instructions. These require that the encoding-specific pseudocode set the corresponding integer variable (such as `d`, `n`, or `t`) to the corresponding register number, using 0-12 for R0-R12, 13 for SP, or 14 for LR:
  — Normally, software can do this by setting the corresponding field in the instruction, typically named Rd, Rn, Rt, to the binary encoding of that number.
  — In the case of 16-bit T32 encodings, the field is normally of length 3, and so the encoding is available only when the assembler syntax specifies one of R0-R7. Such encodings often use a register field name like Rdn. This indicates that the encoding is available only if `<Rd>` and `<Rn>` specify the same register, and that the register number of that register is encoded in the field if they do.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The description of a syntax field that specifies a register sometimes extends or restricts the permitted range of registers or documents other differences from the default rules for such fields. Examples of extensions are permitting the use of the SP in a T32 instruction, or permitting the use of the PC, identified using register number 15.

- Where appropriate, text that briefly describes changes from the pre-UAL assembler syntax. Where present, this usually consists of an alternative pre-UAL form of the assembler mnemonic. The pre-UAL assembler syntax does not conflict with UAL. Arm recommends that it is supported, as an optional extension to UAL, so that pre-UAL assembler source files can be assembled.

F1.1.4.1 Assembler syntax prototype line conventions

The following conventions are used in assembler syntax prototype lines and their subfields:

`< >`

Any item bracketed by < and > is a short description of a type of value to be supplied by the user in that position. A longer description of the item is normally supplied by subsequent text. Such items often correspond to a similarly named field in an encoding diagram for an instruction. When the correspondence requires only the binary encoding of an integer value or register number to be substituted into the instruction encoding, it is not described explicitly. For example, if the assembler syntax for an instruction contains an item `<Rn>` and the instruction encoding diagram contains a 4-bit field named Rn, the number of the register specified in the assembler syntax is encoded in binary in the instruction field.

If the correspondence between the assembler syntax item and the instruction encoding is more complex than simple binary encoding of an integer or register number, the item description indicates how it is encoded. This is often done by specifying a required output from the encoding-specific pseudocode, such as `add = TRUE`. The assembler must use only encodings that produce that output.

`{ }`

Any item bracketed by { and } is optional. A description of the item and of how its presence or absence is encoded in the instruction is normally supplied by subsequent text.

Many instructions have an optional destination register. Unless otherwise stated, if such a destination register is omitted, it is the same as the immediately following source register in the instruction syntax.

`#`

In the assembler syntax, numeric constants are normally preceded by a #. Some UAL instruction syntax descriptions explicitly show this # as optional. Any UAL assembler:

- Must treat the # as optional where an instruction syntax description shows it as optional.
- Can treat the # either as mandatory or as optional where an instruction syntax description does not show it as optional.

Note

Arm recommends that UAL assemblers treat all uses of # shown in this manual as optional.

spaces

Single spaces are used for clarity, to separate items. When a space is obligatory in the assembler syntax, two or more consecutive spaces are used.

`+/-`

This indicates an optional + or - sign. If neither is coded, + is assumed.

All other characters must be encoded precisely as they appear in the assembler syntax. Apart from { and }, the special characters described above do not appear in the basic forms of assembler instructions documented in this manual. In a few places, the { and } characters must be encoded as part of a variable item. When this happens, the long description of the variable item indicates how they must be used.

F1.1.5 Pseudocode describing how the instruction operates

The *Operation for all classes* subsection contains encoding-independent pseudocode that describes the main operation of the instruction. For a detailed description of the pseudocode used and of the relationship between the encoding diagram, the encoding-specific pseudocode and the encoding-independent pseudocode, see Arm Pseudocode Definition.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F1.2 Standard assembler syntax fields

The following assembler syntax fields are standard across all or most instructions:

`<c>` Is an optional field. It specifies the condition under which the instruction is executed. See Conditional execution for the range of available conditions and their encoding. If `<c>` is omitted, it defaults to *always* (`AL`).

`<q>` Specifies optional assembler qualifiers on the instruction. The following qualifiers are defined:

`.N` Meaning narrow, specifies that the assembler must select a 16-bit encoding for the instruction. If this is not possible, an assembler error is produced.

`.W` Meaning wide, specifies that the assembler must select a 32-bit encoding for the instruction. If this is not possible, an assembler error is produced.

If neither `.W` nor `.N` is specified, the assembler can select either 16-bit or 32-bit encodings. If both are available, it must select a 16-bit encoding. In a few cases, more than one encoding of the same length can be available for an instruction. The rules for selecting between such encodings are instruction-specific and are part of the instruction description. The assembler syntax includes a mandatory `.W` qualifier, along with a note describing the cases in which it applies, where this qualifier is required to select a particular encoding for an instruction. Additional assembler syntax will describe the syntax when the conditions are not met.

Note

When assembling to the A32 instruction set, the `.N` qualifier produces an assembler error and the `.W` qualifier has no effect.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F1.3 Conditional execution

Most T32 and A32 instructions can be executed conditionally, based on the values of the APSR Condition flags. Table F1-1 lists the available conditions.

Table F1-1 Condition codes

cond	Mnemonic extension	Meaning (integer)	Meaning (floating-point)[a]	Condition flags
0000	`EQ`	Equal	Equal	Z == 1
0001	`NE`	Not equal	Not equal, or unordered	Z == 0
0010	`CS`[b]	Carry set	Greater than, equal, or unordered	C == 1
0011	`CC`[c]	Carry clear	Less than	C == 0
0100	`MI`	Minus, negative	Less than	N == 1
0101	`PL`	Plus, positive or zero	Greater than, equal, or unordered	N == 0
0110	`VS`	Overflow	Unordered	V == 1
0111	`VC`	No overflow	Not unordered	V == 0
1000	`HI`	Unsigned higher	Greater than, or unordered	C == 1 and Z == 0
1001	`LS`	Unsigned lower or same	Less than or equal	C == 0 or Z == 1
1010	`GE`	Signed greater than or equal	Greater than or equal	N == V
1011	`LT`	Signed less than	Less than, or unordered	N != V
1100	`GT`	Signed greater than	Greater than	Z == 0 and N == V
1101	`LE`	Signed less than or equal	Less than, equal, or unordered	Z == 1 or N != V
1110	None (`AL`)[d]	Always (unconditional)	Always (unconditional)	Any

a. Unordered means at least one NaN operand.

b. `HS` (unsigned higher or same) is a synonym for `CS`.

c. `LO` (unsigned lower) is a synonym for `CC`.

d. `AL` is an optional mnemonic extension for always, except in `IT` instructions. For details, see IT.

In T32 instructions, the condition, if it is not `AL`, is normally encoded in a preceding `IT` instruction. For more information, see Conditional instructions and IT. Some conditional branch instructions do not require a preceding `IT` instruction, because they include a Condition code in their encoding.

Implementations can provide a set of ITD control fields, SCTLR.ITD, SCTLR_EL1.ITD, and HSCTLR.ITD, to disable use of IT for some instructions, making them UNDEFINED.

In A32 instructions, bits[31:28] of the instruction contain either:

- The Condition code, see The Condition code field in A32 instruction encodings.
- `0b1111` for some A32 instructions that can only be executed unconditionally.

F1.3.1 The Condition code field in A32 instruction encodings

Every conditional A32 instruction contains a 4-bit Condition code field, the `cond` field, in bits 31-28:

31 30 29 28	27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
cond	

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

This field contains one of the values `0b0000-0b1110`, as shown in Table F1-1. Most instruction mnemonics can be extended with the letters defined in the Mnemonic extension column of that table.

If the *always* (`AL`) condition is specified, the instruction is executed irrespective of the value of the Condition flags. The absence of a Condition code on an instruction mnemonic implies the `AL` Condition code.

F1.3.2 Pseudocode description of conditional execution

The `AArch32.CurrentCond()` function returns a 4-bit condition specifier as follows:

- For A32 instructions, it returns bits[31:28] of the instruction.
- For the T1 and T3 encodings of the Branch instruction (see B), it returns the 4-bit `cond` field of the encoding.
- For all other T32 instructions:
  - If `PSTATE.IT<3:0> != '0000'` it returns `PSTATE.IT<7:4>`.
  - If `PSTATE.IT<7:0> == '00000000'` it returns `'1110'`.
  - Otherwise, execution of the instruction is CONSTRAINED UNPREDICTABLE.

  For more information, see Process state, PSTATE.

The `ConditionPassed()` function uses this condition specifier and the Condition flags to determine whether the instruction must be executed, by calling the `ConditionHolds()` function.

A-profile Architecture Pseudocode includes the definitions of these functions.

Undefined Instruction exception describes the handling of conditional instructions that are UNDEFINED, UNPREDICTABLE, or CONSTRAINED UNPREDICTABLE. The pseudocode in the manual, as a sequential description of the instructions, has limitations in this respect. For more information, see Limitations of the instruction pseudocode.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F1.4 Shifts applied to a register

A32 register offset load/store word and unsigned byte instructions can apply a wide range of different constant shifts to the offset register. Both T32 and A32 data-processing instructions can apply the same range of different constant shifts to the second operand register. For details, see Constant shifts.

A32 data-processing instructions can apply a register-controlled shift to the second operand register.

F1.4.1 Constant shifts

These are the same in T32 and A32 instructions, except that the input bits come from different positions.

`<shift>` is an optional shift to be applied to `<Rm>`. It can be any one of:

(omitted) No shift.

`LSL #<n>` Logical shift left `<n>` bits. 1 <= `<n>` <= 31.

`LSR #<n>` Logical shift right `<n>` bits. 1 <= `<n>` <= 32.

`ASR #<n>` Arithmetic shift right `<n>` bits. 1 <= `<n>` <= 32.

`ROR #<n>` Rotate right `<n>` bits. 1 <= `<n>` <= 31.

`RRX` Rotate right one bit, with extend. Bit[0] is written to `shifter_carry_out`, bits[31:1] are shifted right one bit, and the Carry flag is shifted into bit[31].

Note

Assemblers can permit the use of some or all of `ASR #0`, `LSL #0`, `LSR #0`, and `ROR #0` to specify that no shift is to be performed. This is not standard UAL, and the encoding selected for T32 instructions might vary between UAL assemblers if it is used. To ensure disassembled code assembles to the original instructions, disassemblers must omit the shift specifier when the instruction specifies no shift.
Similarly, assemblers can permit the use of `#0` in the immediate forms of `ASR`, `LSL`, `LSR`, and `ROR` instructions to specify that no shift is to be performed, that is, that a `MOV` (register) instruction is wanted. Again, this is not standard UAL, and the encoding selected for T32 instructions might vary between UAL assemblers if it is used. To ensure disassembled code assembles to the original instructions, disassemblers must use the `MOV` (register) syntax when the instruction specifies no shift.

F1.4.1.1 Encoding

The assembler encodes `<shift>` into two type bits and five immediate bits, as follows:

(omitted) type = `0b00`, immediate = 0.

`LSL #<n>` type = `0b00`, immediate = `<n>`.

`LSR #<n>` type = `0b01`.

If `<n>` < 32, immediate = `<n>`.

If `<n>` == 32, immediate = 0.

`ASR #<n>` type = `0b10`.

If `<n>` < 32, immediate = `<n>`.

If `<n>` == 32, immediate = 0.

`ROR #<n>` type = `0b11`, immediate = `<n>`.

`RRX` type = `0b11`, immediate = 0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F1.4.2 Register controlled shifts

These are available only in A32 instructions.

`<type>` is the type of shift to apply to the value read from `<Rm>`. It must be one of:

`ASR`	Arithmetic shift right, encoded as type = `0b10`.
`LSL`	Logical shift left, encoded as type = `0b00`.
`LSR`	Logical shift right, encoded as type = `0b01`.
`ROR`	Rotate right, encoded as type = `0b11`.

The bottom byte of `<Rs>` contains the shift amount.

F1.4.3 Pseudocode description of instruction-specified shifts and rotates

The pseudocode enumeration `SRType {}` defines the shift types. Shift and rotate instruction decode is described by the pseudocode function:

- `DecodeImmShift()` for a constant shift.
- `DecodeRegShift()` for a register controlled shift.

Shift and rotate operations are made by the pseudocode function `Shift()`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F1.5 Memory accesses

Commonly, the following addressing modes are permitted for memory access instructions:

Offset addressing

The offset value is applied to an address obtained from the base register. The result is used as the address for the memory access. The value of the base register is unchanged.

The assembly language syntax for this mode is:

```
[<Rn>, <offset>]
```

Pre-indexed addressing

The offset value is applied to an address obtained from the base register. The result is used as the address for the memory access, and written back into the base register.

The assembly language syntax for this mode is:

```
[<Rn>, <offset>]!
```

Post-indexed addressing

The address obtained from the base register is used, unchanged, as the address for the memory access. The offset value is applied to the address, and written back into the base register

The assembly language syntax for this mode is:

```
[<Rn>], <offset>
```

In each case, `<Rn>` is the base register. `<offset>` can be:

- An immediate constant, such as `<imm8>` or `<imm12>`.
- An index register, `<Rm>`.
- A shifted index register, such as `<Rm>, LSL #<shift>`.

For information about unaligned access, endianness, and exclusive access, see:

- Alignment support.
- Endian support.
- Synchronization and semaphores.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F1.6 Encoding of lists of general-purpose registers and the PC

A number of instructions operate on lists of general-purpose registers. For some load instructions, the list of registers to be loaded can include the PC. For these instructions, the assembler syntax includes a `<registers>` field, which provides a list of the registers to be operated on, with list entries separated by commas.

The registers list is encoded in the instruction encoding. Most often, this is done using an 8-bit, 13-bit, or 16-bit `register_list` field. This section gives more information about these and other possible register list encodings.

In a `register_list` field, each bit corresponds to a single register, and if the `<registers>` field of the assembler instruction includes Rt then `register_list<t>` is set to 1, otherwise it is set to 0.

The full rules for the encoding of lists of general-purpose registers, and possibly the PC, are:

- Except for the cases listed here, 16-bit T32 encodings use an 8-bit register list, and can access only registers R0-R7.

  The exceptions to this rule are:

  — The T1 encoding of `POP` uses an 8-bit register list, and an additional bit, `P`, that corresponds to the PC. This means it can access any of R0-R7 and the PC.

  — The T1 encoding of `PUSH` uses an 8-bit register list, and an additional bit, `M`, that corresponds to the LR. This means it can access any of R0-R7 and the LR.
- 32-bit T32 encodings of load operations use a 13-bit register list, and two additional bits, `M`, corresponding to the LR, and `P`, corresponding to the PC. This means these instructions can access any of R0-R12 and the LR and PC.
- 32-bit T32 encodings of store operations use a 13-bit register list, and one additional bit, `M`, corresponding to the LR. This means these instructions can access any of R0-R12 and the LR.
- Except for the case listed here, A32 encodings use a 16-bit register list. This means these instructions can access any of R0-R12 and the SP, LR, and PC.

  The exception to this rule is:

  — The System instructions `LDM` (exception return) and `LDM` (User registers) use a 15-bit register list. This means these instructions can access any of R0-R12 and the SP and LR.
- The T3 and A2 encodings of `POP`, and the T3 and A2 encodings of `PUSH`, access a single register from the set of registers {R0-R12, LR, PC} and encode the register number in the `Rt` field.

Note

`POP` is a load operation, and `PUSH` is a store operation.

In every case, the encoding-specific pseudocode converts the register list into a 32-bit variable, `registers`, with a bit corresponding to each of the registers R0-R12, SP, LR, and PC.

Note

Some Advanced SIMD and floating-point instructions operate on lists of SIMD and floating-point registers. The assembler syntax of these instructions includes a `<list>` field that specifies the registers to be operated on, and the description of the instruction in Alphabetical list of T32 and A32 base instruction set instructions defines the use and encoding of this field.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F1.7 General information about the T32 and A32 instruction descriptions

T32 Instruction Set Encoding describes the T32 instruction encodings, and A32 Instruction Set Encoding describes the A32 instruction encodings. The following subsections give more information about the descriptions of these instructions and their encodings:

- Execution of instructions in debug state.
- Fixed values in AArch32 instruction and System register descriptions.
- UNDEFINED, UNPREDICTABLE, and CONSTRAINED UNPREDICTABLE instruction set space.
- T32 and A32 Advanced SIMD and floating-point instruction encodings.
- The PC and the use of `0b1111` as a register specifier in T32 and A32 instructions.
- The SP and the use of `0b1101` as a register specifier in T32 and A32 instructions.
- Modified immediate constants in T32 and A32 instructions.

F1.7.1 Execution of instructions in debug state

In general, except for the instructions described in Debug state, the T32 instruction descriptions do not indicate any differences in the behavior of the instruction if it is executed in Debug state. For this information, see Executing instructions in Debug state.

Note

- A32 instructions cannot be executed in Debug state.
- For many T32 instructions, execution is unchanged in Debug state. Executing instructions in Debug state identifies these instructions.

F1.7.2 Fixed values in AArch32 instruction and System register descriptions

This section summarizes the terms used to describe fixed values in AArch64 register and instruction descriptions. The Glossary gives full descriptions of these terms, and each entry in this section includes a link to the corresponding Glossary entry.

Note

In register descriptions, the meaning of some bits depends on the PE state. This affects the definitions of RES0 and RES1, as shown in the Glossary.

The following terms are used to describe bits or fields with fixed values:

RAZ Read-As-Zero. See Read-As-Zero (RAZ).

In diagrams, a RAZ bit can be shown as 0.

(0), RES0 Reserved, Should-Be-Zero (SBZ) or RES0.

In instruction encoding diagrams, and sometimes in other descriptions, `(0)` indicates an SBZ bit. If the bit is set to 1, behavior is CONSTRAINED UNPREDICTABLE, and must be one of the following:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if the value of the bit was 0.
- Any destination registers of the instruction become UNKNOWN.

This notation can be expanded for fields, so a three-bit field can be shown as either `(0)(0)(0)` or as `(000)`.

In register diagrams, but not in the A64 encoding and instruction descriptions, bits or fields can be shown as RES0. For more information, see the Glossary definition of RES0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Note

Some of the System instruction descriptions in this chapter are based on the *field description* of the input value for the instruction. These are register descriptions and therefore can include RES0 fields,

The `(0)` and RES0 descriptions can be applied to bits or bit fields that are read-only, or are write-only. The Glossary definitions cover these cases.

RAO Read-As-One. See Read-As-One (RAO).

In diagrams, a RAO bit can be shown as 1.

(1), RES1 Reserved, Should-Be-One (SBO) or RES1.

In instruction encoding diagrams, and sometimes in other descriptions, `(1)` indicates an SBO bit. If the bit is set to 0, behavior is CONSTRAINED UNPREDICTABLE, and must be one of the following:

- The instruction is UNDEFINED.
- The instruction is treated as a `NOP`.
- The instruction executes as if the value of the bit was 1.
- Any destination registers of the instruction become UNKNOWN.

This notation can be expanded for fields, so a three-bit field can be shown as either `(1)(1)(1)` or as `(111)`.

In register diagrams, but not in the A64 encoding and instruction descriptions, bits or fields can be shown as RES1. For more information, see the Glossary definition of *RES1*.

Note

Some of the System instruction descriptions in this chapter are based on the *field description* of the input value for the instruction. These are register descriptions and therefore can include RES1 fields.

The `(1)` and RES1 descriptions can be applied to bits or bit fields that are read-only, or are write-only. The Glossary definitions cover these cases.

Note

In register diagrams, (0) is a synonym for RES0, and (1) is a synonym for RES1, where RES0 and RES1 are defined in the Glossary. However, when used in an instruction encoding diagram, (0) and (1) have the narrower definition that behavior is UNPREDICTABLE or CONSTRAINED UNPREDICTABLE if either:

- A bit marked as (0) has the value 1.
- A bit marked as (1) has the value 0.

F1.7.3 UNDEFINED, UNPREDICTABLE, and CONSTRAINED UNPREDICTABLE instruction set space

An attempt to execute an unallocated instruction results in either:

- Unpredictable behavior. The instruction is described as UNPREDICTABLE or CONSTRAINED UNPREDICTABLE.

  From the introduction of Armv8-A, the architecturally UNPREDICTABLE behavior in AArch32 state is greatly reduced. Most cases that earlier versions of the architecture describe as UNPREDICTABLE become either:

  — CONSTRAINED UNPREDICTABLE, meaning the architecture defines a limited range of permitted behaviors.

  — Fully predictable.

  For more information, see Architectural Constraints on UNPREDICTABLE Behaviors.
- An Undefined Instruction exception. The instruction is described as UNDEFINED.

An instruction is UNDEFINED if it is declared as UNDEFINED in an instruction description, or in T32 Instruction Set Encoding or A32 Instruction Set Encoding.

An instruction is UNPREDICTABLE only if:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- It is declared as UNPREDICTABLE in an instruction description or in T32 Instruction Set Encoding or A32 Instruction Set Encoding, and Architectural Constraints on UNPREDICTABLE Behaviors does not redefine the behavior as CONSTRAINED UNPREDICTABLE.
- The pseudocode for that encoding does not indicate that a different special case applies, and a bit marked (0) or (1) in the encoding diagram of an instruction is not 0 or 1 respectively. In most cases, Armv8 makes these cases CONSTRAINED UNPREDICTABLE, as described in SBZ or SBO fields T32 and A32 in instructions.

Unless otherwise specified, T32 and A32 instructions provided as part of an architectural extension, or by an optional feature of the architecture, are UNDEFINED in an implementation that does not include that extension or feature.

Note

Examples of where this rule applies are:

- The instructions provided by the Cryptographic Extension.
- The System instructions that provide access to the System registers of the OPTIONAL Performance Monitors Extension.
- The Advanced SIMD and floating-point instructions.

For more information about UNDEFINED, UNPREDICTABLE, and CONSTRAINED UNPREDICTABLE instruction behavior, see Undefined Instruction exception.

F1.7.4 T32 and A32 Advanced SIMD and floating-point instruction encodings

The T32 and A32 encodings of Advanced SIMD and floating-point instructions that are described in T32 Instruction Set Encoding and in A32 Instruction Set Encoding are common to the T32 and A32 instruction sets. This means:

- The instruction groups, and the set of instructions in each group, are identical for T32 and A32.
- For each instruction:
  - Each T32 encoding is exactly equivalent to an A32 encoding.
  - There is no T32 encoding without an equivalent A32 encoding, and no A32 encoding without an equivalent T32 encoding.

Note

- In the T32 instruction sets, the Advanced SIMD and floating-point instructions have 32-bit encodings.
- In the base instruction sets, some instructions are common to the T32 and A32 instruction sets, whereas other instructions have equivalent but not identical functionality in the two instruction sets.

32-bit T32 encodings are described as two contiguous halfwords, `{hw1:hw2}`, as described in Instruction encodings. In general:

- `hw1` of a T32 encoding maps onto bits[31:16] of an equivalent A32 encoding.
- `hw2` of a T32 encoding maps onto bits[15:0] of an equivalent A32 encoding.

However, the different structures of the T32 instruction encoding space and the A32 instruction encoding space mean that:

- For a given Advanced SIMD and floating-point instruction group:
  - The positions of the fields that identify the instruction, or instruction encoding, within the instruction group might differ between the T32 encodings and the A32 encodings.
  - However, the field values that identify the instruction of instruction encoding are identical for the T32 encoding and the A32 encoding.

The remainder of this section describes the equivalence of the T32 and A32 encodings for each of the Advanced SIMD and floating-point instruction groups.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F1.7.4.1 Advanced SIMD data-processing

The T32 encoding of the Advanced SIMD data-processing group is:

The A32 encoding of the Advanced SIMD data-processing group is:

The encodings in this group are identified by:

- `hw1`[15:13] of the T32 encoding is equivalent to bits[27:25] of the A32 encoding, and:
  - Has the value `0b111` in the T32 encoding.
  - Has the value `0b001` in the A32 encoding.
- `hw1`[11:8] of the T32 encoding is equivalent to bits[31:28] of the A32 encoding, and has the value `0b111`.

This table shows the equivalence of the fields that identify the instructions, or instruction encodings, within this group:

T32 encoding	A32 encoding	Field size
`op0:op1`	`op0`	2 bits
`op2`	`op1`	15 bits
`op3`	`op2`	1 bit
`op4`	`op3`	1 bit

F1.7.4.2 Advanced SIMD element or structure load/store

The T32 encoding of the Advanced SIMD element or structure load/store group is:

The A32 encoding of the Advanced SIMD element or structure load/store group is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The encodings in this group are identified by:

- `hw1`[15:12] of the T32 encoding is equivalent to bits[31:28] of the A32 encoding, and has the value `0b1111`.
- `hw1`[11:8] of the T32 encoding is equivalent to bits[27:24] of the A32 encoding, and:
  - Has the value `0b1001` in the T32 encoding.
  - Has the value `0b0100` in the A32 encoding.
- `hw1`[4] of the T32 encoding is equivalent to bit[20] of the A32 encoding, and has the value `0b0`.

`op0`, `op1`, and `op2` are the fields that identify the instructions, or instruction encodings, within this group, and they are in equivalent positions in the T32 and A32 encodings.

F1.7.4.3 Advanced SIMD and floating-point load/store and 64-bit register moves

The T32 encoding of the Advanced SIMD and floating-point load/store and 64-bit register moves group is:

The A32 encoding of the Advanced SIMD and floating-point load/store and 64-bit register moves group is:

The encodings in the group are identified by:

- `hw1`[15:12] of the T32 encoding is equivalent to bits[31:28] of the A32 encoding, and:
  - Has the value `0b1110` in the T32 encoding.
  - Can have any value other than `0b1111` in the A32 encoding.
    This range of values is required because A32 instructions in this group can be executed conditionally, see Conditional execution.
- `hw1`[11:9] of the T32 encoding is equivalent to bits[27:25] of the A32 encoding, and has the value `0b110`.
- `hw2`[11:9] of the T32 encoding is equivalent to bits[11:9] of the A32 encoding, and has the value `0b101`.

`op0` is the field that identifies the instructions, or instruction encodings, within this group, and is in equivalent positions in the T32 and A32 encodings.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F1.7.4.4 Advanced SIMD and floating-point 32-bit register moves

The T32 encoding of the Advanced SIMD and floating-point 32-bit register moves group is:

The A32 encoding of the Advanced SIMD 32-bit register moves group is:

The encodings in this group are identified by:

- `hw1`[15:12] of the T32 encoding is equivalent to bits[31:28] of the A32 encoding, and:
  - Has the value `0b1110` in the T32 encoding.
  - Can have any value other than `0b1111` in the A32 encoding.
    This range of values is required because A32 instructions in this group can be executed conditionally, see Conditional execution.
- `hw1`[11:8] of the T32 encoding is equivalent to bits[27:24] of the A32 encoding, and has the value `0b1110`.
- `hw2`[11:9] of the T32 encoding is equivalent to bits[11:9] of the A32 encoding, and has the value `0b101`.
- `hw2`[4] of the T32 encoding is equivalent to bit[4] of the A32 encoding, and has the value `0b1`.

`op0` is the field that identifies the instructions, or instruction encodings, within this group, and is in equivalent positions in the T32 and A32 encodings.

F1.7.4.5 Floating-point data-processing

The T32 encoding of the Floating-point data-processing group is:

The A32 encoding of the Floating-point data-processing group is:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The encodings in this group are identified by:

- `hw1`[15:12] of the T32 encoding is equivalent to bits[31:28] of the A32 encoding, and:
  - In the T32 encoding, `hw1`[15:13] has the value `0b111`, and `hw1`[12] is the `op0` parameter used in identifying instruction encodings within this group.
  - In the A32 encoding, is the `cond` field and also implies the value of bit[28] of some A32 instruction encodings within this group, as the following table shows:

cond	Significance of bit[28] in A32 encodings
!= `0b1111`	Part of the `cond` field.
`0b1111`	Has fixed value of 1.

The range of `cond` values other than `0b1111` is required because A32 instructions in this group can be executed conditionally, see Conditional execution.

- `hw1`[11:8] of the T32 encoding is equivalent to bits[27:24] of the A32 encoding, and has the value `0b1110`.
- `hw2`[11:9] of the T32 encoding is equivalent to bits[11:9] of the A32 encoding, and has the value `0b101`.
- `hw2`[4] of the T32 encoding is equivalent to bit[4] of the A32 encoding, and has the value `0b0`.

This table shows the equivalence of the fields that identify the instructions, or instruction encodings, within this group:

T32 encoding	A32 encoding
`op0`	Bit[28] of the instruction encoding is 1 when `cond` is `0b1111`.
`op1`	`op0`
`op2`	`op1`
`op3`	`op2`

F1.7.5 The PC and the use of `0b1111` as a register specifier in T32 and A32 instructions

Restrictions on the use of PC or `0b1111` as a register specifier differ between the T32 and the A32 instruction sets, as described in:

- T32 restrictions on the use of the PC, and use of `0b1111` as a register specifier.
- A32 restrictions on the use of PC or `0b1111` as a register specifier.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F1.7.5.1 T32 restrictions on the use of the PC, and use of `0b1111` as a register specifier

The use of `0b1111` as a register specifier is not normally permitted in T32 instructions. When a value of `0b1111` is permitted, various meanings are possible. For register reads, these meanings include:

- Read the PC value, that is, the address of the current instruction + 4. The base register of the table branch instructions `TBB` and `TBH` can be the PC. This means branch tables can be placed in memory immediately after the instruction.

  Note

  Arm deprecates use of the PC as the base register in the `STC` instruction.

- Read the word-aligned PC value, that is, the address of the current instruction + 4, with bits[1:0] forced to zero. The base register of `LDC`, `LDR`, `LDRB`, `LDRD` (pre-indexed, no writeback), `LDRH`, `LDRSB`, and `LDRSH` instructions can be the word-aligned PC. This provides PC-relative data addressing. In addition, some encodings of the `ADD` and `SUB` instructions permit their source registers to be `0b1111` for the same purpose.
- Read zero. This is done in some cases when one instruction is a special case of another, more general instruction, but with one operand zero. In these cases, the instructions are listed on separate pages, with a special case in the pseudocode for the more general instruction cross-referencing the other page.

For register writes, these meanings include:

- The PC can be specified as the destination register of an `LDR` instruction. This is done by encoding Rt as `0b1111`. The loaded value is treated as an address, and the effect of execution is a branch to that address. Bit[0] of the loaded value selects whether to execute A32 or T32 instructions after the branch.
- Some other instructions write the PC in similar ways. An instruction can specify that the PC is written:
  - Implicitly, for example, branch instructions.
  - Explicitly by a register specifier of `0b1111`, for example 16-bit `MOV` (register) instructions.
  - Explicitly by using a register mask, for example `LDM` instructions.

  The address to branch to can be:
  - A loaded value, for example, `RFE`.
  - A register value, for example, `BX`.
  - The result of a calculation, for example, `TBB` or `TBH`.

  The method of choosing the instruction set used after the branch can be:
  - Similar to the `LDR` case, for example, `LDM` or `BX`.
  - A fixed instruction set other than the one currently being used, for example, the immediate form of `BLX`.
  - Unchanged, for example, branch instructions or 16-bit `MOV` (register) instructions.
  - Set from the SPSR.T bit, for `RFE` and `SUBS PC, LR, #imm8`.
- Discard the result of a calculation. This is done in some cases when one instruction is a special case of another, more general instruction, but with the result discarded. In these cases, the instructions are listed on separate pages, with a special case in the pseudocode for the more general instruction cross-referencing the other page.
- If the destination register specifier of an `LDRB`, `LDRH`, `LDRSB`, or `LDRSH` instruction is `0b1111`, the instruction is a memory hint instead of a load operation.
- If the destination register specifier of an `MRC` instruction is `0b1111`, bits[31:28] of the value transferred from the System register are written to the N, Z, C, and V condition flags in the APSR, and bits[27:0] are discarded.

F1.7.5.2 A32 restrictions on the use of PC or `0b1111` as a register specifier

In A32 instructions, the use of `0b1111` as a register specifier specifies the PC.

Many instructions are CONSTRAINED UNPREDICTABLE if they use `0b1111` as a register specifier. This is specified by pseudocode in the instruction description. Armv8-A constrains the resulting CONSTRAINED UNPREDICTABLE behavior, see Using R15 by instruction.

Note

Arm deprecates use of the PC as the base register in any store instruction.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F1.7.6 The SP and the use of `0b1101` as a register specifier in T32 and A32 instructions

In the T32 and A32 instruction sets, Arm recommends that the use of `0b1101` as a register specifier specifies the SP.

Note

- The recommendation that the register specifier `0b1101` is used only to specify the SP applies to both the T32 and the A32 instruction sets.
- Despite this recommendation, T32 instructions that can access R13, or the SP, behave predictably from the introduction of Armv8.

F1.7.7 Modified immediate constants in T32 and A32 instructions

The following sections describe the encoding of modified immediate constants:

- Modified immediate constants in T32 instructions.
- Modified immediate constants in A32 instructions.
- Modified immediate constants in T32 and A32 Advanced SIMD instructions.
- Modified immediate constants in T32 and A32 floating-point instructions.

F1.7.7.1 Modified immediate constants in T32 instructions

The encoding of a modified immediate constant in a 32-bit T32 instruction is:

15 14 13 12 11	10	9 8 7 6 5 4 3 2 1 0	15	14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
	i			imm3		a b c d	e f g h

Table F1-5 shows the range of modified immediate constants available in T32 data-processing instructions, and their encoding in the a, b, c, d, e, f, g, h, and i bits, and the imm3 field, in the instruction.

Table F1-5 Encoding of modified immediates in T32 data-processing instructions

i:imm3:a	<const>[a]
0000x	`00000000 00000000 00000000 abcdefgh`
0001x	`00000000 abcdefgh 00000000 abcdefgh`[b]
0010x	`abcdefgh 00000000 abcdefgh 00000000`[b]
0011x	`abcdefgh abcdefgh abcdefgh abcdefgh`[b]
01000	`1bcdefgh 00000000 00000000 00000000`
01001	`01bcdefg h0000000 00000000 00000000`[c]
01010	`001bcdef gh000000 00000000 00000000`
01011	`0001bcde fgh00000 00000000 00000000`[c]
.	. 8-bit values shifted to other positions
.	.
.	.
11101	`00000000 00000000 000001bc defgh000`[c]
11110	`00000000 00000000 0000001b cdefgh00`
11111	`00000000 00000000 00000001 bcdefgh0`[c]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

a. This table shows the immediate constant value in binary form, to relate `abcdefgh` to the encoding diagram. In assembly syntax, the immediate value is specified in the usual way (a decimal number by default).

b. Arm deprecates using a modified immediate with abcdefgh == 00000000, and these cases are CONSTRAINED UNPREDICTABLE, see UNPREDICTABLE cases in immediate constants in T32 data-processing instructions.

c. Not available in A32 instructions if `h` == 1.

Note

As the footnotes to Table F1-5 show, the range of values available in T32 modified immediate constants is slightly different from the range of values available in A32 instructions. See Modified immediate constants in A32 instructions for the A32 values.

F1.7.7.1.1 Carry out

A logical instruction with i:imm3:a == '00xxx' does not affect the Carry flag. Otherwise, a logical flag-setting instruction sets the Carry flag to the value of bit[31] of the modified immediate constant.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F1.7.7.1.2 Operation of modified immediate constants, T32 instructions

For a T32 data-processing instruction, the `T32ExpandImm()` pseudocode function returns the value of the 32-bit immediate constant, calling `T32ExpandImm_C()` to evaluate the constant.

F1.7.7.2 Modified immediate constants in A32 instructions

The encoding of a modified immediate constant in an A32 instruction is:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																				rotation				a	b	c	d	e	f	g	h

Table F1-6 shows the range of modified immediate constants available in A32 data-processing instructions, and their encoding in the a, b, c, d, e, f, g, and h bits and the rotation field in the instruction.

Table F1-6 Encoding of modified immediates in A32 processing instructions

rotation	<const>[a]
0000	`00000000 00000000 00000000 abcdefgh`
0001	`gh000000 00000000 00000000 00abcdef`
0010	`efgh0000 00000000 00000000 0000abcd`
0011	`cdefgh00 00000000 00000000 000000ab`
0100	`abcdefgh 00000000 00000000 00000000`
. . .	. . . 8-bit values shifted to other even-numbered positions
1001	`00000000 00abcdef gh000000 00000000`
. . .	. . . 8-bit values shifted to other even-numbered positions
1110	`00000000 00000000 0000abcd efgh0000`
1111	`00000000 00000000 000000ab cdefgh00`

a. This table shows the immediate constant value in binary form, to relate `abcdefgh` to the encoding diagram. In assembly syntax, the immediate value is specified in the usual way (a decimal number by default).

Note

The range of values available in A32 modified immediate constants is slightly different from the range of values available in 32-bit T32 instructions. See Modified immediate constants in T32 instructions.

F1.7.7.2.1 Carry out

A logical instruction with the rotation field set to `0b0000` does not affect APSR.C. Otherwise, a logical flag-setting instruction sets APSR.C to the value of bit[31] of the modified immediate constant.

F1.7.7.2.2 Constants with multiple encodings

Some constant values have multiple possible encodings. In this case, a UAL assembler must select the encoding with the lowest unsigned value of the rotation field. This is the encoding that appears first in Table F1-6. For example, the constant `#3` must be encoded with (rotation, abcdefgh) == (`0b0000`, `0b00000011`), not (`0b0001`, `0b00001100`), (`0b0010`, `0b00110000`), or (`0b0011`, `0b11000000`).

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

In particular, this means that all constants in the range 0-255 are encoded with rotation == `0b0000`, and permitted constants outside that range are encoded with rotation != `0b0000`. A flag-setting logical instruction with a modified immediate constant therefore leaves APSR.C unchanged if the constant is in the range 0-255 and sets it to the most significant bit of the constant otherwise. This matches the behavior of T32 modified immediate constants for all constants that are permitted in both the T32 and A32 instruction sets.

An alternative syntax is available for a modified immediate constant that permits the programmer to specify the encoding directly. In this syntax, `#<const>` is instead written as `#<byte>, #<rot>`, where:

`<byte>` Is the numeric value of abcdefgh, in the range 0-255.

`<rot>` Is twice the numeric value of rotation, an even number in the range 0-30.

This syntax permits all A32 data-processing instructions with modified immediate constants to be disassembled to assembler syntax that assembles to the original instruction.

This syntax also makes it possible to write variants of some flag-setting logical instructions that have different effects on APSR.C to those obtained with the normal `#<const>` syntax. For example, `ANDS R1, R2, #12, #2` has the same behavior as `ANDS R1, R2, #3` except that it sets APSR.C to 0 instead of leaving it unchanged. Such variants of flag-setting logical instructions do not have equivalents in the T32 instruction set, and Arm deprecates their use.

F1.7.7.2.3 Operation of modified immediate constants, A32 instructions

For an A32 data-processing instruction, the `A32ExpandImm()` pseudocode function returns the value of the 32-bit immediate constant, calling `A32ExpandImm_C()` to evaluate the constant.

F1.7.7.3 Modified immediate constants in T32 and A32 Advanced SIMD instructions

Table F1-7 shows the modified immediate constants available with Advanced SIMD instructions, and how they are encoded.

Table F1-7 Modified immediate values for Advanced SIMD instructions

op	cmode	Constant[a]	<dt>[b]	Notes
-	000x	00000000 00000000 00000000 abcdefgh 00000000 00000000 00000000 abcdefgh	I32	c
	001x	00000000 00000000 abcdefgh 00000000 00000000 00000000 abcdefgh 00000000	I32	c,d
	010x	00000000 abcdefgh 00000000 00000000 00000000 abcdefgh 00000000 00000000	I32	c,d
	011x	abcdefgh 00000000 00000000 00000000 abcdefgh 00000000 00000000 00000000	I32	c,d
	100x	00000000 abcdefgh 00000000 abcdefgh 00000000 abcdefgh 00000000 abcdefgh	I16	c
	101x	abcdefgh 00000000 abcdefgh 00000000 abcdefgh 00000000 abcdefgh 00000000	I16	c,d
	1100	00000000 00000000 abcdefgh 11111111 00000000 00000000 abcdefgh 11111111	I32	d,e
	1101	00000000 abcdefgh 11111111 11111111 00000000 abcdefgh 11111111 11111111	I32	d,e
0	1110	abcdefgh abcdefgh abcdefgh abcdefgh abcdefgh abcdefgh abcdefgh abcdefgh	I8	f
	1111	aBbbbbbc defgh000 00000000 00000000 aBbbbbbc defgh000 00000000 00000000	F32	f,g

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op	cmode	Constant[a]	<dt>[b]	Notes
1	1110	`aaaaaaaa bbbbbbbb cccccccc dddddddd eeeeeeee ffffffff gggggggg hhhhhhhh`	`I64`	[f]
	1111	UNDEFINED	-	-

a. In this table, the immediate value is shown in binary form, to relate abcdefgh to the encoding diagram. In assembler syntax, the constant is specified by a data type and a value of that type. That value is specified in the normal way (a decimal number by default) and is replicated enough times to fill the 64-bit immediate. For example, a data type of `I32` and a value of 10 specify the 64-bit constant `0x0000000A0000000A`.

b. This specifies the data type used when the instruction is disassembled. On assembly, the data type must be matched in the table if possible. Other data types are permitted as pseudo-instructions when a program is assembled, provided the 64-bit constant specified by the data type and value is available for the instruction. If a constant is available in more than one way, the first entry in this table that can produce it is used. For example, `VMOV.I64 D0, #0x8000000080000000` does not specify a 64-bit constant that is available from the I64 line of the table, but does specify one that is available from the fourth I32 line or the F32 line. It is assembled to the first of these, and therefore is disassembled as `VMOV.I32 D0, #0x80000000`.

c. This constant is available for the `VBIC`, `VMOV`, `VMVN`, and `VORR` instructions.

d. CONSTRAINED UNPREDICTABLE if `abcdefgh` == `0b00000000`, see UNPREDICTABLE cases in immediate constants in Advanced SIMD instructions. The required behavior is that these encodings produce an immediate constant of zero.

e. This constant is available for the `VMOV` and `VMVN` instructions only.

f. This constant is available for the `VMOV` instruction only.

g. In this entry, `B` = NOT(`b`). The bit pattern represents the floating-point number $(-1)^S \times 2^{exp} \times$ mantissa, where S = `UInt(a)`, exp = `UInt(NOT(b):c:d)-3` and mantissa = `(16+UInt(e:f:g:h))/16`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F1.7.7.3.1 Operation of modified immediate constants, Advanced SIMD instructions

For a T32 or A32 Advanced SIMD instruction that uses a modified immediate constant, the operation described by the `AdvSIMDExpandImm()` pseudocode function returns the value of the 64-bit immediate constant.

F1.7.7.4 Modified immediate constants in T32 and A32 floating-point instructions

Table F1-8 shows the immediate constants available in the `VMOV` (immediate) floating-point instruction, and Table F1-9 shows the resulting floating-point values.

Table F1-8 Floating-point modified immediate constants

Data type	imm4H	imm4L	Constant [a]
`F16`	abcd	efgh	`aBbbcdef gh000000`
`F32`	abcd	efgh	`aBbbbbbc defgh000 00000000 00000000`
`F64`	abcd	efgh	`aBbbbbbb bbcdefgh 00000000 00000000 00000000 00000000 00000000 00000000`

a. In this column, `B` = NOT(`b`). The bit pattern represents the floating-point number $(-1)^S \times 2^{exp} \times$ mantissa, where S = `UInt(a)`, exp = `UInt(NOT(b):c:d)-3` and mantissa = `(16+UInt(e:f:g:h))/16`.

Table F1-9 Floating-point constant values

efgh	bcd							
	`000`	`001`	`010`	`011`	`100`	`101`	`110`	`111`
`0000`	2.0	4.0	8.0	16.0	0.125	0.25	0.5	1.0
`0001`	2.125	4.25	8.5	17.0	0.1328125	0.265625	0.53125	1.0625
`0010`	2.25	4.5	9.0	18.0	0.140625	0.28125	0.5625	1.125
`0011`	2.375	4.75	9.5	19.0	0.1484375	0.296875	0.59375	1.1875
`0100`	2.5	5.0	10.0	20.0	0.15625	0.3125	0.625	1.25
`0101`	2.625	5.25	10.5	21.0	0.1640625	0.328125	0.65625	1.3125
`0110`	2.75	5.5	11.0	22.0	0.171875	0.34375	0.6875	1.375
`0111`	2.875	5.75	11.5	23.0	0.1796875	0.359375	0.71875	1.4375
`1000`	3.0	6.0	12.0	24.0	0.1875	0.375	0.75	1.5
`1001`	3.125	6.25	12.5	25.0	0.1953125	0.390625	0.78125	1.5625
`1010`	3.25	6.5	13.0	26.0	0.203125	0.40625	0.8125	1.625
`1011`	3.375	6.75	13.5	27.0	0.2109375	0.421875	0.84375	1.6875
`1100`	3.5	7.0	14.0	28.0	0.21875	0.4375	0.875	1.75
`1101`	3.625	7.25	14.5	29.0	0.2265625	0.453125	0.90625	1.8125
`1110`	3.75	7.5	15.0	30.0	0.234375	0.46875	0.9375	1.875
`1111`	3.875	7.75	15.5	31.0	0.2421875	0.484375	0.96875	1.9375

F1.7.7.4.1 Operation of modified immediate constants, floating-point instructions

For a T32 or A32 floating-point instruction that uses a modified immediate constant, the operation described by the `VFPExpandImm()` pseudocode function returns the value of the immediate constant.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F1.8 Additional pseudocode support for instruction descriptions

Earlier sections of this chapter include pseudocode that describes features of the execution of T32 and A32 instructions, see:

- Pseudocode description of conditional execution.
- Pseudocode description of instruction-specified shifts and rotates

The following subsection gives additional pseudocode support functions for some of the instructions described in Alphabetical list of T32 and A32 base instruction set instructions. See also Pseudocode support for the banked register transfer instructions.

F1.8.1 Pseudocode description of operations for System register access instructions

The `AArch32.SysRegRead()` function obtains the word for an `MRC` instruction from the System register.

The `AArch32.SysRegRead64()` function obtains the two words for an `MRRC` instruction from the System register.

Note

The relative significance of the two words returned is IMPLEMENTATION DEFINED, but all uses within this manual present the two words in the order (most significant, least significant).

The `AArch32.SysRegWrite()` procedure sends the word for an `MCR` instruction to the System register.

The `AArch32.SysRegWrite64()` procedure sends the two words for an `MCRR` instruction to the System register.

Note

The relative significance of `word2` and `word1` is IMPLEMENTATION DEFINED, but all uses within this manual treat `word2` as more significant than `word1`.

F1.8.2 Pseudocode details of system calls

The `AArch32.CallSupervisor()` pseudocode function generates a Supervisor Call exception. Valid execution of the `SVC` instruction calls this function.

The `AArch32.CallHypervisor()` pseudocode function generates an HVC exception. Valid execution of the `HVC` instruction calls this function.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F1.9 Additional information about Advanced SIMD and floating-point instructions

The following subsections give additional information about the Advanced SIMD and floating-point instructions:

- Advanced SIMD and floating-point instruction syntax.
- The Advanced SIMD addressing mode.
- Advanced SIMD instruction modifiers.
- Advanced SIMD operand shapes.
- Data type specifiers.
- Register specifiers.
- Register lists.
- Register encoding.
- Advanced SIMD scalars.

Note

The Advanced SIMD architecture, its associated implementations, and supporting software, are commonly referred to as NEON™ technology.

F1.9.1 Advanced SIMD and floating-point instruction syntax

Advanced SIMD and floating-point instructions use the general conventions of the T32 and A32 instruction sets.

Advanced SIMD and floating-point data-processing instructions use the following general format:

```
V{<modifier>}<operation>{<shape>}{<c>}{<q>}{.<dt>}  {<dest>,} <src1>, <src2>
```

All Advanced SIMD and floating-point instructions begin with a `V`. This distinguishes Advanced SIMD vector and floating-point instructions from scalar instructions.

The main operation is specified in the `<operation>` field. It is usually a three letter mnemonic the same as or similar to the corresponding scalar integer instruction.

The `<c>` and `<q>` fields are standard assembler syntax fields. For details, see Standard assembler syntax fields.

F1.9.2 The Advanced SIMD addressing mode

All the element and structure load/store instructions use this addressing mode. There is a choice of three formats:

`[<Rn>{:<align>}]` The address is contained in general-purpose register Rn.

Rn is not updated by this instruction.

Encoded as Rm = `0b1111`.

If Rn is encoded as `0b1111`, the instruction is CONSTRAINED UNPREDICTABLE.

`[<Rn>{:<align>}]!` The address is contained in general-purpose register Rn.

Rn is updated by this instruction: `Rn = Rn + transfer_size`

Encoded as Rm = `0b1101`.

`transfer_size` is the number of bytes transferred by the instruction. This means that, after the instruction is executed, Rn points to the address in memory immediately following the last address loaded from or stored to.

If Rn is encoded as `0b1111`, the instruction is CONSTRAINED UNPREDICTABLE.

This addressing mode can also be written as:

`[<Rn>{:align}], #<transfer_size>`

However, disassembly produces the `[<Rn>{:align}]!` form.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

`[<Rn>{:<align>}], <Rm>`

The address is contained in general-purpose register `<Rn>`.

Rn is updated by this instruction: `Rn = Rn + Rm`

Encoded as Rm = Rm. Rm must not be encoded as `0b1111` or `0b1101`, the PC or the SP.

If Rn is encoded as `0b1111`, the instruction is CONSTRAINED UNPREDICTABLE.

The CONSTRAINED UNPREDICTABLE behavior of encodings where Rn is `0b1111` is described in the section: Using R15 by instruction.

In all cases, `<align>` specifies an alignment, as specified by the individual instruction descriptions.

Previous versions of the manual used the `@` character for alignment. So, for example, the first format in this section was shown as `[<Rn>{@<align>}]`. Both `@` and `:` are supported. However, to ensure portability of code to assemblers that treat `@` as a comment character, `:` is preferred.

F1.9.3 Advanced SIMD instruction modifiers

The `<modifier>` field provides additional variants of some instructions. Table F1-10 provides definitions of the modifiers. Modifiers are not available for every instruction.

Table F1-10 Advanced SIMD instruction modifiers

<modifier>	Meaning
Q	The operation uses saturating arithmetic.
R	The operation performs rounding.
D	The operation doubles the result (before accumulation, if any).
H	The operation halves the result.

F1.9.4 Advanced SIMD operand shapes

The `<shape>` field provides additional variants of some instructions. Table F1-11 provides definitions of the shapes. Operand shapes are not available for every instruction.

Table F1-11 Advanced SIMD operand shapes

<shape>	Meaning	Typical register shape	
(none)	The operands and result are all the same width.	Dd, Dn, Dm	Qd, Qn, Qm
L	Long operation - result is twice the width of both operands	Qd, Dn, Dm	
N	Narrow operation - result is half the width of both operands	Dd, Qn, Qm	
W	Wide operation - result and first operand are twice the width of the second operand	Qd, Qn, Dm	

Note

- Some assemblers support a Q shape specifier, which requires all operands to be Q registers. An example of using this specifier is `VADDQ.S32 q0, q1, q2`. This is not standard UAL, and Arm recommends that programmers do not use a Q shape specifier.
- A disassembler must not generate any shape specifier not shown in Table F1-11.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F1.9.5 Data type specifiers

The `<dt>` field normally contains one data type specifier. Unless the assembler syntax description for the instruction indicates otherwise, this indicates the data type contained in:

- The second operand, if any.
- The operand, if there is no second operand.
- The result, if there are no operand registers.

The data types of the other operand and result are implied by the `<dt>` field combined with the instruction shape. For information about data type formats, see Data types supported by the Advanced SIMD implementation.

In the instruction syntax descriptions in About the T32 and A32 Instruction Descriptions, the `<dt>` field is usually specified as a single field. However, where more convenient, it is sometimes specified as a concatenation of two fields, `<type><size>`.

F1.9.5.1 Syntax flexibility

There is some flexibility in the data type specifier syntax:

- Software can specify three data types, specifying the result and both operand data types. For example:
  `VSUBW.I16.I16.S8 Q3, Q5, D0` instead of `VSUBW.S8 Q3, Q5, D0`
- Software can specify two data types, specifying the data types of the two operands. The data type of the result is implied by the instruction shape. For example:
  `VSUBW.I16.S8 Q3, Q5, D0` instead of `VSUBW.S8 Q3, Q5, D0`
- Software can specify two data types, specifying the data types of the single operand and the result. For example:
  `VMOVN.I16.I32 D0, Q1` instead of `VMOVN.I32 D0, Q1`
- Where an instruction requires a less specific data type, software can instead specify a more specific type, as shown in Table F1-12.
- Where an instruction does not require a data type, software can provide one.
- The `F32` data type can be abbreviated to `F`.
- The `F64` data type can be abbreviated to `D`.

In all cases, if software provides additional information, the additional information must match the instruction shape. Disassembly does not regenerate this additional information.

Table F1-12 Data type specification flexibility

Specified data type	Permitted more specific data types				
None	Any				
`.I<size>`	-	`.S<size>`	`.U<size>`	-	-
`.8`	`.I8`	`.S8`	`.U8`	`.P8`	-
`.16`	`.I16`	`.S16`	`.U16`	`.P16`	`.F16`
`.32`	`.I32`	`.S32`	`.U32`	-	`.F32` or `.F`
`.64`	`.I64`	`.S64`	`.U64`	-	`.F64` or `.D`

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F1.9.6 Register specifiers

The `<dest>`, `<src1>`, and `<src2>` fields contain register specifiers, or in some cases scalar specifiers or register lists. Table F1-13 shows the register and scalar specifier formats that appear in the instruction descriptions.

If `<dest>` is omitted, it is the same as `<src1>`.

Table F1-13 Advanced SIMD and floating-point register specifier formats

<specifier>	Usual meaning[a]	Used in
`<Qd>`	A quadword destination register for the result vector.	Advanced SIMD
`<Qn>`	A quadword source register for the first operand vector.	Advanced SIMD
`<Qm>`	A quadword source register for the second operand vector.	Advanced SIMD
`<Dd>`	A doubleword destination register for the result vector.	Both
`<Dn>`	A doubleword source register for the first operand vector.	Both
`<Dm>`	A doubleword source register for the second operand vector.	Both
`<Sd>`	A singleword destination register for the result vector.	Floating-point
`<Sn>`	A singleword source register for the first operand vector.	Floating-point
`<Sm>`	A singleword source register for the second operand vector.	Floating-point
`<Dd[x]>`	A destination scalar for the result. Element x of vector `<Dd>`.	Advanced SIMD
`<Dn[x]>`	A source scalar for the first operand. Element x of vector `<Dn>`.	Both[b]
`<Dm[x]>`	A source scalar for the second operand. Element x of vector `<Dm>`.	Advanced SIMD
`<Rt>`	A general-purpose register, used for a source or destination address.	Both
`<Rt2>`	A general-purpose register, used for a source or destination address.	Both
`<Rn>`	A general-purpose register, used as a load or store base address.	Both
`<Rm>`	A general-purpose register, used as a post-indexed address source.	Both

a. In some instructions the roles of registers are different.

b. In the floating-point instructions, `<Dn[x]>` is used only in `VMOV` (scalar to general-purpose register), see VMOV (scalar to general-purpose register).

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F1.9.7 Register lists

A register list is a list of register specifiers separated by commas and enclosed in brackets { and }. There are restrictions on what registers can appear in a register list. These restrictions are described in the individual instruction descriptions. Table F1-14 shows some register list formats, with examples of actual register lists corresponding to those formats.

Note

Register lists must not wrap around the end of the register bank.

F1.9.7.1 Syntax flexibility

There is some flexibility in the register list syntax:

- Where a register list contains consecutive registers, they can be specified as a range, instead of listing every register, for example `{D0-D3}` instead of `{D0, D1, D2, D3}`.
- Where a register list contains an even number of consecutive doubleword registers starting with an even-numbered register, it can be written as a list of quadword registers instead, for example `{Q1, Q2}` instead of `{D2-D5}`.
- Where a register list contains only one register, the enclosing braces can be omitted, for example `VLD1.8 D0, [R0]` instead of `VLD1.8 {D0}, [R0]`.

Table F1-14 Example register lists

Format	Example	Alternative
`{<Dd>}`	`{D3}`	`D3`
`{<Dd>, <Dd+1>, <Dd+2>}`	`{D3, D4, D5}`	`{D3-D5}`
`{<Dd[x]>, <Dd+2[x]}`	`{D0[3], D2[3]}`	-
`{<Dd[]>}`	`{D7[]}`	`D7[]`

F1.9.8 Register encoding

An Advanced SIMD register is either:

- *Quadword*, meaning it is 128 bits wide.
- *Doubleword*, meaning it is 64 bits wide.

Some instructions have options for either doubleword or quadword registers. This is normally encoded in Q, bit[6], as Q = 0 for doubleword operations, or Q = 1 for quadword operations.

A floating-point register is either:

- Double-precision, meaning it is 64 bits wide.
- Single-precision, meaning it is 32 bits wide.

This is encoded in the `sz` field, bit[8], as sz = 1 for double-precision operations, or sz = 0 for single-precision operations.

The T32 instruction encoding of Advanced SIMD or floating-point registers is:

15 14 13 12 11 10 9 8 7	6	5 4	3 2 1 0	15 14 13 12	11 10 9	8	7	6	5	4	3 2 1 0
	D		Vn	Vd		sz	N	Q	M		Vm

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The A32 instruction encoding of Advanced SIMD or floating-point registers is:

31 30 29 28 27 26 25 24 23	22	21 20	19 18 17 16	15 14 13 12	11 10 9	8	7	6	5	4	3 2 1 0
	D		Vn	Vd		sz	N	Q	M		Vm

Some instructions use only one or two registers, and use the unused register fields as additional opcode bits.

Table F1-15 shows the encodings for the registers.

Table F1-15 Encoding of register numbers

Register mnemonic	Usual usage	Register number encoded in[a]	Notes[a]	Used in
`<Qd>`	Destination (quadword)	D, Vd (bits[22, 15:13])	bit[12] == 0[b]	Advanced SIMD
`<Qn>`	First operand (quadword)	N, Vn (bits[7, 19:17])	bit[16] == 0[b]	Advanced SIMD
`<Qm>`	Second operand (quadword)	M, Vm (bits[5, 3:1])	bit[0] == 0[b]	Advanced SIMD
`<Dd>`	Destination (doubleword)	D, Vd (bits[22, 15:12])	-	Both
`<Dn>`	First operand (doubleword)	N, Vn (bits[7, 19:16])	-	Both
`<Dm>`	Second operand (doubleword)	M, Vm (bits[5, 3:0])	-	Both
`<Sd>`	Destination (single-precision)	Vd, D (bits[15:12, 22])	-	Floating-point
`<Sn>`	First operand (single-precision)	Vn, N (bits[19:16, 7])	-	Floating-point
`<Sm>`	Second operand (single-precision)	Vm, M (bits[3:0, 5])	-	Floating-point

a. Bit numbers given for the A32 instruction encoding. See the figures in this section for the equivalent bits in the T32 encoding.

b. If this bit is 1, the instruction is UNDEFINED.

F1.9.9 Advanced SIMD scalars

Advanced SIMD scalars can be 8-bit, 16-bit, 32-bit, or 64-bit. Instructions other than multiply instructions can access any element in the register set. The instruction syntax refers to the scalars using an index into a doubleword vector. The descriptions of the individual instructions contain details of the encodings.

Table F1-16 shows the form of encoding for scalars used in multiply instructions. These instructions cannot access scalars in some registers. The descriptions of the individual instructions contain cross references to this section where appropriate.

32-bit Advanced SIMD scalars, when used as single-precision floating-point numbers, are equivalent to Floating-point single-precision registers. That is, `Dm[x]` in a 32-bit context (0 <= `m` <= 15, 0 <= `x` <=1) is equivalent to `S[2m + x]`.

Table F1-16 Encoding of scalars in multiply instructions

Scalar mnemonic	Usual usage	Scalar size	Register specifier	Index specifier	Accessible registers
`<Dm[x]>`	Second operand	16-bit	Vm[2:0]	M, Vm[3]	D0-D7
		32-bit	Vm[3:0]	M	D0-D15

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Chapter F2
The AArch32 Instruction Sets Overview

This chapter describes the T32 and A32 instruction sets. It contains the following sections:

- Support for instructions in different versions of the Arm architecture.
- Unified Assembler Language.
- Branch instructions.
- Data-processing instructions.
- PSTATE and banked register access instructions.
- Load/store instructions.
- Load/store multiple instructions.
- Miscellaneous instructions.
- Exception-generating and exception-handling instructions.
- System register access instructions.
- Advanced SIMD and floating-point load/store instructions.
- Advanced SIMD and floating-point register transfer instructions.
- Advanced SIMD data-processing instructions.
- Floating-point data-processing instructions.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F2.1 Support for instructions in different versions of the Arm architecture

This manual describes the T32 and A32 instruction sets for the architecture. Therefore, it indicates how any options or extensions in the architecture affect the available instructions.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F2.2 Unified Assembler Language

This manual uses the Arm *Unified Assembler Language* (UAL). This assembly language syntax provides a canonical form for all T32 and A32 instructions.

UAL describes the syntax for the mnemonic and the operands of each instruction. In addition, it assumes that instructions and data items can be given labels. It does not specify the syntax to be used for labels, nor what assembler directives and options are available. See your assembler documentation for these details.

Most earlier Arm assembly language mnemonics are still supported as synonyms, as described in the instruction details.

Note

Most earlier T32 assembly language mnemonics are *not* supported.

UAL includes *instruction selection* rules that specify which instruction encoding is selected when more than one can provide the required functionality. For example, both 16-bit and 32-bit encodings exist for an `ADD R0, R1, R2` instruction. The most common instruction selection rule is that when both a 16-bit encoding and a 32-bit encoding are available, the 16-bit encoding is selected, to optimize code density.

Syntax options exist to override the normal instruction selection rules and ensure that a particular encoding is selected. These are useful when disassembling code, to ensure that subsequent assembly produces the original code, and in some other situations.

F2.2.1 Conditional instructions

For maximum portability of UAL assembly language between the T32 and A32 instruction sets, Arm recommends that:

- `IT` instructions are written before conditional instructions in the correct way for the T32 instruction set.
- When assembling to the A32 instruction set, assemblers check that any `IT` instructions are correct, but do not generate any code for them.

Although other T32 instructions are unconditional, all instructions that are made conditional by an `IT` instruction must be written with a condition. These conditions must match the conditions imposed by the `IT` instruction. For example, an `ITTEE EQ` instruction imposes the `EQ` condition on the first two following instructions, and the `NE` condition on the next two. Those four instructions must be written with `EQ`, `EQ`, `NE` and `NE` conditions respectively.

Some instructions cannot be made conditional by an `IT` instruction. Some instructions can be conditional if they are the last instruction in the IT block, but not otherwise.

The branch instruction encodings that include a condition code field cannot be made conditional by an `IT` instruction. If the assembler syntax indicates a conditional branch that correctly matches a preceding `IT` instruction, it is assembled using a branch instruction encoding that does not include a condition code field.

F2.2.2 Use of labels in UAL instruction syntax

The UAL syntax for some instructions includes the label of an instruction or a literal data item that is at a fixed offset from the instruction being specified. The assembler must:

1. Calculate the `PC` or `Align(PC, 4)` value of the instruction. The `PC` value of an instruction is its address plus 4 for a T32 instruction, or plus 8 for an A32 instruction. The `Align(PC, 4)` value of an instruction is its `PC` value ANDed with `0xFFFFFFFC` to force it to be word-aligned. There is no difference between the `PC` and `Align(PC, 4)` values for an A32 instruction, but there can be for a T32 instruction.
2. Calculate the offset from the `PC` or `Align(PC, 4)` value of the instruction to the address of the labeled instruction or literal data item.
3. Assemble a *PC-relative* encoding of the instruction, that is, one that reads its `PC` or `Align(PC, 4)` value and adds the calculated offset to form the required address.

   Note

   For instructions that can encode a subtraction operation, if the instruction cannot encode the calculated offset but can encode minus the calculated offset, the instruction encoding specifies a subtraction of minus the calculated offset.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

The syntax of the following instructions includes a label:

- `B`, `BL`, and `BLX` (immediate). The assembler syntax for these instructions always specifies the label of the instruction that they branch to. Their encodings specify a sign-extended immediate offset that is added to the `PC` value of the instruction to form the target address of the branch.
- `CBNZ` and `CBZ`. The assembler syntax for these instructions always specifies the label of the instruction that they branch to. Their encodings specify a zero-extended immediate offset that is added to the `PC` value of the instruction to form the target address of the branch. They do not support backward branches.
- `LDR`, `LDRB`, `LDRD`, `LDRH`, `LDRSB`, `LDRSH`, `PLD`, `PLDW`, `PLI`, and `VLDR`. The normal assembler syntax of these load instructions can specify the label of a literal data item that is to be loaded. The encodings of these instructions specify a zero-extended immediate offset that is either added to or subtracted from the `Align(PC, 4)` value of the instruction to form the address of the data item. A few such encodings perform a fixed addition or a fixed subtraction and must only be used when that operation is required, but most contain a bit that specifies whether the offset is to be added or subtracted.

  When the assembler calculates an offset of 0 for the normal syntax of these instructions, it must assemble an encoding that adds 0 to the `Align(PC, 4)` value of the instruction. Encodings that subtract 0 from the `Align(PC, 4)` value cannot be specified by the normal syntax.

  There is an alternative syntax for these instructions that specifies the addition or subtraction and the immediate offset explicitly. In this syntax, the label is replaced by `[PC, #+/-<imm>]`, where:

  `+/-` Is + or omitted to specify that the immediate offset is to be added to the `Align(PC, 4)` value, or – if it is to be subtracted.

  `<imm>` Is the immediate offset.

  This alternative syntax makes it possible to assemble the encodings that subtract 0 from the `Align(PC, 4)` value, and to disassemble them to a syntax that can be re-assembled correctly.
- `ADR`. The normal assembler syntax for this instruction can specify the label of an instruction or literal data item whose address is to be calculated. Its encoding specifies a zero-extended immediate offset that is either added to or subtracted from the `Align(PC, 4)` value of the instruction to form the address of the data item, and some opcode bits that determine whether it is an addition or subtraction.

  When the assembler calculates an offset of 0 for the normal syntax of this instruction, it must assemble the encoding that adds 0 to the `Align(PC, 4)` value of the instruction. The encoding that subtracts 0 from the `Align(PC, 4)` value cannot be specified by the normal syntax.

  There is an alternative syntax for this instruction that specifies the addition or subtraction and the immediate value explicitly, by writing them as additions `ADD <Rd>, PC, #<imm>` or subtractions `SUB <Rd>, PC, #<imm>`. This alternative syntax makes it possible to assemble the encoding that subtracts 0 from the `Align(PC, 4)` value, and to disassemble it to a syntax that can be re-assembled correctly.

Note

Arm recommends that where possible, software avoids using:

- The alternative syntax for the `ADR`, `LDC`, `LDR`, `LDRB`, `LDRD`, `LDRH`, `LDRSB`, `LDRSH`, `PLD`, `PLI`, `PLDW`, and `VLDR` instructions.
- The encodings of these instructions that subtract 0 from the `Align(PC, 4)` value.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F2.3 Branch instructions

Table F2-1 summarizes the branch instructions in the T32 and A32 instruction sets. In addition to providing for changes in the flow of execution, some branch instructions can change instruction set.

Table F2-1 Branch instructions

Instruction	See	Range, T32	Range, A32
Branch to target address	B	±16MB	±32MB
Compare and Branch on Nonzero Compare and Branch on Zero	CBNZ, CBZ	0-126 bytes	-[a]
Call a subroutine Call a subroutine, change instruction set[b]	BL, BLX (immediate)	±16MB ±16MB	±32MB ±32MB
Call a subroutine, optionally change instruction set	BLX (register)	Any	Any
Branch to target address, change instruction set	BX	Any	Any
Change to Jazelle state	BXJ	-	-
Table Branch (byte offsets) Table Branch (halfword offsets)	TBB, TBH	0-510 bytes 0-131070 bytes	-[a]

a. These instructions do not exist in the A32 instruction set.

b. The range is determined by the instruction set of the BLX instruction, not of the instruction it branches to.

Branches to loaded and calculated addresses can be performed by `LDR`, `LDM` and data-processing instructions. For details, see Load/store instructions, Load/store multiple instructions, Standard data-processing instructions, and Shift instructions.

In addition to the branch instructions shown in Table F2-1:

- In the A32 instruction set, a data-processing instruction that targets the PC behaves as a branch instruction. For more information, see Data-processing instructions.
- In the T32 and A32 instruction sets, a load instruction that targets the PC behaves as a branch instruction. For more information, see Load/store instructions.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F2.4 Data-processing instructions

Core data-processing instructions belong to one of the following groups:

- Standard data-processing instructions.
- Shift instructions.
- Multiply instructions.
- Saturating instructions.
- Saturating addition and subtraction instructions.
- Packing and unpacking instructions.
- Parallel addition and subtraction instructions.
- Divide instructions.
- Miscellaneous data-processing instructions.

For related Advanced SIMD and floating-point instructions see Advanced SIMD data-processing instructions and Floating-point data-processing instructions.

F2.4.1 Standard data-processing instructions

These instructions perform basic data-processing operations, and share a common format with some variations.

These instructions generally have a destination register Rd, a first operand register Rn, and a second operand. The second operand can be another register Rm, or an immediate constant.

If the second operand is an immediate constant, it can be:

- Encoded directly in the instruction.
- A *modified immediate constant* that uses 12 bits of the instruction to encode a range of constants. T32 and A32 instructions have slightly different ranges of modified immediate constants. For more information, see Modified immediate constants in T32 instructions and Modified immediate constants in A32 instructions.

If the second operand is another register, it can optionally be shifted in any of the following ways:

`LSL` Logical Shift Left by 1-31 bits.

`LSR` Logical Shift Right by 1-32 bits.

`ASR` Arithmetic Shift Right by 1-32 bits.

`ROR` Rotate Right by 1-31 bits.

`RRX` Rotate Right with Extend. For details, see Shift and rotate operations.

In T32 code, the amount to shift by is always a constant encoded in the instruction. In A32 code, the amount to shift by is either a constant encoded in the instruction, or the value of a register, Rs.

For instructions other than `CMN`, `CMP`, `TEQ`, and `TST`, the result of the data-processing operation is placed in the destination register. In the A32 instruction set, the destination register can be the PC, causing the result to be treated as a branch address. In the T32 instruction set, this is only permitted for some 16-bit forms of the `ADD` and `MOV` instructions.

These instructions can optionally set the Condition flags, according to the result of the operation. If they do not set the flags, existing flag settings from a previous instruction are preserved.

Table F2-2 summarizes the main data-processing instructions in the T32 and A32 instruction sets. Generally, each of these instructions is described in three sections in About the T32 and A32 Instruction Descriptions, one section for each of the following:

- `INSTRUCTION` (immediate) where the second operand is a modified immediate constant.
- `INSTRUCTION` (register) where the second operand is a register, or a register shifted by a constant.
- `INSTRUCTION` (register-shifted register) where the second operand is a register shifted by a value obtained from another register. These are only available in the A32 instruction set.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F2-2 Standard data-processing instructions

Instruction	Mnemonic	Notes
Add with Carry	`ADC`	-
Add	`ADD`	T32 instruction set permits use of a modified immediate constant or a zero-extended 12-bit immediate constant.
Form PC-relative Address	`ADR`	First operand is the PC. Second operand is an immediate constant. T32 instruction set uses a zero-extended 12-bit immediate constant. Operation is an addition or a subtraction.
Bitwise AND	`AND`	-
Bitwise Bit Clear	`BIC`	-
Compare Negative	`CMN`	Sets flags. Like `ADD` but with no destination register.
Compare	`CMP`	Sets flags. Like `SUB` but with no destination register.
Bitwise Exclusive OR	`EOR`	-
Copy operand to destination	`MOV`	Has only one operand, with the same options as the second operand in most of these instructions. If the operand is a shifted register, the instruction is an `LSL`, `LSR`, `ASR`, or `ROR` instruction instead. For details, see Shift instructions. The T32 and A32 instruction sets permit use of a modified immediate constant or a zero-extended 16-bit immediate constant.
Bitwise NOT	`MVN`	Has only one operand, with the same options as the second operand in most of these instructions.
Bitwise OR NOT	`ORN`	Not available in the A32 instruction set.
Bitwise OR	`ORR`	-
Reverse Subtract	`RSB`	Subtracts first operand from second operand. This permits subtraction from constants and shifted registers.
Reverse Subtract with Carry	`RSC`	Not available in the T32 instruction set.
Subtract with Carry	`SBC`	-
Subtract	`SUB`	T32 instruction set permits use of a modified immediate constant or a zero-extended 12-bit immediate constant.
Test Equivalence	`TEQ`	Sets flags. Like `EOR` but with no destination register.
Test	`TST`	Sets flags. Like `AND` but with no destination register.

F2.4.2 Shift instructions

Table F2-3 lists the shift instructions in the T32 and A32 instruction sets.

Table F2-3 Shift instructions

Instruction	See
Arithmetic Shift Right	ASR (immediate) ASR (register) ASRS (immediate) ASRS (register)

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Instruction	See
Logical Shift Left	LSL (immediate) LSL (register) LSLS (immediate) LSLS (register)
Logical Shift Right	LSR (immediate) LSR (register) LSRS (immediate) LSRS (register)
Rotate Right	ROR (immediate) ROR (register) RORS (immediate) RORS (register)
Rotate Right with Extend	RRX RRXS

In the A32 instruction set only, the destination register of these instructions can be the PC, causing the result to be treated as an address to branch to.

F2.4.3 Multiply instructions

These instructions can operate on signed or unsigned quantities. In some types of operation, the results are the same whether the operands are signed or unsigned.

- Table F2-4 summarizes the multiply instructions where there is no distinction between signed and unsigned quantities.
  The least significant 32 bits of the result are used. More significant bits are discarded.
- Table F2-5 summarizes the signed multiply instructions.
- Table F2-6 summarizes the unsigned multiply instructions.

Table F2-4 General multiply instructions

Instruction	See	Operation (number of bits)
Multiply Accumulate	MLA, MLAS	$32 = 32 + 32 \times 32$
Multiply and Subtract	MLS	$32 = 32 - 32 \times 32$
Multiply	MUL, MULS	$32 = 32 \times 32$

Table F2-5 Signed multiply instructions

Instruction	See	Operation (number of bits)
Signed Multiply Accumulate (halfwords)	SMLABB, SMLABT, SMLATB, SMLATT	$32 = 32 + 16 \times 16$
Signed Multiply Accumulate Dual	SMLAD, SMLADX	$32 = 32 + 16 \times 16 + 16 \times 16$
Signed Multiply Accumulate Long	SMLAL, SMLALS	$64 = 64 + 32 \times 32$

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Instruction	See	Operation (number of bits)
Signed Multiply Accumulate Long (halfwords)	SMLALBB, SMLALBT, SMLALTB, SMLALTT	64 = 64 + 16 × 16
Signed Multiply Accumulate Long Dual	SMLALD, SMLALDX	64 = 64 + 16 × 16 + 16 × 16
Signed Multiply Accumulate (word by halfword)	SMLAWB, SMLAWT	32 = 32 + 32 × 16[a]
Signed Multiply Subtract Dual	SMLSD, SMLSDX	32 = 32 + 16 × 16 - 16 × 16
Signed Multiply Subtract Long Dual	SMLSLD, SMLSLDX	64 = 64 + 16 × 16 - 16 × 16
Signed Most Significant Word Multiply Accumulate	SMMLA, SMMLAR	32 = 32 + 32 × 32[b]
Signed Most Significant Word Multiply Subtract	SMMLS, SMMLSR	32 = 32 - 32 × 32[b]
Signed Most Significant Word Multiply	SMMUL, SMMULR	32 = 32 × 32[b]
Signed Dual Multiply Add	SMUAD, SMUADX	32 = 16 × 16 + 16 × 16
Signed Multiply (halfwords)	SMULBB, SMULBT, SMULTB, SMULTT	32 = 16 × 16
Signed Multiply Long	SMULL, SMULLS	64 = 32 × 32
Signed Multiply (word by halfword)	SMULWB, SMULWT	32 = 32 × 16[a]
Signed Dual Multiply Subtract	SMUSD, SMUSDX	32 = 16 × 16 - 16 × 16

a. The most significant 32 bits of the 48-bit product are used. Less significant bits are discarded.

b. The most significant 32 bits of the 64-bit product are used. Less significant bits are discarded.

Table F2-6 Unsigned multiply instructions

Instruction	See	Operation (number of bits)
Unsigned Multiply Accumulate Accumulate Long	UMAAL	64 = 32 + 32 + 32 × 32
Unsigned Multiply Accumulate Long	UMLAL, UMLALS	64 = 64 + 32 × 32
Unsigned Multiply Long	UMULL, UMULLS	64 = 32 × 32

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F2.4.4 Saturating instructions

Table F2-7 lists the saturating instructions in the T32 and A32 instruction sets. For more information, see Pseudocode description of saturation.

Table F2-7 Saturating instructions

Instruction	See	Operation
Signed Saturate	SSAT	Saturates optionally shifted 32-bit value to selected range
Signed Saturate 16	SSAT16	Saturates two 16-bit values to selected range
Unsigned Saturate	USAT	Saturates optionally shifted 32-bit value to selected range
Unsigned Saturate 16	USAT16	Saturates two 16-bit values to selected range

F2.4.5 Saturating addition and subtraction instructions

Table F2-8 lists the saturating addition and subtraction instructions in the T32 and A32 instruction sets. For more information, see Pseudocode description of saturation.

Table F2-8 Saturating addition and subtraction instructions

Instruction	See	Operation
Saturating Add	QADD	Add, saturating result to the 32-bit signed integer range
Saturating Subtract	QSUB	Subtract, saturating result to the 32-bit signed integer range
Saturating Double and Add	QADD	Doubles one value and adds a second value, saturating the doubling and the addition to the 32-bit signed integer range
Saturating Double and Subtract	QDSUB	Doubles one value and subtracts the result from a second value, saturating the doubling and the subtraction to the 32-bit signed integer range

F2.4.6 Packing and unpacking instructions

Table F2-9 lists the packing and unpacking instructions in the T32 and A32 instruction sets.

Table F2-9 Packing and unpacking instructions

Instruction	See	Operation
Pack Halfword	PKHBT, PKHTB	Combine halfwords
Signed Extend and Add Byte	SXTAB	Extend 8 bits to 32 and add
Signed Extend and Add Byte 16	SXTAB16	Dual extend 8 bits to 16 and add
Signed Extend and Add Halfword	SXTAH	Extend 16 bits to 32 and add
Signed Extend Byte	SXTB	Extend 8 bits to 32

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Instruction	See	Operation
Signed Extend Byte 16	SXTB16	Dual extend 8 bits to 16
Signed Extend Halfword	SXTH	Extend 16 bits to 32
Unsigned Extend and Add Byte	UXTAB	Extend 8 bits to 32 and add
Unsigned Extend and Add Byte 16	UXTAB16	Dual extend 8 bits to 16 and add
Unsigned Extend and Add Halfword	UXTAH	Extend 16 bits to 32 and add
Unsigned Extend Byte	UXTB	Extend 8 bits to 32
Unsigned Extend Byte 16	UXTB16	Dual extend 8 bits to 16
Unsigned Extend Halfword	UXTH	Extend 16 bits to 32

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F2.4.7 Parallel addition and subtraction instructions

These instructions perform additions and subtractions on the values of two registers and write the result to a destination register, treating the register values as sets of two halfwords or four bytes. That is, they perform SIMD additions or subtractions on the general-purpose registers.

These instructions consist of a prefix followed by a main instruction mnemonic. The prefixes are as follows:

`S` Signed arithmetic modulo 2^8 or 2^{16}.

`Q` Signed saturating arithmetic.

`SH` Signed arithmetic, halving the results.

`U` Unsigned arithmetic modulo 2^8 or 2^{16}.

`UQ` Unsigned saturating arithmetic.

`UH` Unsigned arithmetic, halving the results.

The main instruction mnemonics are as follows:

`ADD16` Adds the top halfwords of two operands to form the top halfword of the result, and the bottom halfwords of the same two operands to form the bottom halfword of the result.

`ASX` Exchanges halfwords of the second operand, and then adds top halfwords and subtracts bottom halfwords.

`SAX` Exchanges halfwords of the second operand, and then subtracts top halfwords and adds bottom halfwords.

`SUB16` Subtracts each halfword of the second operand from the corresponding halfword of the first operand to form the corresponding halfword of the result.

`ADD8` Adds each byte of the second operand to the corresponding byte of the first operand to form the corresponding byte of the result.

`SUB8` Subtracts each byte of the second operand from the corresponding byte of the first operand to form the corresponding byte of the result.

The instruction set permits all 36 combinations of prefix and main instruction operand, as Table F2-10 shows.

See also Advanced SIMD parallel addition and subtraction.

Table F2-10 Parallel addition and subtraction instructions

Main instruction	Signed	Saturating	Signed halving	Unsigned	Unsigned saturating	Unsigned halving
`ADD16`, add, two halfwords	`SADD16`	`QADD16`	`SHADD16`	`UADD16`	`UQADD16`	`UHADD16`
`ASX`, add and subtract with exchange	`SASX`	`QASX`	`SHASX`	`UASX`	`UQASX`	`UHASX`
`SAX`, subtract and add with exchange	`SSAX`	`QSAX`	`SHSAX`	`USAX`	`UQSAX`	`UHSAX`
`SUB16`, subtract, two halfwords	`SSUB16`	`QSUB16`	`SHSUB16`	`USUB16`	`UQSUB16`	`UHSUB16`
`ADD8`, add, four bytes	`SADD8`	`QADD8`	`SHADD8`	`UADD8`	`UQADD8`	`UHADD8`
`SUB8`, subtract, four bytes	`SSUB8`	`QSUB8`	`SHSUB8`	`USUB8`	`UQSUB8`	`UHSUB8`

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F2.4.8 Divide instructions

Signed and unsigned integer divide instructions are included in both the T32 instruction set and the A32 instruction set.

For descriptions of the instructions see:

- SDIV.
- UDIV.

For the `SDIV` and `UDIV`

instructions, division by zero always returns a zero result.

The ID_ISAR0.Divide_instrs field indicates the level of support for these instructions. The field value of `0b0010` indicates they are implemented in both the T32 and A32 instruction sets.

F2.4.9 Miscellaneous data-processing instructions

Table F2-11 lists the miscellaneous data-processing instructions in the T32 and A32 instruction sets. Immediate values in these instructions are simple binary numbers.

Table F2-11 Miscellaneous data-processing instructions

Instruction	See	Notes
BitField Clear	BFC	-
BitField Insert	BFI	-
Count Leading Zeros	CLZ	-
Move Top	MOVT	Moves 16-bit immediate value to top halfword. Bottom halfword unchanged.
Reverse Bits	RBIT	-
Byte-Reverse Word	REV	-
Byte-Reverse Packed Halfword	REV16	-
Byte-Reverse Signed Halfword	REVSH	-
Signed BitField Extract	SBFX	-
Select Bytes using GE flags	SEL	-
Unsigned BitField Extract	UBFX	-
Unsigned Sum of Absolute Differences	USAD8	-
Unsigned Sum of Absolute Differences and Accumulate	USADA8	-

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F2.5 PSTATE and banked register access instructions

These instructions transfer PE state information to or from a general-purpose register.

F2.5.1 PSTATE access instructions

PSTATE holds process state information, see Process state, PSTATE. In AArch32 state:

- At EL1 or higher, PSTATE is accessible using the *Current Program Status Register* (CPSR).
- At EL0, a subset of the CPSR is accessible as the *Application Program Status Register* (APSR).
- On taking an exception, the contents of the CPSR are copied to the *Saved Program Status Register* (SPSR) of the mode from which the exception is taken.

The `MRS` and `MSR` instructions move the contents of the CPSR, APSR, or the SPSR of the current mode to or from a general-purpose register, see:

- MRS.
- MSR (immediate).
- MSR (register).

When executed at EL0, `MRS` and `MSR` instructions can only access the APSR.

The PSTATE Condition flags, PSTATE.{N, Z, C, V} are set by the execution of data-processing instructions, and can control the execution of conditional instructions. However, software can set the Condition flags explicitly using the `MSR` instruction, and can read the current state of the Condition flags explicitly using the `MRS` instruction.

In addition, at EL1 or higher, software can use the `CPS` instruction to change the PSTATE.M field and the PSTATE.{A, I, F} interrupt mask bits, see CPS, CPSID, CPSIE.

F2.5.2 Banked register access instructions

At EL1 or higher, the `MRS` (banked register) and `MSR` (banked register) instructions move the contents of a banked general-purpose register, the SPSR, or the ELR_hyp, to or from a general-purpose register. See:

- MRS (Banked register).
- MSR (Banked register).

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F2.6 Load/store instructions

Table F2-12 summarizes the general-purpose register load/store instructions in the T32 and A32 instruction sets. Some of these instructions can also operate on the PC. See also:

- Load/store multiple instructions.
- Synchronization and semaphores, for more information about the Load-Exclusive and Store-Exclusive instructions.
- Load-Acquire, Store-Release, for more information about the Load-Acquire/Store-Release and Load-Acquire Exclusive/Store-Release Exclusive instructions.
- Advanced SIMD and floating-point load/store instructions.

Load/store instructions have several options for addressing memory. For more information, see Addressing modes.

Table F2-12 Load/store instructions

	Load	Store	Unprivileged		Exclusive		Load-Acquire	Store-Release	Exclusive Load-Acquire	Store-Release
Data type			Load	Store	Load	Store				
32-bit word	LDR	STR	LDRT	STRT	LDREX	STREX	LDA	STL	LDAEX	STLEX
16-bit halfword	-	STRH	-	STRHT	-	STREXH	LDAH	STLH	LDAEXH	STLEXH
16-bit unsigned halfword	LDRH	-	LDRHT	–	LDREXH	-	-	-	-	-
16-bit signed halfword	LDRSH	-	LDRSHT	-	-	-	-	-	-	-
8-bit byte	-	STRB	-	STRBT	-	STREXB	LDAB	STLB	LDAEXB	STLEXB
8-bit unsigned byte	LDRB	–	LDRBT	-	LDREXB	-	-	-	-	-
8-bit signed byte	LDRSB	-	LDRSBT	-	-	-	-	-	-	-
Two 32-bit words	LDRD	STRD	-	-	-	-	-	-	-	-
64-bit doubleword	-	-	-	-	LDREXD	STREXD	-	-	LDAEXD	STLEXD

F2.6.1 Loads to the PC

The LDR instruction can load a value into the PC. The value loaded is treated as an interworking address, as described by the LoadWritePC() pseudocode function in Pseudocode description of operations on the AArch32 general-purpose registers and the PC.

F2.6.2 Halfword and byte loads and stores

Halfword and byte stores store the least significant halfword or byte from the register, to 16 or 8 bits of memory respectively. There is no distinction between signed and unsigned stores.

Halfword and byte loads load 16 or 8 bits from memory into the least significant halfword or byte of a register. Unsigned loads zero-extend the loaded value to 32 bits, and signed loads sign-extend the value to 32 bits.

F2.6.3 Load unprivileged and Store unprivileged

When executing at EL0, a Load unprivileged or Store unprivileged instruction operates in the same way as the corresponding ordinary load or store instruction. For example, an LDRT instruction executes in exactly the same way as the equivalent LDR instruction. When executed at PL1, Load unprivileged and Store unprivileged instructions behave as they would if they were executed at EL0. For example, an LDRT instruction executes in exactly the way that the equivalent LDR instruction would execute at EL0. In particular, the instructions make unprivileged memory accesses.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Note

As described in Security state, Exception levels, and AArch32 execution privilege, execution at PL1 describes all of the following:

- Execution at Non-secure EL1 using AArch32.
- Execution at Secure EL1 using AArch32 when EL3 is not implemented.
- Execution at Secure EL1 using AArch32 when EL3 is implemented and is using AArch64.
- Execution at Secure EL3 when EL3 is implemented and is using AArch32.

The Load unprivileged and Store unprivileged instructions are CONSTRAINED UNPREDICTABLE if executed at EL2.

For more information about execution privilege, see About access permissions.

F2.6.4 Load-Exclusive and Store-Exclusive

Load-Exclusive and Store-Exclusive instructions provide shared memory synchronization. For more information, see Synchronization and semaphores.

F2.6.5 Load-Acquire and Store-Release

Load-Acquire and Store-Release instructions provide memory barriers. Load-Acquire Exclusive and Store-Release Exclusive instructions provide memory barriers with shared memory synchronization. For more information, see Load-Acquire, Store-Release.

F2.6.6 Addressing modes

The address for a load or store is formed from two parts: a value from a base register, and an offset.

The base register can be any one of the general-purpose registers R0-R12, SP, or LR.

For loads, the base register can be the PC. This provides PC-relative addressing for position-independent code. Instructions marked (literal) in their title in About the T32 and A32 Instruction Descriptions are PC-relative loads.

The offset takes one of three formats:

Immediate The offset is an unsigned number that can be added to or subtracted from the base register value. Immediate offset addressing is useful for accessing data elements that are a fixed distance from the start of the data object, such as structure fields, stack offsets, and input/output registers.

Register The offset is a value from a general-purpose register. The value can be added to, or subtracted from, the base register value. Register offsets are useful for accessing arrays or blocks of data.

Scaled register The offset is a general-purpose register, shifted by an immediate value, then added to or subtracted from the base register. This means an array index can be scaled by the size of each array element.

The offset and base register can be used in three different ways to form the memory address. The addressing modes are described as follows:

Offset The offset is added to or subtracted from the base register to form the memory address.

Pre-indexed The offset is added to or subtracted from the base register to form the memory address. The base register is then updated with this new address, to permit automatic indexing through an array or memory block.

Post-indexed The value of the base register alone is used as the memory address. The offset is then added to or subtracted from the base register. The result is stored back in the base register, to permit automatic indexing through an array or memory block.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Note

Not every variant is available for every instruction, and the range of permitted immediate values and the options for scaled registers vary from instruction to instruction. See About the T32 and A32 Instruction Descriptions for full details for each instruction.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F2.7 Load/store multiple instructions

Load Multiple instructions load from memory a subset, or possibly all, of the general-purpose registers and the PC.

Store Multiple instructions store to memory a subset, or possibly all, of the general-purpose registers.

The memory locations are consecutive word-aligned words. The addresses used are obtained from a base register, and can be either above or below the value in the base register. The base register can optionally be updated by the total size of the data transferred.

Table F2-13 summarizes the load/store multiple instructions in the T32 and A32 instruction sets.

Table F2-13 Load/store multiple instructions

Instruction	See
Load Multiple, Increment After or Full Descending	LDM, LDMIA, LDMFD
Load Multiple, Decrement After or Full Ascending [a]	LDMDA, LDMFA
Load Multiple, Decrement Before or Empty Ascending	LDMDB, LDMEA
Load Multiple, Increment Before or Empty Descending [a]	LDMIB, LDMED
Pop multiple registers off the stack[b]	POP
Push multiple registers onto the stack[c]	PUSH
Store Multiple, Increment After or Empty Ascending	STM, STMIA, STMEA
Store Multiple, Decrement After or Empty Descending [a]	STMDA, STMED
Store Multiple, Decrement Before or Full Descending	STMDB, STMFD
Store Multiple, Increment Before or Full Ascending [a]	STMIB, STMFA

a. Not available in the T32 instruction set.

b. This instruction is equivalent to an `LDM` instruction with the SP as base register, and base register updating.

c. This instruction is equivalent to an `STMDB` instruction with the SP as base register, and base register updating.

When executing at EL1, variants of the `LDM` and `STM` instructions load and store User mode registers. Another system level variant of the `LDM` instruction performs an exception return.

F2.7.1 Loads to the PC

The `LDM`, `LDMDA`, `LDMDB`, `LDMIB`, and `POP` instructions can load a value into the PC. The value loaded is treated as an interworking address, as described by the `LoadWritePC()` pseudocode function in Pseudocode description of operations on the AArch32 general-purpose registers and the PC.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F2.8 Miscellaneous instructions

Table F2-14 summarizes the miscellaneous instructions in the T32 and A32 instruction sets.

Table F2-14 Miscellaneous instructions

Instruction	See
Clear Branch History	CLRBHB
Clear-Exclusive	CLREX
Data Memory Barrier	DMB
Data Synchronization Barrier	DSB
Error Synchronization Barrier	ESB
Instruction Synchronization Barrier	ISB
If-Then	IT
No Operation	NOP
Preload Data	PLD, PLDW (immediate) PLD (literal) PLD, PLDW (register)
Preload Instruction	PLI (immediate, literal) PLI (register)
Speculation Barrier	SB
Set Endianness	SETEND[a]
Set Privileged Access Never	SETPAN
Send Event	SEV
Send Event Local	SEVL
Wait For Event	WFE
Wait For Interrupt	WFI
Yield	YIELD[b]

a. Arm deprecates any use of the `SETEND` instruction.

b. See also The Yield instruction.

Note

Previous versions of the architecture defined the `DBG` instruction, that could provide a hint to the debug system, in this group. From the introduction of Armv8, this instruction executes as a `NOP`. Arm deprecates any use of the `DBG` instruction.

F2.8.1 The Yield instruction

In a *Symmetric Multithreading* (SMT) design, a thread can use the `YIELD` instruction to give a hint to the PE that it is running on. The `YIELD` hint indicates that whatever the thread is currently doing is of low importance, and so could yield. For example, the thread might be sitting in a spin-lock. A similar use might be in modifying the arbitration priority of the snoop bus in a multiprocessor (MP) system. Defining such an instruction permits binary compatibility between SMT and SMP systems.

AArch32 state defines a `YIELD` instruction as a specific NOP (No Operation) hint instruction.

The `YIELD` instruction has no effect in a single-threaded system, but developers of such systems can use the instruction to flag its intended use on migration to a multiprocessor or multithreading system. Operating systems can use `YIELD` in places where a yield hint is wanted, knowing that it will be treated as a `NOP` if there is no implementation benefit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F2.9 Exception-generating and exception-handling instructions

The following instructions are intended specifically to cause a synchronous exception to occur:

- The `SVC` instruction generates a Supervisor Call exception. For more information, see Supervisor Call (SVC) exception.
- The Breakpoint instruction `BKPT` provides software breakpoints. For more information, see Breakpoint Instruction exceptions.
- In an implementation that includes EL3 the `SMC` instruction generates a Secure Monitor Call exception. For more information, see Secure Monitor Call (SMC) exception.
- In an implementation that includes EL2 the `HVC` instruction generates a Hypervisor Call exception. For more information, see Hypervisor Call (HVC) exception.

Debug state summarizes the Debug state instructions.

For an exception taken to an EL1 mode:

- The system level variants of the `SUBS` and `LDM` instructions can perform a return from an exception.

  Note

  The variants of `SUBS` include `MOVS`. See the references to Subtract (exception return), Move (exception return), and Load Multiple (exception return) in Table F2-15 for more information.

- The `SRS` instruction can be used near the start of the handler, to store return information. The `RFE` instruction can then perform a return from the exception using the stored return information.

In an implementation that includes EL2, the `ERET` instruction performs a return from an exception taken to Hyp mode.

For more information, see Exception return to an Exception level using AArch32.

Table F2-15 summarizes the instructions, in the T32 and A32 instruction sets, for generating or handling an exception. Except for `BKPT` and `SVC`, these are system level instructions.

Table F2-15 Exception-generating and exception-handling instructions

Instruction	See
Supervisor Call	SVC
Breakpoint	BKPT
Secure Monitor Call	SMC
Return From Exception	RFE, RFEDA, RFEDB, RFEIA, RFEIB
Subtract (exception return)[a]	SUB, SUBS (immediate)[a]
Move (exception return)[a]	MOV, MOVS (register)[a]
Hypervisor Call	HVC
Exception Return	ERET
Load Multiple (exception return)	LDM (exception return)
Store Return State	SRS, SRSDA, SRSDB, SRSIA, SRSIB

a. The A32 instruction set includes other instruction forms that can be used for an exception return, that have previously been described as variants of `SUBS PC, LR`. Arm deprecates any use of these instruction forms.

F2.9.1 Debug state

Table F2-16 shows the Debug state instructions that are implemented in the T32 instruction set:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F2-16 T32 Debug state instructions

Mnemonic	Instruction	See	Note
`DCPSn`	Debug switch to EL*n*	DCPS1 DCPS2 DCPS3	-
`ERET`	Debug restore PE state (DRPS)	ERET	When executed in Debug state, the T1 encoding of `ERET` performs the DRPS operation

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F2.10 System register access instructions

The System register encoding space is indexed using the parameters {`coproc`, `opc1`, `CRn`, `CRm`, `opc2`}, see The AArch32 System register interface. This encoding space provides System registers and System instructions. The only permitted values of `coproc` are `0b1110` and `0b1111`, and the following instructions give access to this encoding space:

- Instructions that transfer data between general-purpose registers and System registers. See:
  - MCR.
  - MCRR.
  - MRC.
  - MRRC.
- Instructions that load or store from memory to a System register. See:
  - LDC (immediate).
  - LDC (literal).
  - STC.

Note

The System register encoding space with `coproc`== `0b101x` is redefined to provide some of the Advanced SIMD and floating-point functionality. That is, to:

- Initiate a floating-point data-processing operation, see Floating-point data-processing instructions.
- Transfer data between general-purpose registers and the Advanced SIMD and floating-point registers, see Advanced SIMD and floating-point register transfer instructions.
- Load or store data to the Advanced SIMD and floating-point registers, see Advanced SIMD and floating-point load/store instructions.

System register access instructions are part of the instruction stream executed by the PE, and therefore any System register access instruction that cannot be executed by the implementation causes an Undefined Instruction exception. In the A-profile and the R-profile architectures, the instruction encodings in the System register access instruction encoding space are unallocated, and generate Undefined Instruction exceptions, except for:

- The instructions summarized in this section that access the `coproc` == `0b111x` encoding space.
- The instructions in the `coproc` == `0b101x` encoding space that are redefined to provide Advanced SIMD and floating-point functionality, as summarized in the Note in this section.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F2.11 Advanced SIMD and floating-point load/store instructions

Table F2-17 summarizes the SIMD and floating-point register file load/store instructions in the Advanced SIMD and floating-point instruction sets.

Advanced SIMD also provides instructions for loading and storing multiple elements, or structures of elements, see Element and structure load/store instructions.

Table F2-17 SIMD and floating-point register file load/store instructions

Instruction	See	Operation
Vector Load Multiple	VLDM, VLDMDB, VLDMIA	Load 1-16 consecutive 64-bit registers, Advanced SIMD and floating-point. Load 1-32 consecutive 32-bit registers, floating-point only.
Vector Load Register	VLDR (immediate) VLDR (literal)	Load one 64-bit register, Advanced SIMD and floating-point. Load one 32-bit register, floating-point only.
Vector Store Multiple	VSTM, VSTMDB, VSTMIA	Store 1-16 consecutive 64-bit registers, Advanced SIMD and floating-point. Store 1-32 consecutive 32-bit registers, floating-point only.
Vector Store Register	VSTR	Store one 64-bit register, Advanced SIMD and floating-point. Store one 32-bit register, floating-point only.

F2.11.1 Element and structure load/store instructions

Table F2-18 shows the element and structure load/store instructions available in the Advanced SIMD instruction set. Loading and storing structures of more than one element automatically de-interleaves or interleaves the elements, see Figure F2-1 for an example of de-interleaving. Interleaving is the inverse process.

Table F2-18 Element and structure load/store instructions

Instruction	See
Load single element	
Multiple elements	VLD1 (multiple single elements)
To one lane	VLD1 (single element to one lane)
To all lanes	VLD1 (single element to all lanes)
Load 2-element structure	
Multiple structures	VLD2 (multiple 2-element structures)
To one lane	VLD2 (single 2-element structure to one lane)
To all lanes	VLD2 (single 2-element structure to all lanes)
Load 3-element structure	
Multiple structures	VLD3 (multiple 3-element structures)
To one lane	VLD3 (single 3-element structure to one lane)
To all lanes	VLD3 (single 3-element structure to all lanes)

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Instruction	See
Load 4-element structure	
Multiple structures	VLD4 (multiple 4-element structures)
To one lane	VLD4 (single 4-element structure to one lane)
To all lanes	VLD4 (single 4-element structure to all lanes)
Store single element	
Multiple elements	VST1 (multiple single elements)
From one lane	VST1 (single element from one lane)
Store 2-element structure	
Multiple structures	VST2 (multiple 2-element structures)
From one lane	VST2 (single 2-element structure from one lane)
Store 3-element structure	
Multiple structures	VST3 (multiple 3-element structures)
From one lane	VST3 (single 3-element structure from one lane)
Store 4-element structure	
Multiple structures	VST4 (multiple 4-element structures)
From one lane	VST4 (single 4-element structure from one lane)

Figure F2-1 shows the de-interleaving of a `VLD3.16` (multiple 3-element structures) instruction:

Figure F2-1 De-interleaving an array of 3-element structures

Figure F2-1 shows the `VLD3.16` instruction operating to three 64-bit registers that comprise four 16-bit elements:

- Different instructions in this group would produce similar figures, but operate on different numbers of registers. For example, `VLD4` and `VST4` instructions operate on four registers.
- Different element sizes would produce similar figures but with 8-bit or 32-bit elements.
- These instructions operate only on doubleword (64-bit) registers.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F2.12 Advanced SIMD and floating-point register transfer instructions

Table F2-19 summarizes the SIMD and floating-point register file transfer instructions in the Advanced SIMD and floating-point instruction sets. These instructions transfer data between the general-purpose registers and the registers in the SIMD and floating-point register file.

Advanced SIMD vectors, and single-precision and double-precision floating-point registers, are all views of the same register file. For details, see The SIMD and floating-point register file.

Table F2-19 SIMD and floating-point register file transfer instructions

Instruction	See
Copy element from general-purpose register to every element of an Advanced SIMD vector	VDUP (general-purpose register)
Copy byte, halfword, or word from general-purpose register to a register in the SIMD and floating-point register file	VMOV (general-purpose register to scalar)
Copy byte, halfword, or word from a register in the SIMD and floating-point register file to a general-purpose register	VMOV (scalar to general-purpose register)
Copy from half-precision floating-point register to general-purpose register, or from general-purpose register to half-precision floating-point register Only supported if FEAT_FP16 is implemented	VMOV (between general-purpose register and half-precision)
Copy from single-precision floating-point register to general-purpose register, or from general-purpose register to single-precision floating-point register	VMOV (between general-purpose register and single-precision)
Copy two words from general-purpose registers to consecutive single-precision floating-point registers, or from consecutive single-precision floating-point registers to general-purpose registers	VMOV (between two general-purpose registers and two single-precision registers)
Copy two words from general-purpose registers to a doubleword register in the SIMD and floating-point register file, or from a doubleword register in the SIMD and floating-point register file to general-purpose registers	VMOV (between two general-purpose registers and a doubleword floating-point register)
Copy from an Advanced SIMD and floating-point System Register to a general-purpose register	VMRS
Copy from a general-purpose register to an Advanced SIMD and floating-point System Register	VMSR

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F2.13 Advanced SIMD data-processing instructions

Advanced SIMD data-processing instructions process registers containing vectors of elements of the same type packed together, enabling the same operation to be performed on multiple items in parallel.

Instructions operate on vectors held in 64-bit or 128-bit registers. Figure F2-2 shows an operation on two 64-bit operand vectors, generating a 64-bit vector result.

Note

Figure F2-2 and other similar figures show 64-bit vectors that consist of four 16-bit elements, and 128-bit vectors that consist of four 32-bit elements. Other element sizes produce similar figures, but with 1, 2, 8, or 16 operations performed in parallel instead of 4.

Figure F2-2 Advanced SIMD instruction operating on 64-bit registers

Many Advanced SIMD instructions have variants that produce vectors of elements double the size of the inputs. In this case, the number of elements in the result vector is the same as the number of elements in the operand vectors, but each element, and the whole vector, is double the size.

Figure F2-3 shows an example of an Advanced SIMD instruction operating on 64-bit registers, and generating a 128-bit result.

Figure F2-3 Advanced SIMD instruction producing wider result

There are also Advanced SIMD instructions that have variants that produce vectors containing elements half the size of the inputs. Figure F2-4 shows an example of an Advanced SIMD instruction operating on one 128-bit register, and generating a 64-bit result.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Figure F2-4 Advanced SIMD instruction producing narrower result

Some Advanced SIMD instructions do not conform to these standard patterns. Their operation patterns are described in the individual instruction descriptions.

Advanced SIMD instructions that perform floating-point arithmetic use the Arm standard floating-point arithmetic defined in Floating-point support.

The following sections summarize the Advanced SIMD data-processing instructions:

- Advanced SIMD parallel addition and subtraction.
- Bitwise Advanced SIMD data-processing instructions.
- Advanced SIMD comparison instructions.
- Advanced SIMD shift instructions.
- Advanced SIMD multiply instructions.
- Advanced SIMD dot product instructions.
- Miscellaneous Advanced SIMD data-processing instructions.
- Advanced SIMD BFloat16 instructions.
- Advanced SIMD matrix multiply instructions.
- The Cryptographic Extension in AArch32 state.

F2.13.1 Advanced SIMD parallel addition and subtraction

Table F2-20 shows the Advanced SIMD parallel add and subtract instructions.

Table F2-20 Advanced SIMD parallel add and subtract instructions

Instruction	See
Vector Add	VADD (integer) VADD (floating-point)
Vector Add and Narrow, returning High Half	VADDHN
Vector Add Long	VADDL
Vector Add Wide	VADDW
Vector Halving Add	VHADD
Vector Halving Subtract	VHSUB
Vector Pairwise Add and Accumulate Long	VPADAL
Vector Pairwise Add	VPADD (integer) VPADD (floating-point)

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Instruction	See
Vector Pairwise Add Long	VPADDL
Vector Rounding Add and Narrow, returning High Half	VRADDHN
Vector Rounding Halving Add	VRHADD
Vector Rounding Subtract and Narrow, returning High Half	VRSUBHN
Vector Saturating Add	VQADD
Vector Saturating Subtract	VQSUB
Vector Subtract	VSUB (integer) VSUB (floating-point)
Vector Subtract and Narrow, returning High Half	VSUBHN
Vector Subtract Long	VSUBL
Vector Subtract Wide	VSUBW

F2.13.2 Bitwise Advanced SIMD data-processing instructions

Table F2-21 shows bitwise Advanced SIMD data-processing instructions. These operate on the doubleword (64-bit) or quadword (128-bit) registers in the SIMD and floating-point register file, and there is no division into vector elements.

Table F2-21 Bitwise Advanced SIMD data-processing instructions

Instruction	See
Vector Bitwise AND	VAND (register)
Vector Bitwise Bit Clear (AND complement)	VBIC (immediate) VBIC (register)
Vector Bitwise Exclusive OR	VEOR
Vector Bitwise Insert if False	VBIF
Vector Bitwise Insert if True	VBIT
Vector Bitwise Move	VMOV (immediate) VMOV (register)
Vector Bitwise NOT	VMVN (immediate) VMVN (register)
Vector Bitwise OR	VORR (immediate) VORR (register)
Vector Bitwise OR NOT	VORN (register)
Vector Bitwise Select	VBSL

F2.13.3 Advanced SIMD comparison instructions

Table F2-22 shows Advanced SIMD comparison instructions.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F2-22 Advanced SIMD comparison instructions

Instruction	See
Vector Absolute Compare Greater Than or Equal	VACGE
Vector Absolute Compare Greater Than	VACGT
Vector Compare Equal	VCEQ (register)
Vector Compare Equal to Zero	VCEQ (immediate #0)
Vector Compare Greater Than or Equal	VCGE (register)
Vector Compare Greater Than or Equal to Zero	VCGE (immediate #0)
Vector Compare Greater Than	VCGT (register)
Vector Compare Greater Than Zero	VCGT (immediate #0)
Vector Compare Less Than or Equal to Zero	VCLE (immediate #0)
Vector Compare Less Than Zero	VCLT (immediate #0)
Vector Test Bits	VTST

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F2.13.4 Advanced SIMD shift instructions

Table F2-23 lists the shift instructions in the Advanced SIMD instruction set.

Table F2-23 Advanced SIMD shift instructions

Instruction	See
Vector Saturating Rounding Shift Left	VQRSHL
Vector Saturating Rounding Shift Right and Narrow	VQRSHRN, VQRSHRUN
Vector Saturating Shift Left	VQSHL (register) VQSHL, VQSHLU (immediate)
Vector Saturating Shift Right and Narrow	VQSHRN, VQSHRUN
Vector Rounding Shift Left	VRSHL
Vector Rounding Shift Right	VRSHR
Vector Rounding Shift Right and Accumulate	VRSRA
Vector Rounding Shift Right and Narrow	VRSHRN
Vector Shift Left	VSHL (immediate) VSHL (register)
Vector Shift Left Long	VSHLL
Vector Shift Right	VSHR
Vector Shift Right and Narrow	VSHRN
Vector Shift Left and Insert	VSLI
Vector Shift Right and Accumulate	VSRA
Vector Shift Right and Insert	VSRI

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F2.13.5 Advanced SIMD multiply instructions

Table F2-24 shows the Advanced SIMD multiply instructions.

Table F2-24 Advanced SIMD multiply instructions

Instruction	See
Vector Multiply Accumulate	VMLA (integer) VMLA (floating-point) VMLA (by scalar)
Vector Multiply Accumulate Long	VMLAL (integer) VMLAL (by scalar)
Vector Multiply Subtract	VMLS (integer) VMLS (floating-point) VMLS (by scalar)
Vector Multiply Subtract Long	VMLSL (integer) VMLSL (by scalar)
Vector Multiply	VMUL (integer and polynomial) VMUL (floating-point) VMUL (by scalar)
Vector Multiply Long	VMULL (integer and polynomial) VMULL (by scalar)
Vector Fused Multiply Accumulate	VFMA
Vector Floating-Point Multiply-Add Long	VFMAL (vector) VFMAL (by scalar)
Vector Fused Multiply Subtract	VFMS
Vector Floating-Point Multiply-Subtract Long	VFMSL (vector) VFMSL (by scalar)
Vector Saturating Doubling Multiply Accumulate Long	VQDMLAL
Vector Saturating Doubling Multiply Subtract Long	VQDMLSL
Vector Saturating Doubling Multiply Returning High Half	VQDMULH
Vector Saturating Doubling Multiply Long	VQDMULL
Vector Saturating Rounding Doubling Multiply Accumulate Returning High Half	VQRDMLAH
Vector Saturating Rounding Doubling Multiply Subtract Returning High Half	VQRDMLSH
Vector Saturating Rounding Doubling Multiply Returning High Half	VQRDMULH

Advanced SIMD multiply instructions can operate on vectors of:

- 8-bit, 16-bit, or 32-bit unsigned integers.
- 8-bit, 16-bit, or 32-bit signed integers.
- 8-bit polynomials over {0, 1}. `VMUL` and `VMULL` are the only instructions that operate on polynomials. `VMULL` produces a 16-bit polynomial over {0, 1}.
- Single-precision (32-bit) or half-precision (16-bit) floating-point numbers.

They can also act on one vector and one scalar.

Long instructions have doubleword (64-bit) operands, and produce quadword (128-bit) results. Other Advanced SIMD multiply instructions can have either doubleword or quadword operands, and produce results of the same size.

Floating-point multiply instructions can operate on:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- Half-precision (16-bit) floating-point numbers.
- Single-precision (32-bit) floating-point numbers.
- Double-precision (64-bit) floating-point numbers.

F2.13.6 Advanced SIMD dot product instructions

FEAT_DotProd provides SIMD instructions that perform the dot product of the four 8-bit subelements of the 32-bit elements of one vector with the four 8-bit subelements of a second vector. It provides two forms of the instructions, each with signed and unsigned versions:

Vector form The dot product is calculated for each element of the first vector with the corresponding element of the second vector.

Indexed form The dot product is calculated for each element of the first vector with the element of the second vector that is indicated by the index argument to the instruction.

Note

That is, a single element from the second vector is used, and the dot product is calculated between each element of the first vector and this single element from the second vector.

Table F2-25 shows the Advanced SIMD dot product instructions.

Table F2-25 Advanced SIMD dot product instructions

Mnemonic	Instruction	See
VSDOT	Signed dot product (vector form)	VSDOT (vector)
VUDOT	Unsigned dot product (vector form)	VUDOT (vector)
VSDOT	Signed dot product (indexed form)	VSDOT (by element)
VSUDOT	Mixed sign integer dot product by indexed quadruplet[a]	VSUDOT (by element)
VUDOT	Unsigned dot product (indexed form)	VUDOT (by element)
VUSDOT	Mixed sign integer dot product (vector format)[a]	VUSDOT (vector)
	Mixed sign integer dot product by indexed quadruplet[a]	VUSDOT (by element)

a. This instruction is only supported when FEAT_AA32I8MM is implemented.

F2.13.7 AArch32 Advanced SIMD complex number arithmetic instructions

FEAT_FCMA provides AArch32 Advanced SIMD instructions that perform arithmetic on complex numbers held in element pairs in vector registers, where the less significant element of the pair contains the real component and the more significant element contains the imaginary component.

These instructions provide single-precision versions. If FEAT_FP16 is implemented they also provide half-precision versions, otherwise the half-precision encodings are UNDEFINED.

Table F2-26 shows the FEAT_FCMA AArch32 Advanced SIMD instructions.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F2-26 Advanced SIMD complex number arithmetic instructions

Mnemonic	Instruction	See
VCADD	Floating-point complex add	VCADD
VCMLA	Floating-point complex multiply accumulate (vector form)	VCMLA
VCMLA	Floating-point complex multiply accumulate (indexed form)	VCMLA (by element)

A pair of VCMLA instructions can be used to perform a complex number multiplication. In Complex multiplication, this is demonstrated for the similar AArch64 instruction FCMLA. The usage of VCMLA in this manner is identical.

F2.13.8 Advanced SIMD BFloat16 instructions

When FEAT_AA32BF16 is implemented, BFloat16 instructions are available in AArch32 state.

Table F2-27 shows the Advanced SIMD BFloat16 instructions.

Table F2-27 BFloat16 Advanced SIMD instructions

Mnemonic	Instruction	See
VDOT	BFloat16 floating-point vector dot product (vector and by scalar formats)	VDOT (vector) VDOT (by element)
VMMLA	BFloat16 floating-point matrix multiply-accumulate	VMMLA
VFMAB, VFMAT	BFloat16 floating-point widening multiply-add long (vector and by scalar formats)	VFMAB, VFMAT (BFloat16, vector) VFMAB, VFMAT (BFloat16, by scalar)
VCVT	BFloat16 convert from single-precision to BFloat16 format	VCVT (from single-precision to BFloat16, Advanced SIMD)

F2.13.9 Advanced SIMD matrix multiply instructions

When FEAT_AA32I8MM is implemented, these instructions are available in AArch32 state. They include integer and mixed sign dot product instructions.

The matrix multiply-accumulate instructions delimit source and destination vectors into segments. Within each segment:

- The first source vector matrix is organized in row-by-row order.
- The second source vector matrix elements are organized in a column-by-column order.
- The destination vector matrix is organized in row-by-row order.

One matrix multiplication is performed per segment.

Table F2-28 shows the Advanced SIMD matrix multiply instructions.

Table F2-28 Matrix multiply Advanced SIMD instructions

Mnemonic	Instruction	See
VSMMLA	Widening 8-bit signed integer matrix multiply-accumulate into 2x2 matrix	VSMMLA

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Mnemonic	Instruction	See
VUMMLA	Widening 8-bit unsigned integer matrix multiply-accumulate into 2x2 matrix	VUMMLA
VUSMMLA	Widening 8-bit mixed sign integer matrix multiply-accumulate into 2x2 matrix	VUSMMLA

F2.13.10 Miscellaneous Advanced SIMD data-processing instructions

Table F2-29 shows miscellaneous Advanced SIMD data-processing instructions.

Table F2-29 Miscellaneous Advanced SIMD data-processing instructions

Instruction	See
Vector Absolute Difference and Accumulate	VABA
Vector Absolute Difference and Accumulate Long	VABAL
Vector Absolute Difference	VABD (integer) VABD (floating-point)
Vector Absolute Difference Long	VABDL (integer)
Vector Absolute	VABS
Vector Convert between floating-point and fixed-point	VCVT (between floating-point and fixed-point, Advanced SIMD)
Vector Convert between floating-point and integer	VCVT (between floating-point and integer, Advanced SIMD)
Vector Convert between half-precision and single-precision	VCVT (between half-precision and single-precision, Advanced SIMD)
Vector Count Leading Sign Bits	VCLS
Vector Count Leading Zeros	VCLZ
Vector Count Set Bits	VCNT
Vector Duplicate scalar	VDUP (scalar)
Vector Extract	VEXT (byte elements)
Vector move Insertion	VINS
Vector Move and Narrow	VMOVN
Vector Move Long	VMOVL
Vector Move extraction	VMOVX
Vector Maximum	VMAX (integer) VMAX (floating-point)
Vector Minimum	VMIN (integer) VMIN (floating-point)
Vector Negate	VNEG
Vector Pairwise Maximum	VPMAX (integer) VPMAX (floating-point)
Vector Pairwise Minimum	VPMIN (integer) VPMIN (floating-point)
Vector Reciprocal Estimate	VRECPE
Vector Reciprocal Step	VRECPS
Vector Reciprocal Square Root Estimate	VRSQRTE

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Instruction	See
Vector Reciprocal Square Root Step	VRSQRTS
Vector Reverse in halfwords	VREV16
Vector Reverse in words	VREV32
Vector Reverse in doublewords	VREV64
Vector Saturating Absolute	VQABS
Vector Saturating Move and Narrow	VQMOVN, VQMOVUN
Vector Saturating Negate	VQNEG
Vector Swap	VSWP
Vector Table Lookup	VTBL, VTBX
Vector Transpose	VTRN
Vector Unzip	VUZP
Vector Zip	VZIP

F2.13.11 The Cryptographic Extension in AArch32 state

The instructions provided by the optional Cryptographic Extension use the Advanced SIMD and floating-point register file. For more information about the functions they provide see:

- *Advanced Encryption Standard (AES)*.
- *The Galois/Counter Mode of Operation*.
- *Secure Hash Algorithm (SHA)*.

Table F2-30 shows the AArch32 Cryptographic Extension instructions.

Table F2-30 AArch32 Cryptographic Extension instructions

Mnemonic	Instruction	See
`AESD`	AES single round decryption	AESD
`AESE`	AES single round encryption	AESE
`AESIMC`	AES inverse mix columns	AESIMC
`AESMC`	AES mix columns	AESMC
`VMULL`	Polynomial multiply long	VMULL (integer and polynomial) [a]
`SHA1C`	SHA1 hash update (choose)	SHA1C
`SHA1H`	SHA1 fixed rotate	SHA1H
`SHA1M`	SHA1 hash update (majority)	SHA1M
`SHA1P`	SHA1 hash update (parity)	SHA1P
`SHA1SU0`	SHA1 schedule update 0	SHA1SU0
`SHA1SU1`	SHA1 schedule update 1	SHA1SU1
`SHA256H`	SHA256 hash update (part 1)	SHA256H
`SHA256H2`	SHA256 hash update (part 2)	SHA256H2
`SHA256SU0`	SHA256 schedule update 0	SHA256SU0

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Mnemonic	Instruction	See
SHA256SU1	SHA256 schedule update 1	SHA256SU1

a. The Cryptographic Extension adds the variant of the instruction that operates on two 64-bit polynomials.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F2.14 Floating-point data-processing instructions

Table F2-31 summarizes the data-processing instructions in the floating-point instruction set. In this table, *floating-point register* means a register in the SIMD and floating-point register file. The BFloat16 floating-point instructions are provided by FEAT_AA32BF16.

For details of the floating-point arithmetic used by floating-point instructions, see Floating-point support.

Table F2-31 Floating-point data-processing instructions

Instruction	See
BFloat16 convert from single-precision to BFloat16 format writing to bottom half of single-precision register	VCVTB (BFloat16)
BFloat16 convert from single-precision to BFloat16 format writing to top half of single-precision register	VCVTT (BFloat16)
Convert between double-precision and single-precision	VCVT (between double-precision and single-precision)
Convert between floating-point and fixed-point	VCVT (between floating-point and fixed-point, floating-point)
Convert between half-precision and single-precision, writing to bottom half of single-precision register	VCVTB
Convert between half-precision and single-precision, writing to top half of single-precision register	VCVTT
Convert from floating-point to integer	VCVT (floating-point to integer, floating-point)
Convert from floating-point to integer using FPSCR rounding mode	VCVTR
Convert from integer to floating-point	VCVT (integer to floating-point, floating-point)
Floating-point Javascript convert to signed fixed-point, rounding toward zero	VJCVT
Copy from one floating-point register to another	VMOV (register)
Divide	VDIV
Move immediate value to a floating-point register	VMOV (immediate)
Square Root	VSQRT
Vector Absolute value	VABS
Vector Add	VADD (floating-point)
Vector Compare with exceptions disabled	VCMPE
Vector Compare with exceptions enabled	VCMP
Vector Fused Multiply Accumulate	VFMA
Vector Fused Multiply Subtract	VFMS
Vector Fused Negate Multiply Accumulate	VFNMA
Vector Fused Negate Multiply Subtract	VFNMS
Vector Multiply	VMUL (floating-point)
Vector Multiply Accumulate	VMLA (floating-point)
Vector Multiply Subtract	VMLS (floating-point)
Vector Negate Multiply	VNMUL
Vector Negate Multiply Accumulate	VNMLA
Vector Negate Multiply Subtract	VNMLS

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Instruction	See
Vector Negate, by inverting the sign bit	VNEG
Vector Subtract	VSUB (floating-point)

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Chapter F3
T32 Instruction Set Encoding

This chapter describes the encoding of the T32 instruction set. It contains the following sections:

- T32 instruction set encoding.
- About the T32 Advanced SIMD and floating-point instructions and their encoding.

In this chapter:

- In the decode tables, an entry of `x` for a field value means the value of the field does not affect the decoding.
- In the decode diagrams, a shaded field indicates that the bits in that field are not used in that level of decode.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F3.1 T32 instruction set encoding

The T32 instruction stream is a sequence of halfword-aligned halfwords. Each T32 instruction is either a single 16-bit halfword in that stream, or a 32-bit instruction consisting of two consecutive halfwords in that stream.

If the value of bits[15:11] of the halfword being decoded is one of the following, the halfword is the first halfword of a 32-bit instruction:

- `0b11101`.
- `0b11110`.
- `0b11111`.

Otherwise, the halfword is a 16-bit instruction.

The T32 instruction encoding is:

Table F3-1 Main encoding table for the T32 instruction set

Decode fields		Decode group or instruction page
op0	**op1**	
`!= 111`	`xx`	16-bit
`111`	`00`	B - T2 variant
`111`	`!= 00`	32-bit

F3.1.1 16-bit

This section describes the encoding of the 16-bit group. The encodings in this section are decoded from T32 instruction set encoding.

This decode also imposes the constraint:

- `op0<5:3> != 111`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F3-2 Encoding table for the 16-bit group

Decode fields op0	Decode group or instruction page
00xxxx	Shift (immediate), add, subtract, move, and compare
010000	Data-processing (two low registers)
010001	Special data instructions and branch and exchange
01001x	LDR (literal) - T1 variant
0101xx	Load/store (register offset)
011xxx	Load/store word/byte (immediate offset)
1000xx	Load/store halfword (immediate offset)
1001xx	Load/store (SP-relative)
1010xx	Add PC/SP (immediate)
1011xx	Miscellaneous 16-bit instructions
1100xx	Load/store multiple
1101xx	Conditional branch, and Supervisor Call

F3.1.1.1 Data-processing (two low registers)

This section describes the encoding of the Data-processing (two low registers) instruction class. The encodings in this section are decoded from 16-bit.

15	14	13	12	11	10	9 … 6	5 … 3	2 … 0
0	1	0	0	0	0	op	Rs	Rd

Table F3-3

Decode fields op	Instruction page
0000	AND, ANDS (register)
0001	EOR, EORS (register)
0010	MOV, MOVS (register-shifted register) - Logical shift left variant
0011	MOV, MOVS (register-shifted register) - Logical shift right variant
0100	MOV, MOVS (register-shifted register) - Arithmetic shift right variant
0101	ADC, ADCS (register)
0110	SBC, SBCS (register)
0111	MOV, MOVS (register-shifted register) - Rotate right variant
1000	TST (register)
1001	RSB, RSBS (immediate)

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields op	Instruction page
1010	CMP (register)
1011	CMN (register)
1100	ORR, ORRS (register)
1101	MUL, MULS
1110	BIC, BICS (register)
1111	MVN, MVNS (register)

F3.1.1.2 Load/store (register offset)

This section describes the encoding of the Load/store (register offset) instruction class. The encodings in this section are decoded from 16-bit.

15 14 13 12	11	10	9	8 6	5 3	2 0
0 1 0 1	L	B	H	Rm	Rn	Rt

Table F3-4

Decode fields L	B	H	Instruction page
0	0	0	STR (register)
0	0	1	STRH (register)
0	1	0	STRB (register)
0	1	1	LDRSB (register)
1	0	0	LDR (register)
1	0	1	LDRH (register)
1	1	0	LDRB (register)
1	1	1	LDRSH (register)

F3.1.1.3 Load/store word/byte (immediate offset)

This section describes the encoding of the Load/store word/byte (immediate offset) instruction class. The encodings in this section are decoded from 16-bit.

15 14 13 12	11	10 6	5 3	2 0
0 1 1 B	L	imm5	Rn	Rt

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F3-5

Decode fields		Instruction page
B	**L**	
0	0	STR (immediate)
0	1	LDR (immediate)
1	0	STRB (immediate)
1	1	LDRB (immediate)

F3.1.1.4 Load/store halfword (immediate offset)

This section describes the encoding of the Load/store halfword (immediate offset) instruction class. The encodings in this section are decoded from 16-bit.

15 14 13 12	11	10 … 6	5 … 3	2 … 0
1 0 0 0	L	imm5	Rn	Rt

Table F3-6

Decode fields	Instruction page
L	
0	STRH (immediate)
1	LDRH (immediate)

F3.1.1.5 Load/store (SP-relative)

This section describes the encoding of the Load/store (SP-relative) instruction class. The encodings in this section are decoded from 16-bit.

15 14 13 12	11	10 … 8	7 … 0
1 0 0 1	L	Rt	imm8

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F3-7

Decode fields L	Instruction page
0	STR (immediate)
1	LDR (immediate)

F3.1.1.6 Add PC/SP (immediate)

This section describes the encoding of the Add PC/SP (immediate) instruction class. The encodings in this section are decoded from 16-bit.

15 14 13 12	11	10 8	7 0
1 0 1 0	SP	Rd	imm8

Table F3-8

Decode fields SP	Instruction page
0	ADR
1	ADD, ADDS (SP plus immediate)

F3.1.1.7 Load/store multiple

This section describes the encoding of the Load/store multiple instruction class. The encodings in this section are decoded from 16-bit.

15 14 13 12	11	10 8	7 0
1 1 0 0	L	Rn	register_list

Table F3-9

Decode fields L	Instruction page
0	STM, STMIA, STMEA
1	LDM, LDMIA, LDMFD

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F3.1.2 Shift (immediate), add, subtract, move, and compare

This section describes the encoding of the Shift (immediate), add, subtract, move, and compare group. The encodings in this section are decoded from 16-bit.

Table F3-10 Encoding table for the Shift (immediate), add, subtract, move, and compare group

Decode fields			Decode group or instruction page
op0	op1	op2	
0	11	0	Add, subtract (three low registers)
0	11	1	Add, subtract (two low registers and immediate)
0	!= 11	x	MOV, MOVS (register) - T2 variant
1	xx	x	Add, subtract, compare, move (one low register and immediate)

F3.1.2.1 Add, subtract (three low registers)

This section describes the encoding of the Add, subtract (three low registers) instruction class. The encodings in this section are decoded from Shift (immediate), add, subtract, move, and compare.

Table F3-11

Decode fields	Instruction page
S	
0	ADD, ADDS (register)
1	SUB, SUBS (register)

F3.1.2.2 Add, subtract (two low registers and immediate)

This section describes the encoding of the Add, subtract (two low registers and immediate) instruction class. The encodings in this section are decoded from Shift (immediate), add, subtract, move, and compare.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

15 14 13 12	11 10	9	8 6	5 3	2 0
0 0 0 1	1 1	S	imm3	Rn	Rd

Table F3-12

Decode fields S	Instruction page
0	ADD, ADDS (immediate)
1	SUB, SUBS (immediate)

F3.1.2.3 Add, subtract, compare, move (one low register and immediate)

This section describes the encoding of the Add, subtract, compare, move (one low register and immediate) instruction class. The encodings in this section are decoded from Shift (immediate), add, subtract, move, and compare.

15 14 13 12	11 10 8	7 0
0 0 1 op	Rd	imm8

Table F3-13

Decode fields op	Instruction page
00	MOV, MOVS (immediate)
01	CMP (immediate)
10	ADD, ADDS (immediate)
11	SUB, SUBS (immediate)

F3.1.3 Special data instructions and branch and exchange

This section describes the encoding of the Special data instructions and branch and exchange group. The encodings in this section are decoded from 16-bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

15	10	9	8	7	0
010001		op0			

Table F3-14 Encoding table for the Special data instructions and branch and exchange group

Decode fields op0	Decode group or instruction page
`11`	Branch and exchange
`!= 11`	Add, subtract, compare, move (two high registers)

F3.1.3.1 Branch and exchange

This section describes the encoding of the Branch and exchange instruction class. The encodings in this section are decoded from Special data instructions and branch and exchange.

15	14	13	12	11	10	9	8	7	6	3	2	1	0
0	1	0	0	0	1	1	1	L	Rm		(0)	(0)	(0)

Table F3-15

Decode fields L	Instruction page
`0`	BX
`1`	BLX (register)

F3.1.3.2 Add, subtract, compare, move (two high registers)

This section describes the encoding of the Add, subtract, compare, move (two high registers) instruction class. The encodings in this section are decoded from Special data instructions and branch and exchange.

15	14	13	12	11	10	9	8	7	6	3	2	0
0	1	0	0	0	1	!=11		D	Rs		Rd	
						op						

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F3-16

Decode fields			Instruction page
op	D:Rd	Rs	
00	!= 1101	!= 1101	ADD, ADDS (register)
00	xxxx	1101	ADD, ADDS (SP plus register) - T1
00	1101	!= 1101	ADD, ADDS (SP plus register) - T2
01	xxxx	xxxx	CMP (register)
10	xxxx	xxxx	MOV, MOVS (register)

F3.1.4 Miscellaneous 16-bit instructions

This section describes the encoding of the Miscellaneous 16-bit instructions group. The encodings in this section are decoded from 16-bit.

Table F3-17 Encoding table for the Miscellaneous 16-bit instructions group

Decode fields				Decode group or instruction page	Feature
op0	op1	op2	op3		
0000	xx	x	xxxx	Adjust SP (immediate)	-
0010	xx	x	xxxx	Extend	-
0110	00	0	xxxx	SETPAN	FEAT_PAN
0110	00	1	xxxx	Unallocated.	-
0110	01	x	xxxx	Change Processor State	-
0110	1x	x	xxxx	Unallocated.	-
0111	xx	x	xxxx	Unallocated.	-
1000	xx	x	xxxx	Unallocated.	-
1010	10	x	xxxx	HLT	-
1010	!= 10	x	xxxx	Reverse bytes	-
1110	xx	x	xxxx	BKPT	-
1111	xx	x	0000	Hints	-
1111	xx	x	!= 0000	IT	-

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields				Decode group or instruction page	Feature
op0	op1	op2	op3		
x0x1	xx	x	xxxx	CBNZ, CBZ	-
x10x	xx	x	xxxx	Push and Pop	-

F3.1.4.1 Adjust SP (immediate)

This section describes the encoding of the Adjust SP (immediate) instruction class. The encodings in this section are decoded from Miscellaneous 16-bit instructions.

15	14	13	12	11	10	9	8	7	6		0
1	0	1	1	0	0	0	0	S	imm7		

Table F3-18

Decode fields	Instruction page
S	
0	ADD, ADDS (SP plus immediate)
1	SUB, SUBS (SP minus immediate)

F3.1.4.2 Extend

This section describes the encoding of the Extend instruction class. The encodings in this section are decoded from Miscellaneous 16-bit instructions.

15	14	13	12	11	10	9	8	7	6	5	3	2	0
1	0	1	1	0	0	1	0	U	B	Rm		Rd	

Table F3-19

Decode fields		Instruction page
U	B	
0	0	SXTH
0	1	SXTB
1	0	UXTH
1	1	UXTB

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F3.1.4.3 Change Processor State

This section describes the encoding of the Change Processor State instruction class. The encodings in this section are decoded from Miscellaneous 16-bit instructions.

15	14	13	12	11	10	9	8	7	6	5	4	0
1	0	1	1	0	1	1	0	0	1	op	flags	

Table F3-20

Decode fields		Instruction page
op	flags	
0	xxxxx	SETEND
1	0xxxx	CPS, CPSID, CPSIE - Interrupt enable variant
1	1xxxx	CPS, CPSID, CPSIE - Interrupt disable variant

F3.1.4.4 Reverse bytes

This section describes the encoding of the Reverse bytes instruction class. The encodings in this section are decoded from Miscellaneous 16-bit instructions.

15	14	13	12	11	10	9	8	7	6	5	3	2	0
1	0	1	1	1	0	1	0	!=10		Rm		Rd	
								op					

Table F3-21

Decode fields	Instruction page
op	
00	REV
01	REV16
11	REVSH

F3.1.4.5 Hints

This section describes the encoding of the Hints instruction class. The encodings in this section are decoded from Miscellaneous 16-bit instructions.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

15	14	13	12	11	10	9	8	7 – 4	3	2	1	0
1	0	1	1	1	1	1	1	hint	0	0	0	0

Table F3-22

Decode fields hint	Instruction page
0000	NOP
0001	YIELD
0010	WFE
0011	WFI
0100	SEV
0101	SEVL
011x	Reserved hint, behaves as NOP.
1xxx	Reserved hint, behaves as NOP.

F3.1.4.6 Push and Pop

This section describes the encoding of the Push and Pop instruction class. The encodings in this section are decoded from Miscellaneous 16-bit instructions.

15	14	13	12	11	10	9	8	7 – 0
1	0	1	1	L	1	0	P	register_list

Table F3-23

Decode fields L	Instruction page
0	PUSH
1	POP

F3.1.5 Conditional branch, and Supervisor Call

This section describes the encoding of the Conditional branch, and Supervisor Call group. The encodings in this section are decoded from 16-bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F3-24 Encoding table for the Conditional branch, and Supervisor Call group

Decode fields op0	Decode group or instruction page
`111x`	Exception generation
`!= 111x`	B - T1 variant

F3.1.5.1 Exception generation

This section describes the encoding of the Exception generation instruction class. The encodings in this section are decoded from Conditional branch, and Supervisor Call.

Table F3-25

Decode fields S	Instruction page
`0`	UDF
`1`	SVC

F3.1.6 32-bit

This section describes the encoding of the 32-bit group. The encodings in this section are decoded from T32 instruction set encoding.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

This decode also imposes the constraint:

- `op0<3:2> != 00`.

Table F3-26 Encoding table for the 32-bit group

Decode fields			Decode group or instruction page
op0	op1	op3	
x11x	xxxxx	x	System register access, Advanced SIMD, and floating-point
0100	xx0xx	x	Load/store multiple
0100	xx1xx	x	Load/store dual, load/store exclusive, load-acquire/store-release, and table branch
0101	xxxxx	x	Data-processing (shifted register)
10xx	xxxxx	1	Branches and miscellaneous control
10x0	xxxxx	0	Data-processing (modified immediate)
10x1	xxxx0	0	Data-processing (plain binary immediate)
10x1	xxxx1	0	Unallocated.
1100	1xxx0	x	Advanced SIMD element or structure load/store
1100	!= 1xxx0	x	Load/store single
1101	0xxxx	x	Data-processing (register)
1101	10xxx	x	Multiply, multiply accumulate, and absolute difference
1101	11xxx	x	Long multiply and divide

F3.1.6.1 Load/store multiple

This section describes the encoding of the Load/store multiple instruction class. The encodings in this section are decoded from 32-bit.

15 14 13 12	11 10 9 8	7 6 5 4	3 0	15 14 13	0
1 1 1 0 1 0 0	opc	0 W L	Rn	P M	register_list

Table F3-27

Decode fields		Instruction page
opc	L	
00	0	SRS, SRSDA, SRSDB, SRSIA, SRSIB - T1
00	1	RFE, RFEDA, RFEDB, RFEIA, RFEIB - T1
01	0	STM, STMIA, STMEA
01	1	LDM, LDMIA, LDMFD

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields		Instruction page
opc	L	
10	0	STMDB, STMFD
10	1	LDMDB, LDMEA
11	0	SRS, SRSDA, SRSDB, SRSIA, SRSIB - T2
11	1	RFE, RFEDA, RFEDB, RFEIA, RFEIB - T2

F3.1.6.2 Data-processing (shifted register)

This section describes the encoding of the Data-processing (shifted register) instruction class. The encodings in this section are decoded from 32-bit.

15 14 13 12	11 10 9 8	5 4	3 0	15 14 12	11 8	7 6 5 4	3 0
1 1 1 0 1 0 1	op1	S	Rn	(0) imm3	Rd	imm2 stype	Rm

Table F3-28

Decode fields					Instruction page
op1	S	Rn	imm3:imm2:stype	Rd	
0000	0	xxxx	!= 0000011	xxxx	AND, ANDS (register) - AND, shift or rotate by value variant
0000	0	xxxx	0000011	xxxx	AND, ANDS (register) - AND, rotate right with extend variant
0000	1	xxxx	!= 0000011	!= 1111	AND, ANDS (register) - ANDS, shift or rotate by value variant
0000	1	xxxx	!= 0000011	1111	TST (register) - Shift or rotate by value variant
0000	1	xxxx	0000011	!= 1111	AND, ANDS (register) - ANDS, rotate right with extend variant
0000	1	xxxx	0000011	1111	TST (register) - Rotate right with extend variant
0001	x	xxxx	!= 0000011	xxxx	BIC, BICS (register) - BICS, shift or rotate by value variant
0001	x	xxxx	0000011	xxxx	BIC, BICS (register) - BICS, rotate right with extend variant
0010	0	!= 1111	!= 0000011	xxxx	ORR, ORRS (register) - ORR, shift or rotate by value variant
0010	0	!= 1111	0000011	xxxx	ORR, ORRS (register) - ORR, rotate right with extend variant
0010	0	1111	!= 0000011	xxxx	MOV, MOVS (register) - MOV, shift or rotate by value variant
0010	0	1111	0000011	xxxx	MOV, MOVS (register) - MOV, rotate right with extend variant
0010	1	!= 1111	!= 0000011	xxxx	ORR, ORRS (register) - ORRS, shift or rotate by value variant
0010	1	!= 1111	0000011	xxxx	ORR, ORRS (register) - ORRS, rotate right with extend variant
0010	1	1111	!= 0000011	xxxx	MOV, MOVS (register) - MOVS, shift or rotate by value variant
0010	1	1111	0000011	xxxx	MOV, MOVS (register) - MOVS, rotate right with extend variant
0011	0	!= 1111	!= 0000011	xxxx	ORN, ORNS (register) - ORN, shift or rotate by value variant
0011	0	!= 1111	0000011	xxxx	ORN, ORNS (register) - ORN, rotate right with extend variant
0011	0	1111	!= 0000011	xxxx	MVN, MVNS (register) - MVN, shift or rotate by value variant

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields					Instruction page
op1	S	Rn	imm3:imm2:stype	Rd	
0011	0	1111	0000011	xxxx	MVN, MVNS (register) - MVN, rotate right with extend variant
0011	1	!= 1111	!= 0000011	xxxx	ORN, ORNS (register) - ORNS, shift or rotate by value variant
0011	1	!= 1111	0000011	xxxx	ORN, ORNS (register) - ORNS, rotate right with extend variant
0011	1	1111	!= 0000011	xxxx	MVN, MVNS (register) - MVNS, shift or rotate by value variant
0011	1	1111	0000011	xxxx	MVN, MVNS (register) - MVNS, rotate right with extend variant
0100	0	xxxx	!= 0000011	xxxx	EOR, EORS (register) - EOR, shift or rotate by value variant
0100	0	xxxx	0000011	xxxx	EOR, EORS (register) - EOR, rotate right with extend variant
0100	1	xxxx	!= 0000011	!= 1111	EOR, EORS (register) - EORS, shift or rotate by value variant
0100	1	xxxx	!= 0000011	1111	TEQ (register) - Shift or rotate by value variant
0100	1	xxxx	0000011	!= 1111	EOR, EORS (register) - EORS, rotate right with extend variant
0100	1	xxxx	0000011	1111	TEQ (register) - Rotate right with extend variant
0101	x	xxxx	xxxxxxx	xxxx	Unallocated.
0110	0	xxxx	xxxxx00	xxxx	PKHBT, PKHTB - PKHBT variant
0110	0	xxxx	xxxxx01	xxxx	Unallocated.
0110	0	xxxx	xxxxx10	xxxx	PKHBT, PKHTB - PKHTB variant
0110	0	xxxx	xxxxx11	xxxx	Unallocated.
0111	x	xxxx	xxxxxxx	xxxx	Unallocated.
1000	0	!= 1101	!= 0000011	xxxx	ADD, ADDS (register) - ADD, shift or rotate by value variant
1000	0	!= 1101	0000011	xxxx	ADD, ADDS (register) - ADD, rotate right with extend variant
1000	0	1101	!= 0000011	xxxx	ADD, ADDS (SP plus register) - ADD, shift or rotate by value variant
1000	0	1101	0000011	xxxx	ADD, ADDS (SP plus register) - ADD, rotate right with extend variant
1000	1	xxxx	!= 0000011	1111	CMN (register) - Shift or rotate by value variant
1000	1	!= 1101	!= 0000011	!= 1111	ADD, ADDS (register) - ADDS, shift or rotate by value variant
1000	1	!= 1101	0000011	!= 1111	ADD, ADDS (register) - ADDS, rotate right with extend variant
1000	1	xxxx	0000011	1111	CMN (register) - Rotate right with extend variant
1000	1	1101	!= 0000011	!= 1111	ADD, ADDS (SP plus register) - ADDS, shift or rotate by value variant
1000	1	1101	0000011	!= 1111	ADD, ADDS (SP plus register) - ADDS, rotate right with extend variant
1001	x	xxxx	xxxxxxx	xxxx	Unallocated.
1010	x	xxxx	!= 0000011	xxxx	ADC, ADCS (register) - ADCS, shift or rotate by value variant
1010	x	xxxx	0000011	xxxx	ADC, ADCS (register) - ADCS, rotate right with extend variant
1011	x	xxxx	!= 0000011	xxxx	SBC, SBCS (register) - SBCS, shift or rotate by value variant
1011	x	xxxx	0000011	xxxx	SBC, SBCS (register) - SBCS, rotate right with extend variant
1100	x	xxxx	xxxxxxx	xxxx	Unallocated.
1101	0	!= 1101	!= 0000011	xxxx	SUB, SUBS (register) - SUB, shift or rotate by value variant
1101	0	!= 1101	0000011	xxxx	SUB, SUBS (register) - SUB, rotate right with extend variant
1101	0	1101	!= 0000011	xxxx	SUB, SUBS (SP minus register) - SUB, shift or rotate by value variant

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields					Instruction page
op1	S	Rn	imm3:imm2:stype	Rd	
1101	0	1101	0000011	xxxx	SUB, SUBS (SP minus register) - SUB, rotate right with extend variant
1101	1	xxxx	!= 0000011	1111	CMP (register) - Shift or rotate by value variant
1101	1	!= 1101	!= 0000011	!= 1111	SUB, SUBS (register) - SUBS, shift or rotate by value variant
1101	1	!= 1101	0000011	!= 1111	SUB, SUBS (register) - SUBS, rotate right with extend variant
1101	1	xxxx	0000011	1111	CMP (register) - Rotate right with extend variant
1101	1	1101	!= 0000011	!= 1111	SUB, SUBS (SP minus register) - SUBS, shift or rotate by value variant
1101	1	1101	0000011	!= 1111	SUB, SUBS (SP minus register) - SUBS, rotate right with extend variant
1110	x	xxxx	!= 0000011	xxxx	RSB, RSBS (register) - RSBS, shift or rotate by value variant
1110	x	xxxx	0000011	xxxx	RSB, RSBS (register) - RSBS, rotate right with extend variant
1111	x	xxxx	xxxxxxx	xxxx	Unallocated.

F3.1.6.3 Data-processing (modified immediate)

This section describes the encoding of the Data-processing (modified immediate) instruction class. The encodings in this section are decoded from 32-bit.

15 14 13 12	11	10	9	8	5	4	3 0	15	14 12	11 8	7 0
1 1 1 1 0	i	0	op1			S	Rn	0	imm3	Rd	imm8

Table F3-29

Decode fields				Instruction page
op1	S	Rn	Rd	
0000	0	xxxx	xxxx	AND, ANDS (immediate) - AND variant
0000	1	xxxx	!= 1111	AND, ANDS (immediate) - ANDS variant
0000	1	xxxx	1111	TST (immediate)
0001	x	xxxx	xxxx	BIC, BICS (immediate)
0010	0	!= 1111	xxxx	ORR, ORRS (immediate) - ORR variant
0010	0	1111	xxxx	MOV, MOVS (immediate) - MOV variant
0010	1	!= 1111	xxxx	ORR, ORRS (immediate) - ORRS variant
0010	1	1111	xxxx	MOV, MOVS (immediate) - MOVS variant
0011	0	!= 1111	xxxx	ORN, ORNS (immediate) - Not flag setting variant
0011	0	1111	xxxx	MVN, MVNS (immediate) - MVN variant
0011	1	!= 1111	xxxx	ORN, ORNS (immediate) - Flag setting variant

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields				Instruction page
op1	S	Rn	Rd	
0011	1	1111	xxxx	MVN, MVNS (immediate) - MVNS variant
0100	0	xxxx	xxxx	EOR, EORS (immediate) - EOR variant
0100	1	xxxx	!= 1111	EOR, EORS (immediate) - EORS variant
0100	1	xxxx	1111	TEQ (immediate)
0101	x	xxxx	xxxx	Unallocated.
011x	x	xxxx	xxxx	Unallocated.
1000	0	!= 1101	xxxx	ADD, ADDS (immediate) - ADD variant
1000	0	1101	xxxx	ADD, ADDS (SP plus immediate) - ADD variant
1000	1	!= 1101	!= 1111	ADD, ADDS (immediate) - ADDS variant
1000	1	1101	!= 1111	ADD, ADDS (SP plus immediate) - ADDS variant
1000	1	xxxx	1111	CMN (immediate)
1001	x	xxxx	xxxx	Unallocated.
1010	x	xxxx	xxxx	ADC, ADCS (immediate)
1011	x	xxxx	xxxx	SBC, SBCS (immediate)
1100	x	xxxx	xxxx	Unallocated.
1101	0	!= 1101	xxxx	SUB, SUBS (immediate) - SUB variant
1101	0	1101	xxxx	SUB, SUBS (SP minus immediate) - SUB variant
1101	1	!= 1101	!= 1111	SUB, SUBS (immediate) - SUBS variant
1101	1	1101	!= 1111	SUB, SUBS (SP minus immediate) - SUBS variant
1101	1	xxxx	1111	CMP (immediate)
1110	x	xxxx	xxxx	RSB, RSBS (immediate)
1111	x	xxxx	xxxx	Unallocated.

F3.1.6.4 Long multiply and divide

This section describes the encoding of the Long multiply and divide instruction class. The encodings in this section are decoded from 32-bit.

15 14 13 12	11 10 9 8	7 6 4	3 0	15 12	11 8	7 4	3 0
1 1 1 1	1 0 1 1 1	op1	Rn	RdLo	RdHi	op2	Rm

Table F3-30

Decode fields		Instruction page
op1	op2	
000	!= 0000	Unallocated.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields		Instruction page
op1	op2	
000	0000	SMULL, SMULLS
001	!= 1111	Unallocated.
001	1111	SDIV
010	!= 0000	Unallocated.
010	0000	UMULL, UMULLS
011	!= 1111	Unallocated.
011	1111	UDIV
100	0000	SMLAL, SMLALS
100	0001	Unallocated.
100	001x	Unallocated.
100	01xx	Unallocated.
100	1000	SMLALBB, SMLALBT, SMLALTB, SMLALTT - SMLALBB variant
100	1001	SMLALBB, SMLALBT, SMLALTB, SMLALTT - SMLALBT variant
100	1010	SMLALBB, SMLALBT, SMLALTB, SMLALTT - SMLALTB variant
100	1011	SMLALBB, SMLALBT, SMLALTB, SMLALTT - SMLALTT variant
100	1100	SMLALD, SMLALDX - SMLALD variant
100	1101	SMLALD, SMLALDX - SMLALDX variant
100	111x	Unallocated.
101	0xxx	Unallocated.
101	10xx	Unallocated.
101	1100	SMLSLD, SMLSLDX - SMLSLD variant
101	1101	SMLSLD, SMLSLDX - SMLSLDX variant
101	111x	Unallocated.
110	0000	UMLAL, UMLALS
110	0001	Unallocated.
110	001x	Unallocated.
110	010x	Unallocated.
110	0110	UMAAL
110	0111	Unallocated.
110	1xxx	Unallocated.
111	xxxx	Unallocated.

F3.1.7 System register access, Advanced SIMD, and floating-point

This section describes the encoding of the System register access, Advanced SIMD, and floating-point group. The encodings in this section are decoded from 32-bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F3-31 Encoding table for the System register access, Advanced SIMD, and floating-point group

Decode fields				Decode group or instruction page
op0	op1	op2	op3	
x	0x	0x	x	Unallocated.
x	10	0x	x	Unallocated.
x	11	xx	x	Advanced SIMD data-processing
0	0x	1x	x	Advanced SIMD and System register load/store and 64-bit move
0	10	1x	1	Advanced SIMD and System register 32-bit move
0	10	10	0	Floating-point data-processing
0	10	11	0	Unallocated.
1	!= 11	1x	x	Additional Advanced SIMD and floating-point instructions

F3.1.8 Advanced SIMD data-processing

This section describes the encoding of the Advanced SIMD data-processing group. The encodings in this section are decoded from System register access, Advanced SIMD, and floating-point.

This group has encodings in both the T32 and A32 instruction sets. For information about mappings between the encodings of this group, see About the T32 Advanced SIMD and floating-point instructions and their encoding

Table F3-32 Encoding table for the Advanced SIMD data-processing group

Decode fields		Decode group or instruction page
op0	op1	
0	x	Advanced SIMD three registers of the same length

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields		Decode group or instruction page
op0	op1	
1	0	Advanced SIMD two registers, or three registers of different lengths
1	1	Advanced SIMD shifts and immediate generation

F3.1.8.1 Advanced SIMD three registers of the same length

This section describes the encoding of the Advanced SIMD three registers of the same length instruction class. The encodings in this section are decoded from Advanced SIMD data-processing.

15 14 13	12	11 10 9 8	7	6	5 4	3 0	15 12	11 8	7	6	5	4	3 0
1 1 1	U	1 1 1 1	0	D	size	Vn	Vd	opc	N	Q	M	o1	Vm

Table F3-33

Decode fields					Instruction page	Feature
U	size	opc	Q	o1		
0	0x	1100	x	1	VFMA	-
0	0x	1101	x	0	VADD (floating-point)	-
0	0x	1101	x	1	VMLA (floating-point)	-
0	0x	1110	x	0	VCEQ (register) - T2	-
0	0x	1111	x	0	VMAX (floating-point)	-
0	0x	1111	x	1	VRECPS	-
x	xx	0000	x	0	VHADD	-
0	00	0001	x	1	VAND (register)	-
x	xx	0000	x	1	VQADD	-
x	xx	0001	x	0	VRHADD	-
0	00	1100	x	0	SHA1C	FEAT_SHA1
x	xx	0010	x	0	VHSUB	-
0	01	0001	x	1	VBIC (register)	-
x	xx	0010	x	1	VQSUB	-
x	xx	0011	x	0	VCGT (register) - T1	-
x	xx	0011	x	1	VCGE (register) - T1	-
0	01	1100	x	0	SHA1P	FEAT_SHA1
0	1x	1100	x	1	VFMS	-
0	1x	1101	x	0	VSUB (floating-point)	-
0	1x	1101	x	1	VMLS (floating-point)	-
0	1x	1110	x	0	Unallocated.	-

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields					Instruction page	Feature
U	size	opc	Q	o1		
0	1x	1111	x	0	VMIN (floating-point)	-
0	1x	1111	x	1	VRSQRTS	-
x	xx	0100	x	0	VSHL (register)	-
0	xx	1000	x	0	VADD (integer)	-
0	10	0001	x	1	VORR (register)	-
0	xx	1000	x	1	VTST	-
x	xx	0100	x	1	VQSHL (register)	-
0	xx	1001	x	0	VMLA (integer)	-
x	xx	0101	x	0	VRSHL	-
x	xx	0101	x	1	VQRSHL	-
0	xx	1011	x	0	VQDMULH	-
0	10	1100	x	0	SHA1M	FEAT_SHA1
0	xx	1011	x	1	VPADD (integer)	-
x	xx	0110	x	0	VMAX (integer)	-
0	11	0001	x	1	VORN (register)	-
x	xx	0110	x	1	VMIN (integer)	-
x	xx	0111	x	0	VABD (integer)	-
x	xx	0111	x	1	VABA	-
0	11	1100	x	0	SHA1SU0	FEAT_SHA1
1	0x	1101	x	0	VPADD (floating-point)	-
1	0x	1101	x	1	VMUL (floating-point)	-
1	0x	1110	x	0	VCGE (register) - T2	-
1	0x	1110	x	1	VACGE	-
1	0x	1111	0	0	VPMAX (floating-point)	-
1	0x	1111	x	1	VMAXNM	-
1	00	0001	x	1	VEOR	-
x	xx	1001	x	1	VMUL (integer and polynomial)	-
1	00	1100	x	0	SHA256H	FEAT_SHA256
x	xx	1010	0	0	VPMAX (integer)	-
1	01	0001	x	1	VBSL	-
x	xx	1010	0	1	VPMIN (integer)	-
x	xx	1010	1	x	Unallocated.	-
1	01	1100	x	0	SHA256H2	FEAT_SHA256
1	1x	1101	x	0	VABD (floating-point)	-
1	1x	1110	x	0	VCGT (register) - T2	-
1	1x	1110	x	1	VACGT	-
1	1x	1111	0	0	VPMIN (floating-point)	-
1	1x	1111	x	1	VMINNM	-
1	xx	1000	x	0	VSUB (integer)	-

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields					Instruction page	Feature
U	size	opc	Q	o1		
1	10	0001	x	1	VBIT	-
1	xx	1000	x	1	VCEQ (register) - T1	-
1	xx	1001	x	0	VMLS (integer)	-
1	xx	1011	x	0	VQRDMULH	-
1	10	1100	x	0	SHA256SU1	FEAT_SHA256
1	xx	1011	x	1	VQRDMLAH	FEAT_RDM
1	11	0001	x	1	VBIF	-
1	xx	1100	x	1	VQRDMLSH	FEAT_RDM
1	xx	1111	1	0	Unallocated.	-

F3.1.9 Advanced SIMD two registers, or three registers of different lengths

This section describes the encoding of the Advanced SIMD two registers, or three registers of different lengths group. The encodings in this section are decoded from Advanced SIMD data-processing.

Table F3-34 Encoding table for the Advanced SIMD two registers, or three registers of different lengths group

Decode fields				Decode group or instruction page
op0	op1	op2	op3	
0	11	xx	x	VEXT (byte elements)
1	11	0x	x	Advanced SIMD two registers misc
1	11	10	x	VTBL, VTBX
1	11	11	x	Advanced SIMD duplicate (scalar)
x	!= 11	xx	0	Advanced SIMD three registers of different lengths
x	!= 11	xx	1	Advanced SIMD two registers and a scalar

F3.1.9.1 Advanced SIMD two registers misc

This section describes the encoding of the Advanced SIMD two registers misc instruction class. The encodings in this section are decoded from Advanced SIMD two registers, or three registers of different lengths.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

15 14 13 12	11 10 9 8	7 6	5 4	3 2 1 0	15 12	11 10	7 6	5	4	3 0
1 1 1 1	1 1 1 1	1 D	1 1	size	opc1	Vd	0	opc2	Q	M	0	Vm

Table F3-35

Decode fields				Instruction page	Feature
size	**opc1**	**opc2**	**Q**		
xx	00	0000	x	VREV64	-
xx	00	0001	x	VREV32	-
xx	00	0010	x	VREV16	-
xx	00	0011	x	Unallocated.	-
xx	00	010x	x	VPADDL	-
xx	00	0110	0	AESE	FEAT_AES
xx	00	0110	1	AESD	FEAT_AES
xx	00	0111	0	AESMC	FEAT_AES
xx	00	0111	1	AESIMC	FEAT_AES
xx	00	1000	x	VCLS	-
00	10	0000	x	VSWP	-
xx	00	1001	x	VCLZ	-
xx	00	1010	x	VCNT	-
xx	00	1011	x	VMVN (register)	-
00	10	1100	1	Unallocated.	-
xx	00	110x	x	VPADAL	-
xx	00	1110	x	VQABS	-
xx	00	1111	x	VQNEG	-
xx	01	x000	x	VCGT (immediate #0)	-
xx	01	x001	x	VCGE (immediate #0)	-
xx	01	x010	x	VCEQ (immediate #0)	-
xx	01	x011	x	VCLE (immediate #0)	-
xx	01	x100	x	VCLT (immediate #0)	-
xx	01	x110	x	VABS	-
xx	01	x111	x	VNEG	-
xx	01	0101	1	SHA1H	FEAT_SHA1
01	10	1100	1	VCVT (from single-precision to BFloat16, Advanced SIMD)	FEAT_AA32BF16
xx	10	0001	x	VTRN	-
xx	10	0010	x	VUZP	-
xx	10	0011	x	VZIP	-

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields				Instruction page	Feature
size	opc1	opc2	Q		
xx	10	0100	0	VMOVN	-
xx	10	0100	1	VQMOVN, VQMOVUN - Unsigned result variant	-
xx	10	0101	x	VQMOVN, VQMOVUN - Signed result variant	-
xx	10	0110	0	VSHLL	-
xx	10	0111	0	SHA1SU1	FEAT_SHA1
xx	10	0111	1	SHA256SU0	FEAT_SHA256
xx	10	1000	x	VRINTN (Advanced SIMD)	-
xx	10	1001	x	VRINTX (Advanced SIMD)	-
xx	10	1010	x	VRINTA (Advanced SIMD)	-
xx	10	1011	x	VRINTZ (Advanced SIMD)	-
10	10	1100	1	Unallocated.	-
xx	10	1100	0	VCVT (between half-precision and single-precision, Advanced SIMD) - Single-precision to half-precision variant	-
xx	10	1101	x	VRINTM (Advanced SIMD)	-
xx	10	1110	0	VCVT (between half-precision and single-precision, Advanced SIMD) - Half-precision to single-precision variant	-
xx	10	1110	1	Unallocated.	-
xx	10	1111	x	VRINTP (Advanced SIMD)	-
xx	11	000x	x	VCVTA (Advanced SIMD)	-
xx	11	001x	x	VCVTN (Advanced SIMD)	-
xx	11	010x	x	VCVTP (Advanced SIMD)	-
xx	11	011x	x	VCVTM (Advanced SIMD)	-
xx	11	10x0	x	VRECPE	-
xx	11	10x1	x	VRSQRTE	-
11	10	1100	1	Unallocated.	-
xx	11	11xx	x	VCVT (between floating-point and integer, Advanced SIMD)	-

F3.1.9.2 Advanced SIMD duplicate (scalar)

This section describes the encoding of the Advanced SIMD duplicate (scalar) instruction class. The encodings in this section are decoded from Advanced SIMD two registers, or three registers of different lengths.

15 14 13 12	11 10 9 8	7	6	5 4	3 0	15 12	11 10	9 7	6	5	4	3 0
1 1 1 1	1 1 1 1	1	D	1 1	imm4	Vd	1 1	opc	Q	M	0	Vm

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F3-36

Decode fields opc	Instruction page
000	VDUP (scalar)
001	Unallocated.
01x	Unallocated.
1xx	Unallocated.

F3.1.9.3 Advanced SIMD three registers of different lengths

This section describes the encoding of the Advanced SIMD three registers of different lengths instruction class. The encodings in this section are decoded from Advanced SIMD two registers, or three registers of different lengths.

15 14 13	12	11 10 9 8	7	6	5 4	3 0	15 12	11 8	7	6	5	4	3 0
1 1 1	U	1 1 1 1	1	D	!=11 size	Vn	Vd	opc	N	0	M	0	Vm

Table F3-37

Decode fields U	 opc	Instruction page
x	0000	VADDL
x	0001	VADDW
x	0010	VSUBL
0	0100	VADDHN
x	0011	VSUBW
0	0110	VSUBHN
0	1001	VQDMLAL
x	0101	VABAL
0	1011	VQDMLSL
0	1101	VQDMULL
x	0111	VABDL (integer)
x	1000	VMLAL (integer)
x	1010	VMLSL (integer)
1	0100	VRADDHN
1	0110	VRSUBHN
x	11x0	VMULL (integer and polynomial)
1	1001	Unallocated.
1	1011	Unallocated.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields		Instruction page
U	opc	
1	1101	Unallocated.
x	1111	Unallocated.

F3.1.9.4 Advanced SIMD two registers and a scalar

This section describes the encoding of the Advanced SIMD two registers and a scalar instruction class. The encodings in this section are decoded from Advanced SIMD two registers, or three registers of different lengths.

15 14 13	12	11 10 9 8	7	6	5 4	3 0	15 12	11 8	7	6	5	4	3 0
1 1 1	Q	1 1 1 1	1	D	!=11 (size)	Vn	Vd	opc	N	1	M	0	Vm

Table F3-38

Decode fields		Instruction page	Feature
Q	opc		
x	000x	VMLA (by scalar)	-
0	0011	VQDMLAL	-
x	0010	VMLAL (by scalar)	-
0	0111	VQDMLSL	-
x	010x	VMLS (by scalar)	-
0	1011	VQDMULL	-
x	0110	VMLSL (by scalar)	-
x	100x	VMUL (by scalar)	-
1	0011	Unallocated.	-
x	1010	VMULL (by scalar)	-
1	0111	Unallocated.	-
x	1100	VQDMULH	-
x	1101	VQRDMULH	-
1	1011	Unallocated.	-
x	1110	VQRDMLAH	FEAT_RDM
x	1111	VQRDMLSH	FEAT_RDM

F3.1.10 Advanced SIMD shifts and immediate generation

This section describes the encoding of the Advanced SIMD shifts and immediate generation group. The encodings in this section are decoded from Advanced SIMD data-processing.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

15 13	12	11 7	6	5 0	15 7	6	5 4	3 0
111		11111		op0			1	

Table F3-39 Encoding table for the Advanced SIMD shifts and immediate generation group

Decode fields op0	Decode group or instruction page
`000xxxxxxxxxxx0`	Advanced SIMD one register and modified immediate
`!= 000xxxxxxxxxxx0`	Advanced SIMD two registers and shift amount

F3.1.10.1 Advanced SIMD one register and modified immediate

This section describes the encoding of the Advanced SIMD one register and modified immediate instruction class. The encodings in this section are decoded from Advanced SIMD shifts and immediate generation.

15 14 13	12	11 10 9 8	7	6	5 4	3 2 0	15 12	11 8	7	6	5	4	3 0
1 1 1	i	1 1 1 1	1	D	0 0	0 imm3	Vd	cmode	0	Q	op	1	imm4

Table F3-40

Decode fields cmode	op	Instruction page
`0xx0`	`0`	VMOV (immediate) - T1
`0xx0`	`1`	VMVN (immediate) - T1
`0xx1`	`0`	VORR (immediate) - T1
`0xx1`	`1`	VBIC (immediate) - T1
`10x0`	`0`	VMOV (immediate) - T3
`10x0`	`1`	VMVN (immediate) - T2
`10x1`	`0`	VORR (immediate) - T2
`10x1`	`1`	VBIC (immediate) - T2
`11xx`	`0`	VMOV (immediate) - T4
`110x`	`1`	VMVN (immediate) - T3
`1110`	`1`	VMOV (immediate) - T5
`1111`	`1`	Unallocated.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F3.1.10.2 Advanced SIMD two registers and shift amount

This section describes the encoding of the Advanced SIMD two registers and shift amount instruction class. The encodings in this section are decoded from Advanced SIMD shifts and immediate generation.

15 14 13	12	11 10 9 8	7	6	5 3	2 0	15 12	11 8	7	6	5	4	3 0
1 1 1	U	1 1 1 1	1	D	imm3H	imm3L	Vd	opc	L	Q	M	1	Vm

Table F3-41

Decode fields						Instruction page
U	imm3H:L	imm3L	opc	L	Q	
x	!= 0000	xxx	0000	x	x	VSHR
x	!= 0000	xxx	0001	x	x	VSRA
x	!= 0000	000	1010	0	0	VMOVL
x	!= 0000	xxx	0010	x	x	VRSHR
x	!= 0000	xxx	0011	x	x	VRSRA
x	!= 0000	xxx	0111	x	x	VQSHL, VQSHLU (immediate) - 128-bit SIMD vector, signed result variant
x	!= 0000	xxx	1001	0	0	VQSHRN, VQSHRUN - Signed result variant
x	!= 0000	xxx	1001	0	1	VQRSHRN, VQRSHRUN - Signed result variant
x	!= 0000	xxx	1010	0	0	VSHLL
x	!= 0000	xxx	11xx	0	x	VCVT (between floating-point and fixed-point, Advanced SIMD)
0	!= 0000	xxx	0101	x	x	VSHL (immediate)
0	!= 0000	xxx	1000	0	0	VSHRN
0	!= 0000	xxx	1000	0	1	VRSHRN
1	!= 0000	xxx	0100	x	x	VSRI
1	!= 0000	xxx	0101	x	x	VSLI
1	!= 0000	xxx	0110	x	x	VQSHL, VQSHLU (immediate) - 128-bit SIMD vector, unsigned result variant
1	!= 0000	xxx	1000	0	0	VQSHRN, VQSHRUN - Unsigned result variant
1	!= 0000	xxx	1000	0	1	VQRSHRN, VQRSHRUN - Unsigned result variant

F3.1.11 Advanced SIMD and System register load/store and 64-bit move

This section describes the encoding of the Advanced SIMD and System register load/store and 64-bit move group. The encodings in this section are decoded from System register access, Advanced SIMD, and floating-point.

This group has encodings in both the T32 and A32 instruction sets. For information about mappings between the encodings of this group, see About the T32 Advanced SIMD and floating-point instructions and their encoding

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

15 … 9	8 … 5	4 … 0	15 … 12	11	10 … 9	8 … 0
1110110	op0			1	op1	

Table F3-42 Encoding table for the Advanced SIMD and System register load/store and 64-bit move group

Decode fields		Decode group or instruction page
op0	**op1**	
`00x0`	`0x`	Advanced SIMD and floating-point 64-bit move
`00x0`	`11`	System register 64-bit move
`!= 00x0`	`0x`	Advanced SIMD and floating-point load/store
`!= 00x0`	`11`	System register Load/Store
`xxxx`	`10`	Unallocated.

F3.1.11.1 Advanced SIMD and floating-point 64-bit move

This section describes the encoding of the Advanced SIMD and floating-point 64-bit move instruction class. The encodings in this section are decoded from Advanced SIMD and System register load/store and 64-bit move.

15 14 13 12	11 10 9 8	7 6 5 4	3 … 0	15 … 12	11 10	9 8	7 6 5 4	3 … 0
1 1 1 0	1 1 0 0	0 D 0 op	Rt2	Rt	1 0	size	opc2 M o3	Vm

Table F3-43

Decode fields					Instruction page
D	**op**	**size**	**opc2**	**o3**	
`0`	`x`	`xx`	`xx`	`x`	Unallocated.
`1`	`x`	`xx`	`xx`	`0`	Unallocated.
`1`	`x`	`0x`	`00`	`1`	Unallocated.
`1`	`x`	`xx`	`01`	`x`	Unallocated.
`1`	`0`	`10`	`00`	`1`	VMOV (between two general-purpose registers and two single-precision registers) - From general-purpose registers variant
`1`	`0`	`11`	`00`	`1`	VMOV (between two general-purpose registers and a doubleword floating-point register) - From general-purpose registers variant
`1`	`x`	`xx`	`1x`	`x`	Unallocated.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields					Instruction page
D	op	size	opc2	o3	
1	1	10	00	1	VMOV (between two general-purpose registers and two single-precision registers) - To general-purpose registers variant
1	1	11	00	1	VMOV (between two general-purpose registers and a doubleword floating-point register) - To general-purpose registers variant

F3.1.11.2 System register 64-bit move

This section describes the encoding of the System register 64-bit move instruction class. The encodings in this section are decoded from Advanced SIMD and System register load/store and 64-bit move.

15 14 13 12	11 10 9 8	7 6 5 4	3 0	15 12	11 10 9 8	7 4	3 0
1 1 1 0	1 1 0 0 0	D 0 L	Rt2	Rt	1 1 1	opc1	CRm

cp15

Table F3-44

Decode fields		Instruction page
D	L	
0	x	Unallocated.
1	0	MCRR
1	1	MRRC

F3.1.11.3 Advanced SIMD and floating-point load/store

This section describes the encoding of the Advanced SIMD and floating-point load/store instruction class. The encodings in this section are decoded from Advanced SIMD and System register load/store and 64-bit move.

15 14 13 12	11 10 9 8	7 6 5 4	3 0	15 12	11 10 9 8	7 0
1 1 1 0	1 1 0 P	U D W L	Rn	Vd	1 0 size	imm8

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F3-45

Decode fields							Instruction page	Feature
P	U	W	L	Rn	size	imm8		
0	0	1	x	xxxx	xx	xxxxxxxx	Unallocated.	-
0	1	x	x	xxxx	0x	xxxxxxxx	Unallocated.	-
0	1	x	0	xxxx	10	xxxxxxxx	VSTM, VSTMDB, VSTMIA - Increment After variant	-
0	1	x	0	xxxx	11	xxxxxxx0	VSTM, VSTMDB, VSTMIA - Increment After variant	-
0	1	x	0	xxxx	11	xxxxxxx1	FSTMDBX, FSTMIAX - Increment After variant	-
0	1	x	1	xxxx	10	xxxxxxxx	VLDM, VLDMDB, VLDMIA - Increment After variant	-
0	1	x	1	xxxx	11	xxxxxxx0	VLDM, VLDMDB, VLDMIA - Increment After variant	-
0	1	x	1	xxxx	11	xxxxxxx1	FLDM*X (FLDMDBX, FLDMIAX) - Increment After variant	-
1	x	0	0	xxxx	01	xxxxxxxx	VSTR - Half-precision scalar variant	FEAT_FP16
1	x	0	0	xxxx	10	xxxxxxxx	VSTR - Single-precision scalar variant	-
1	x	0	0	xxxx	11	xxxxxxxx	VSTR - Double-precision scalar variant	-
1	x	0	1	!= 1111	01	xxxxxxxx	VLDR (immediate) - Half-precision scalar variant	FEAT_FP16
1	x	0	1	!= 1111	10	xxxxxxxx	VLDR (immediate) - Single-precision scalar variant	-
1	x	0	1	!= 1111	11	xxxxxxxx	VLDR (immediate) - Double-precision scalar variant	-
1	0	1	x	xxxx	0x	xxxxxxxx	Unallocated.	-
1	0	1	0	xxxx	10	xxxxxxxx	VSTM, VSTMDB, VSTMIA - Decrement Before variant	-
1	0	1	0	xxxx	11	xxxxxxx0	VSTM, VSTMDB, VSTMIA - Decrement Before variant	-
1	0	1	0	xxxx	11	xxxxxxx1	FSTMDBX, FSTMIAX - Decrement Before variant	-
1	0	1	1	xxxx	10	xxxxxxxx	VLDM, VLDMDB, VLDMIA - Decrement Before variant	-
1	0	1	1	xxxx	11	xxxxxxx0	VLDM, VLDMDB, VLDMIA - Decrement Before variant	-
1	0	1	1	xxxx	11	xxxxxxx1	FLDM*X (FLDMDBX, FLDMIAX) - Decrement Before variant	-
1	x	0	1	1111	01	xxxxxxxx	VLDR (literal) - Half-precision scalar variant	FEAT_FP16
1	x	0	1	1111	10	xxxxxxxx	VLDR (literal) - Single-precision scalar variant	-
1	x	0	1	1111	11	xxxxxxxx	VLDR (literal) - Double-precision scalar variant	-
1	1	1	x	xxxx	xx	xxxxxxxx	Unallocated.	-

F3.1.11.4 System register Load/Store

This section describes the encoding of the System register Load/Store instruction class. The encodings in this section are decoded from Advanced SIMD and System register load/store and 64-bit move.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

15 14 13 12	11 10 9 8	7	6	5	4	3 0	15 12	11 10 9	8	7 0
1 1 1 0	1 1 0	P	U	D	W	L	Rn	CRd	1 1 1		imm8

cp15

Table F3-46

Decode fields						Instruction page
P:U:W	**D**	**L**	**Rn**	**CRd**	**cp15**	
`!= 000`	`x`	`x`	`xxxx`	`!= 0101`	`0`	Unallocated.
`!= 000`	`0`	`1`	`1111`	`0101`	`0`	LDC (literal)
`!= 000`	`x`	`x`	`xxxx`	`xxxx`	`1`	Unallocated.
`!= 000`	`1`	`x`	`xxxx`	`0101`	`0`	Unallocated.
`0x1`	`0`	`0`	`xxxx`	`0101`	`0`	STC - Post-indexed variant
`0x1`	`0`	`1`	`!= 1111`	`0101`	`0`	LDC (immediate) - Post-indexed variant
`010`	`0`	`0`	`xxxx`	`0101`	`0`	STC - Unindexed variant
`010`	`0`	`1`	`!= 1111`	`0101`	`0`	LDC (immediate) - Unindexed variant
`1x0`	`0`	`0`	`xxxx`	`0101`	`0`	STC - Offset variant
`1x0`	`0`	`1`	`!= 1111`	`0101`	`0`	LDC (immediate) - Offset variant
`1x1`	`0`	`0`	`xxxx`	`0101`	`0`	STC - Pre-indexed variant
`1x1`	`0`	`1`	`!= 1111`	`0101`	`0`	LDC (immediate) - Pre-indexed variant

F3.1.12 Advanced SIMD and System register 32-bit move

This section describes the encoding of the Advanced SIMD and System register 32-bit move group. The encodings in this section are decoded from System register access, Advanced SIMD, and floating-point.

15 8	7 5	4 0	15 12	11	10 8	7 5	4	3 0
11101110	op0			1	op1		1	

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F3-47 Encoding table for the Advanced SIMD and System register 32-bit move group

Decode fields		Decode group or instruction page	Feature
op0	op1		
000	000	Unallocated.	-
000	001	VMOV (between general-purpose register and half-precision)	FEAT_FP16
000	010	VMOV (between general-purpose register and single-precision)	-
001	010	Unallocated.	-
01x	010	Unallocated.	-
10x	010	Unallocated.	-
110	010	Unallocated.	-
111	010	Floating-point move special register	-
xxx	011	Advanced SIMD 8/16/32-bit element move/duplicate	-
xxx	10x	Unallocated.	-
xxx	11x	System register 32-bit move	-

F3.1.12.1 Floating-point move special register

This section describes the encoding of the Floating-point move special register instruction class. The encodings in this section are decoded from Advanced SIMD and System register 32-bit move.

15 14 13 12	11 10 9 8	7 6 5 4	3 0	15 12	11 10 9 8	7	6	5	4	3	2	1	0
1 1 1 0	1 1 1 0	1 1 1 L	reg	Rt	1 0 1 0	(0)	(0)	(0)	1	(0)	(0)	(0)	(0)

Table F3-48

Decode fields	Instruction page
L	
0	VMSR
1	VMRS

F3.1.12.2 Advanced SIMD 8/16/32-bit element move/duplicate

This section describes the encoding of the Advanced SIMD 8/16/32-bit element move/duplicate instruction class. The encodings in this section are decoded from Advanced SIMD and System register 32-bit move.

15 14 13 12	11 10 9 8	7 5	4	3 0	15 12	11 10 9 8	7	6 5	4	3	2	1	0
1 1 1 0	1 1 1 0	opc1	L	Vn	Rt	1 0 1 1	N	opc2	1	(0)	(0)	(0)	(0)

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F3-49

Decode fields			Instruction page
opc1	**L**	**opc2**	
`0xx`	`0`	`xx`	VMOV (general-purpose register to scalar)
`xxx`	`1`	`xx`	VMOV (scalar to general-purpose register)
`1xx`	`0`	`0x`	VDUP (general-purpose register)
`1xx`	`0`	`1x`	Unallocated.

F3.1.12.3 System register 32-bit move

This section describes the encoding of the System register 32-bit move instruction class. The encodings in this section are decoded from Advanced SIMD and System register 32-bit move.

Table F3-50

Decode fields	Instruction page
L	
`0`	MCR
`1`	MRC

F3.1.13 Floating-point data-processing

This section describes the encoding of the Floating-point data-processing group. The encodings in this section are decoded from System register access, Advanced SIMD, and floating-point.

This group has encodings in both the T32 and A32 instruction sets. For information about mappings between the encodings of this group, see About the T32 Advanced SIMD and floating-point instructions and their encoding

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F3-51 Encoding table for the Floating-point data-processing group

Decode fields		Decode group or instruction page
op0	op1	
1x11	1	Floating-point data-processing (two registers)
1x11	0	Floating-point move immediate
!= 1x11	x	Floating-point data-processing (three registers)

F3.1.13.1 Floating-point data-processing (two registers)

This section describes the encoding of the Floating-point data-processing (two registers) instruction class. The encodings in this section are decoded from Floating-point data-processing.

15 14 13 12	11 10 9 8	7	6	5	4	3	2 0	15 12	11	10	9 8	7	6	5	4	3 0
1 1 1 0	1 1 1 0	1	D	1	1	o1	opc2	Vd	1	0	size	o3	1	M	0	Vm

Table F3-52

Decode fields				Instruction page	Feature
o1	opc2	size	o3		
x	xxx	00	x	Unallocated.	-
0	000	01	0	Unallocated.	-
0	000	01	1	VABS - Half-precision scalar variant	FEAT_FP16
0	000	10	0	VMOV (register) - Single-precision scalar variant	-
0	000	10	1	VABS - Single-precision scalar variant	-
0	000	11	0	VMOV (register) - Double-precision scalar variant	-
0	000	11	1	VABS - Double-precision scalar variant	-
0	001	01	0	VNEG - Half-precision scalar variant	FEAT_FP16
0	001	01	1	VSQRT - Half-precision scalar variant	FEAT_FP16
0	001	10	0	VNEG - Single-precision scalar variant	-
0	001	10	1	VSQRT - Single-precision scalar variant	-
0	001	11	0	VNEG - Double-precision scalar variant	-
0	001	11	1	VSQRT - Double-precision scalar variant	-
0	010	01	x	Unallocated.	-
0	010	10	0	VCVTB - Half-precision to single-precision variant	-
0	010	10	1	VCVTT - Half-precision to single-precision variant	-
0	010	11	0	VCVTB - Half-precision to double-precision variant	-

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields				Instruction page	Feature
o1	opc2	size	o3		
0	010	11	1	VCVTT - Half-precision to double-precision variant	-
0	011	01	0	VCVTB (BFloat16)	FEAT_AA32BF16
0	011	01	1	VCVTT (BFloat16)	FEAT_AA32BF16
0	011	10	0	VCVTB - Single-precision to half-precision variant	-
0	011	10	1	VCVTT - Single-precision to half-precision variant	-
0	011	11	0	VCVTB - Double-precision to half-precision variant	-
0	011	11	1	VCVTT - Double-precision to half-precision variant	-
0	100	01	0	VCMP	FEAT_FP16
0	100	01	1	VCMPE	FEAT_FP16
0	100	10	0	VCMP	-
0	100	10	1	VCMPE	-
0	100	11	0	VCMP	-
0	100	11	1	VCMPE	-
0	101	01	0	VCMP	FEAT_FP16
0	101	01	1	VCMPE	FEAT_FP16
0	101	10	0	VCMP	-
0	101	10	1	VCMPE	-
0	101	11	0	VCMP	-
0	101	11	1	VCMPE	-
0	110	01	0	VRINTR - Half-precision scalar variant	FEAT_FP16
0	110	01	1	VRINTZ (floating-point) - Half-precision scalar variant	FEAT_FP16
0	110	10	0	VRINTR - Single-precision scalar variant	-
0	110	10	1	VRINTZ (floating-point) - Single-precision scalar variant	-
0	110	11	0	VRINTR - Double-precision scalar variant	-
0	110	11	1	VRINTZ (floating-point) - Double-precision scalar variant	-
0	111	01	0	VRINTX (floating-point) - Half-precision scalar variant	FEAT_FP16
0	111	01	1	Unallocated.	-
0	111	10	0	VRINTX (floating-point) - Single-precision scalar variant	-
0	111	10	1	VCVT (between double-precision and single-precision) - Single-precision to double-precision variant	-
0	111	11	0	VRINTX (floating-point) - Double-precision scalar variant	-
0	111	11	1	VCVT (between double-precision and single-precision) - Double-precision to single-precision variant	-
1	000	01	x	VCVT (integer to floating-point, floating-point) - Half-precision scalar variant	FEAT_FP16
1	000	10	x	VCVT (integer to floating-point, floating-point) - Single-precision scalar variant	-
1	000	11	x	VCVT (integer to floating-point, floating-point) - Double-precision scalar variant	-
1	001	01	x	Unallocated.	-
1	001	10	x	Unallocated.	-

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields				Instruction page	Feature
o1	opc2	size	o3		
1	001	11	0	Unallocated.	-
1	001	11	1	VJCVT	FEAT_JSCVT
1	01x	01	x	VCVT (between floating-point and fixed-point, floating-point)	FEAT_FP16
1	01x	10	x	VCVT (between floating-point and fixed-point, floating-point)	-
1	01x	11	x	VCVT (between floating-point and fixed-point, floating-point)	-
1	100	01	0	VCVTR	FEAT_FP16
1	100	01	1	VCVT (floating-point to integer, floating-point)	FEAT_FP16
1	100	10	0	VCVTR	-
1	100	10	1	VCVT (floating-point to integer, floating-point)	-
1	100	11	0	VCVTR	-
1	100	11	1	VCVT (floating-point to integer, floating-point)	-
1	101	01	0	VCVTR	FEAT_FP16
1	101	01	1	VCVT (floating-point to integer, floating-point)	FEAT_FP16
1	101	10	0	VCVTR	-
1	101	10	1	VCVT (floating-point to integer, floating-point)	-
1	101	11	0	VCVTR	-
1	101	11	1	VCVT (floating-point to integer, floating-point)	-
1	11x	01	x	VCVT (between floating-point and fixed-point, floating-point)	FEAT_FP16
1	11x	10	x	VCVT (between floating-point and fixed-point, floating-point)	-
1	11x	11	x	VCVT (between floating-point and fixed-point, floating-point)	-

F3.1.13.2 Floating-point move immediate

This section describes the encoding of the Floating-point move immediate instruction class. The encodings in this section are decoded from Floating-point data-processing.

15 14 13 12	11 10 9 8	7	6	5 4	3 0	15 12	11 10	9 8	7	6	5	4	3 0
1 1 1 0	1 1 1 0	1	D	1 1	imm4H	Vd	1 0	size	(0)	0	(0)	0	imm4L

Table F3-53

Decode fields	Instruction page	Feature
size		
00	Unallocated.	-
01	VMOV (immediate) - Half-precision scalar variant	FEAT_FP16
10	VMOV (immediate) - Single-precision scalar variant	-

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields size	Instruction page	Feature
11	VMOV (immediate) - Double-precision scalar variant	-

F3.1.13.3 Floating-point data-processing (three registers)

This section describes the encoding of the Floating-point data-processing (three registers) instruction class. The encodings in this section are decoded from Floating-point data-processing.

15 14 13 12	11 10 9 8	7	6	5 4	3 0	15 12	11 10	9 8	7	6	5	4	3 0
1 1 1 0	1 1 1 0	o0	D	o1	Vn	Vd	1 0	size	N	o2	M	0	Vm

Table F3-54

Decode fields o0:o1	size	o2	Instruction page	Feature
!= 111	00	x	Unallocated.	-
000	01	0	VMLA (floating-point) - Half-precision scalar variant	FEAT_FP16
000	01	1	VMLS (floating-point) - Half-precision scalar variant	FEAT_FP16
000	10	0	VMLA (floating-point) - Single-precision scalar variant	-
000	10	1	VMLS (floating-point) - Single-precision scalar variant	-
000	11	0	VMLA (floating-point) - Double-precision scalar variant	-
000	11	1	VMLS (floating-point) - Double-precision scalar variant	-
001	01	0	VNMLS - Half-precision scalar variant	FEAT_FP16
001	01	1	VNMLA - Half-precision scalar variant	FEAT_FP16
001	10	0	VNMLS - Single-precision scalar variant	-
001	10	1	VNMLA - Single-precision scalar variant	-
001	11	0	VNMLS - Double-precision scalar variant	-
001	11	1	VNMLA - Double-precision scalar variant	-
010	01	0	VMUL (floating-point) - Half-precision scalar variant	FEAT_FP16
010	01	1	VNMUL - Half-precision scalar variant	FEAT_FP16
010	10	0	VMUL (floating-point) - Single-precision scalar variant	-
010	10	1	VNMUL - Single-precision scalar variant	-
010	11	0	VMUL (floating-point) - Double-precision scalar variant	-
010	11	1	VNMUL - Double-precision scalar variant	-
011	01	0	VADD (floating-point) - Half-precision scalar variant	FEAT_FP16
011	01	1	VSUB (floating-point) - Half-precision scalar variant	FEAT_FP16
011	10	0	VADD (floating-point) - Single-precision scalar variant	-

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields			Instruction page	Feature
o0:o1	size	o2		
011	10	1	VSUB (floating-point) - Single-precision scalar variant	-
011	11	0	VADD (floating-point) - Double-precision scalar variant	-
011	11	1	VSUB (floating-point) - Double-precision scalar variant	-
100	01	0	VDIV - Half-precision scalar variant	FEAT_FP16
100	10	0	VDIV - Single-precision scalar variant	-
100	11	0	VDIV - Double-precision scalar variant	-
101	01	0	VFNMS - Half-precision scalar variant	FEAT_FP16
101	01	1	VFNMA - Half-precision scalar variant	FEAT_FP16
101	10	0	VFNMS - Single-precision scalar variant	-
101	10	1	VFNMA - Single-precision scalar variant	-
101	11	0	VFNMS - Double-precision scalar variant	-
101	11	1	VFNMA - Double-precision scalar variant	-
110	01	0	VFMA - Half-precision scalar variant	FEAT_FP16
110	01	1	VFMS - Half-precision scalar variant	FEAT_FP16
110	10	0	VFMA - Single-precision scalar variant	-
110	10	1	VFMS - Single-precision scalar variant	-
110	11	0	VFMA - Double-precision scalar variant	-
110	11	1	VFMS - Double-precision scalar variant	-

F3.1.14 Additional Advanced SIMD and floating-point instructions

This section describes the encoding of the Additional Advanced SIMD and floating-point instructions group. The encodings in this section are decoded from System register access, Advanced SIMD, and floating-point.

This decode also imposes the constraint:

- `op0<2:1> != 11`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F3-55 Encoding table for the Additional Advanced SIMD and floating-point instructions group

Decode fields						Decode group or instruction page
op0	op1	op2	op3	op4	op5	
0xx	xxxxxx	x	0x	x	x	Advanced SIMD three registers of the same length extension
100	xxxxxx	0	!= 00	0	0	Floating-point conditional select
101	00xxxx	0	!= 00	x	0	Floating-point minNum/maxNum
101	110000	0	!= 00	1	0	Floating-point extraction and insertion
101	111xxx	0	!= 00	1	0	Floating-point directed convert to integer
10x	xxxxxx	0	00	x	x	Advanced SIMD and floating-point multiply with accumulate
10x	xxxxxx	1	0x	x	x	Advanced SIMD and floating-point dot product

F3.1.14.1 Advanced SIMD three registers of the same length extension

This section describes the encoding of the Advanced SIMD three registers of the same length extension instruction class. The encodings in this section are decoded from Additional Advanced SIMD and floating-point instructions.

Table F3-56

Decode fields						Instruction page	Feature
op1	op2	op3	op4	Q	U		
x1	0x	0	0	0	0	VCADD - 64-bit SIMD vector variant	FEAT_FCMA
x1	0x	0	0	0	1	Unallocated.	-
x1	0x	0	0	1	0	VCADD - 128-bit SIMD vector variant	FEAT_FCMA
x1	0x	0	0	1	1	Unallocated.	-
00	0x	0	0	x	x	Unallocated.	-
00	0x	0	1	x	x	Unallocated.	-
00	00	1	0	0	0	Unallocated.	-
00	00	1	0	0	1	Unallocated.	-
00	00	1	0	1	0	VMMLA	FEAT_AA32BF16
00	00	1	0	1	1	Unallocated.	-
00	00	1	1	0	0	VDOT (vector) - 64-bit SIMD vector variant	FEAT_AA32BF16
00	00	1	1	0	1	Unallocated.	-
00	00	1	1	1	0	VDOT (vector) - 128-bit SIMD vector variant	FEAT_AA32BF16

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields						Instruction page	Feature
op1	op2	op3	op4	Q	U		
00	00	1	1	1	1	Unallocated.	-
00	01	1	0	x	x	Unallocated.	-
00	01	1	1	x	x	Unallocated.	-
00	10	0	0	x	1	VFMAL (vector)	FEAT_FHM
00	10	0	1	x	x	Unallocated.	-
00	10	1	0	0	x	Unallocated.	-
00	10	1	0	1	0	VSMMLA	FEAT_AA32I8MM
00	10	1	0	1	1	VUMMLA	FEAT_AA32I8MM
00	10	1	1	0	0	VSDOT (vector) - 64-bit SIMD vector variant	FEAT_DotProd
00	10	1	1	0	1	VUDOT (vector) - 64-bit SIMD vector variant	FEAT_DotProd
00	10	1	1	1	0	VSDOT (vector) - 128-bit SIMD vector variant	FEAT_DotProd
00	10	1	1	1	1	VUDOT (vector) - 128-bit SIMD vector variant	FEAT_DotProd
00	11	0	0	x	1	VFMAB, VFMAT (BFloat16, vector)	FEAT_AA32BF16
00	11	0	1	x	x	Unallocated.	-
00	11	1	0	x	x	Unallocated.	-
00	11	1	1	x	x	Unallocated.	-
01	10	0	0	x	1	VFMSL (vector)	FEAT_FHM
01	10	0	1	x	x	Unallocated.	-
01	10	1	0	0	x	Unallocated.	-
01	10	1	0	1	0	VUSMMLA	FEAT_AA32I8MM
01	10	1	0	1	1	Unallocated.	-
01	10	1	1	0	0	VUSDOT (vector) - 64-bit SIMD vector variant	FEAT_AA32I8MM
01	10	1	1	x	1	Unallocated.	-
01	10	1	1	1	0	VUSDOT (vector) - 128-bit SIMD vector variant	FEAT_AA32I8MM
01	11	0	1	x	x	Unallocated.	-
01	11	1	0	x	x	Unallocated.	-
01	11	1	1	x	x	Unallocated.	-
xx	1x	0	0	x	0	VCMLA	FEAT_FCMA
10	11	0	1	x	x	Unallocated.	-
10	11	1	0	x	x	Unallocated.	-
10	11	1	1	x	x	Unallocated.	-
11	11	0	1	x	x	Unallocated.	-
11	11	1	0	x	x	Unallocated.	-
11	11	1	1	x	x	Unallocated.	-

F3.1.14.2 Floating-point conditional select

This section describes the encoding of the Floating-point conditional select instruction class. The encodings in this section are decoded from Additional Advanced SIMD and floating-point instructions.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

15 14 13 12	11 10 9 8	7 6 5 4	3 0	15 12	11 10	9 8	7	6	5	4	3 0
1 1 1 1	1 1 1 0	0 D cc	Vn	Vd	1 0	!=00	N	0	M	0	Vm
						size					

Table F3-57

Decode fields	Instruction page	Feature
size		
01	VSELEQ, VSELGE, VSELGT, VSELVS - Greater than, half-precision scalar variant	FEAT_FP16
10	VSELEQ, VSELGE, VSELGT, VSELVS - Greater than, single-precision scalar variant	-
11	VSELEQ, VSELGE, VSELGT, VSELVS - Greater than, double-precision scalar variant	-

F3.1.14.3 Floating-point minNum/maxNum

This section describes the encoding of the Floating-point minNum/maxNum instruction class. The encodings in this section are decoded from Additional Advanced SIMD and floating-point instructions.

15 14 13 12	11 10 9 8	7 6 5 4	3 0	15 12	11 10	9 8	7	6	5	4	3 0
1 1 1 1	1 1 1 0	1 D 0 0	Vn	Vd	1 0	!=00	N	op	M	0	Vm
						size					

Table F3-58

Decode fields		Instruction page	Feature
size	op		
01	0	VMAXNM - Half-precision scalar variant	FEAT_FP16
01	1	VMINNM - Half-precision scalar variant	FEAT_FP16
10	0	VMAXNM - Single-precision scalar variant	-
10	1	VMINNM - Single-precision scalar variant	-
11	0	VMAXNM - Double-precision scalar variant	-
11	1	VMINNM - Double-precision scalar variant	-

F3.1.14.4 Floating-point extraction and insertion

This section describes the encoding of the Floating-point extraction and insertion instruction class. The encodings in this section are decoded from Additional Advanced SIMD and floating-point instructions.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

15 14 13 12	11 10 9 8	7	6	5 4	3 2 1 0	15 12	11 10	9 8	7	6	5	4	3 0
1 1 1 1	1 1 1 0	1	D	1 1	0 0 0 0	Vd	1 0	!=00	op	1	M	0	Vm
								size					

Table F3-59

Decode fields		Instruction page	Feature
size	**op**		
01	x	Unallocated.	-
10	0	VMOVX	FEAT_FP16
10	1	VINS	FEAT_FP16
11	x	Unallocated.	-

F3.1.14.5 Floating-point directed convert to integer

This section describes the encoding of the Floating-point directed convert to integer instruction class. The encodings in this section are decoded from Additional Advanced SIMD and floating-point instructions.

15 14 13 12	11 10 9 8	7	6	5 4	3	2	1 0	15 12	11 10	9 8	7	6	5	4	3 0
1 1 1 1	1 1 1 0	1	D	1 1	1	o1	RM	Vd	1 0	!=00	op	1	M	0	Vm
										size					

Table F3-60

Decode fields				Instruction page	Feature
o1	**RM**	**size**	**op**		
0	xx	!= 00	1	Unallocated.	-
0	00	01	0	VRINTA (floating-point) - Half-precision scalar variant	FEAT_FP16
0	00	10	0	VRINTA (floating-point) - Single-precision scalar variant	-
0	00	11	0	VRINTA (floating-point) - Double-precision scalar variant	-
0	01	01	0	VRINTN (floating-point) - Half-precision scalar variant	FEAT_FP16
0	01	10	0	VRINTN (floating-point) - Single-precision scalar variant	-
0	01	11	0	VRINTN (floating-point) - Double-precision scalar variant	-
0	10	01	0	VRINTP (floating-point) - Half-precision scalar variant	FEAT_FP16
0	10	10	0	VRINTP (floating-point) - Single-precision scalar variant	-
0	10	11	0	VRINTP (floating-point) - Double-precision scalar variant	-

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields				Instruction page	Feature
o1	RM	size	op		
0	11	01	0	VRINTM (floating-point) - Half-precision scalar variant	FEAT_FP16
0	11	10	0	VRINTM (floating-point) - Single-precision scalar variant	-
0	11	11	0	VRINTM (floating-point) - Double-precision scalar variant	-
1	00	01	x	VCVTA (floating-point) - Half-precision scalar variant	FEAT_FP16
1	00	10	x	VCVTA (floating-point) - Single-precision scalar variant	-
1	00	11	x	VCVTA (floating-point) - Double-precision scalar variant	-
1	01	01	x	VCVTN (floating-point) - Half-precision scalar variant	FEAT_FP16
1	01	10	x	VCVTN (floating-point) - Single-precision scalar variant	-
1	01	11	x	VCVTN (floating-point) - Double-precision scalar variant	-
1	10	01	x	VCVTP (floating-point) - Half-precision scalar variant	FEAT_FP16
1	10	10	x	VCVTP (floating-point) - Single-precision scalar variant	-
1	10	11	x	VCVTP (floating-point) - Double-precision scalar variant	-
1	11	01	x	VCVTM (floating-point) - Half-precision scalar variant	FEAT_FP16
1	11	10	x	VCVTM (floating-point) - Single-precision scalar variant	-
1	11	11	x	VCVTM (floating-point) - Double-precision scalar variant	-

F3.1.14.6 Advanced SIMD and floating-point multiply with accumulate

This section describes the encoding of the Advanced SIMD and floating-point multiply with accumulate instruction class. The encodings in this section are decoded from Additional Advanced SIMD and floating-point instructions.

Table F3-61

Decode fields				Instruction page	Feature
op1	op2	Q	U		
0	xx	x	0	VCMLA (by element) - 128-bit SIMD vector of half-precision floating-point variant	FEAT_FCMA
0	00	x	1	VFMAL (by scalar)	FEAT_FHM
0	01	x	1	VFMSL (by scalar)	FEAT_FHM
0	10	x	1	Unallocated.	-
0	11	x	1	VFMAB, VFMAT (BFloat16, by scalar)	FEAT_AA32BF16
1	xx	0	0	VCMLA (by element) - 64-bit SIMD vector of single-precision floating-point variant	FEAT_FCMA

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields				Instruction page	Feature
op1	op2	Q	U		
1	xx	x	1	Unallocated.	-
1	xx	1	0	VCMLA (by element) - 128-bit SIMD vector of single-precision floating-point variant	FEAT_FCMA

F3.1.14.7 Advanced SIMD and floating-point dot product

This section describes the encoding of the Advanced SIMD and floating-point dot product instruction class. The encodings in this section are decoded from Additional Advanced SIMD and floating-point instructions.

Table F3-62

Decode fields					Instruction page	Feature
op1	op2	op4	Q	U		
0	00	0	x	x	Unallocated.	-
0	00	1	0	0	VDOT (by element) - 64-bit SIMD vector variant	FEAT_AA32BF16
0	00	1	x	1	Unallocated.	-
0	00	1	1	0	VDOT (by element) - 128-bit SIMD vector variant	FEAT_AA32BF16
0	01	0	x	x	Unallocated.	-
0	10	0	x	x	Unallocated.	-
0	10	1	0	0	VSDOT (by element) - 64-bit SIMD vector variant	FEAT_DotProd
0	10	1	0	1	VUDOT (by element) - 64-bit SIMD vector variant	FEAT_DotProd
0	10	1	1	0	VSDOT (by element) - 128-bit SIMD vector variant	FEAT_DotProd
0	10	1	1	1	VUDOT (by element) - 128-bit SIMD vector variant	FEAT_DotProd
0	11	x	x	x	Unallocated.	-
1	xx	0	x	x	Unallocated.	-
1	00	1	0	0	VUSDOT (by element) - 64-bit SIMD vector variant	FEAT_AA32I8MM
1	00	1	0	1	VSUDOT (by element) - 64-bit SIMD vector variant	FEAT_AA32I8MM
1	00	1	1	0	VUSDOT (by element) - 128-bit SIMD vector variant	FEAT_AA32I8MM
1	00	1	1	1	VSUDOT (by element) - 128-bit SIMD vector variant	FEAT_AA32I8MM
1	01	1	x	x	Unallocated.	-
1	1x	1	x	x	Unallocated.	-

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F3.1.15 Load/store dual, load/store exclusive, load-acquire/store-release, and table branch

This section describes the encoding of the Load/store dual, load/store exclusive, load-acquire/store-release, and table branch group. The encodings in this section are decoded from 32-bit.

This decode also imposes the constraint:

- `op0<1> == 1`.

Table F3-63 Encoding table for the Load/store dual, load/store exclusive, load-acquire/store-release, and table branch group

Decode fields				Decode group or instruction page
op0	op1	op2	op3	
0010	x	xxxx	xxx	Load/store exclusive
0110	0	xxxx	000	Unallocated.
0110	1	xxxx	000	TBB, TBH
0110	x	xxxx	01x	Load/store exclusive byte/half/dual
0110	x	xxxx	1xx	Load-acquire / Store-release
0x11	x	!= 1111	xxx	Load/store dual (immediate, post-indexed)
1x10	x	!= 1111	xxx	Load/store dual (immediate)
1x11	x	!= 1111	xxx	Load/store dual (immediate, pre-indexed)
!= 0xx0	x	1111	xxx	LDRD (literal)

F3.1.15.1 Load/store exclusive

This section describes the encoding of the Load/store exclusive instruction class. The encodings in this section are decoded from Load/store dual, load/store exclusive, load-acquire/store-release, and table branch.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F3-64

Decode fields	Instruction page
L	
0	STREX
1	LDREX

F3.1.15.2 Load/store exclusive byte/half/dual

This section describes the encoding of the Load/store exclusive byte/half/dual instruction class. The encodings in this section are decoded from Load/store dual, load/store exclusive, load-acquire/store-release, and table branch.

15 14 13 12	11 10 9 8	7 6 5 4	3 0	15 12	11 8	7 6	5 4	3 0
1 1 1 0	1 0 0 0	1 1 0 L	Rn	Rt	Rt2	0 1	sz	Rd

Table F3-65

Decode fields		Instruction page
L	**sz**	
0	00	STREXB
0	01	STREXH
0	10	Unallocated.
0	11	STREXD
1	00	LDREXB
1	01	LDREXH
1	10	Unallocated.
1	11	LDREXD

F3.1.15.3 Load-acquire / Store-release

This section describes the encoding of the Load-acquire / Store-release instruction class. The encodings in this section are decoded from Load/store dual, load/store exclusive, load-acquire/store-release, and table branch.

15 14 13 12	11 10 9 8	7 6 5 4	3 0	15 12	11 8	7	6	5 4	3 0
1 1 1 0	1 0 0 0	1 1 0 L	Rn	Rt	Rt2	1	op	sz	Rd

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F3-66

Decode fields			Instruction page
L	**op**	**sz**	
0	0	00	STLB
0	0	01	STLH
0	0	10	STL
0	0	11	Unallocated.
0	1	00	STLEXB
0	1	01	STLEXH
0	1	10	STLEX
0	1	11	STLEXD
1	0	00	LDAB
1	0	01	LDAH
1	0	10	LDA
1	0	11	Unallocated.
1	1	00	LDAEXB
1	1	01	LDAEXH
1	1	10	LDAEX
1	1	11	LDAEXD

F3.1.15.4 Load/store dual (immediate, post-indexed)

This section describes the encoding of the Load/store dual (immediate, post-indexed) instruction class. The encodings in this section are decoded from Load/store dual, load/store exclusive, load-acquire/store-release, and table branch.

15 14 13 12	11 10 9 8	7	6 5	4	3 0	15 12	11 8	7 0
1 1 1 0	1 0 0 0	U	1 1	L	!=1111	Rt	Rt2	imm8
					Rn			

Table F3-67

Decode fields	Instruction page
L	
0	STRD (immediate)
1	LDRD (immediate)

F3.1.15.5 Load/store dual (immediate)

This section describes the encoding of the Load/store dual (immediate) instruction class. The encodings in this section are decoded from Load/store dual, load/store exclusive, load-acquire/store-release, and table branch.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

15 14 13 12	11 10 9 8	7	6 5 4	3 0	15 12	11 8	7 0
1 1 1 0	1 0 0 1	U	1 0	L	!=1111	Rt	Rt2	imm8
				Rn			

Table F3-68

Decode fields	Instruction page
L	
0	STRD (immediate)
1	LDRD (immediate)

F3.1.15.6 Load/store dual (immediate, pre-indexed)

This section describes the encoding of the Load/store dual (immediate, pre-indexed) instruction class. The encodings in this section are decoded from Load/store dual, load/store exclusive, load-acquire/store-release, and table branch.

15 14 13 12	11 10 9 8	7	6 5 4	3 0	15 12	11 8	7 0
1 1 1 0	1 0 0 1	U	1 1	L	!=1111	Rt	Rt2	imm8
				Rn			

Table F3-69

Decode fields	Instruction page
L	
0	STRD (immediate)
1	LDRD (immediate)

F3.1.16 Branches and miscellaneous control

This section describes the encoding of the Branches and miscellaneous control group. The encodings in this section are decoded from 32-bit.

15	11	10	9 6	5 4	3 0	15	14 12	11	10 8	7 6	5	4 0
11110		op1	op2		1	op3		op4			
op0										op5		

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F3-70 Encoding table for the Branches and miscellaneous control group

Decode fields						Decode group or instruction page
op0	op1	op2	op3	op4	op5	
0	1110	0x	0x0	xxx	0	MSR (register)
0	1110	0x	0x0	xxx	1	MSR (Banked register)
0	1110	10	0x0	000	x	Hints
0	1110	10	0x0	!= 000	x	Change processor state
0	1110	11	0x0	xxx	x	Miscellaneous system
0	1111	00	0x0	xxx	x	BXJ
0	1111	01	0x0	xxx	x	Exception return
0	1111	1x	0x0	xxx	0	MRS
0	1111	1x	0x0	xxx	1	MRS (Banked register)
1	1110	00	000	xxx	x	DCPS
1	1110	00	010	xxx	x	Unallocated.
1	1110	01	0x0	xxx	x	Unallocated.
1	1110	1x	0x0	xxx	x	Unallocated.
1	1111	0x	0x0	xxx	x	Unallocated.
1	1111	1x	0x0	xxx	x	Exception generation
x	!= 111x	x	0x0	xxx	x	B - T3 variant
x	xxxx	x	0x1	xxx	x	B - T4 variant
x	xxxx	x	1x0	xxx	x	BL, BLX (immediate) - T2 variant
x	xxxx	x	1x1	xxx	x	BL, BLX (immediate) - T1 variant

F3.1.16.1 Hints

This section describes the encoding of the Hints instruction class. The encodings in this section are decoded from Branches and miscellaneous control.

15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0	15 14 13 12	11 10 9 8	7 4	3 0
1 1 1 1	0 0 1 1	1 0 1 0	(1)(1)(1)(1)	1 0 (0) 0	(0) 0 0 0	hint	option

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F3-71

Decode fields		Instruction page	Feature
hint	option		
0000	0000	NOP	-
0000	0001	YIELD	-
0000	0010	WFE	-
0000	0011	WFI	-
0000	0100	SEV	-
0000	0101	SEVL	-
0000	011x	Reserved hint, behaves as NOP.	-
0000	1xxx	Reserved hint, behaves as NOP.	-
0001	0000	ESB	FEAT_RAS
0001	0001	Reserved hint, behaves as NOP.	-
0001	0010	TSB	FEAT_TRF
0001	0011	Reserved hint, behaves as NOP.	-
0001	0100	CSDB	-
0001	0101	Reserved hint, behaves as NOP.	-
0001	0110	CLRBHB	FEAT_CLRBHB
0001	0111	Reserved hint, behaves as NOP.	-
0001	1xxx	Reserved hint, behaves as NOP.	-
001x	xxxx	Reserved hint, behaves as NOP.	-
01xx	xxxx	Reserved hint, behaves as NOP.	-
10xx	xxxx	Reserved hint, behaves as NOP.	-
110x	xxxx	Reserved hint, behaves as NOP.	-
1110	xxxx	Reserved hint, behaves as NOP.	-
1111	xxxx	DBG	-

F3.1.16.2 Change processor state

This section describes the encoding of the Change processor state instruction class. The encodings in this section are decoded from Branches and miscellaneous control.

15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0	15 14 13 12	11 10 9 8	7 6 5 4	0
1 1 1 1	0 0 1 1	1 0 1 0	(1) (1) (1) (1)	1 0 (0) 0	(0) imod M	A I F	mode

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F3-72

Decode fields		Instruction page
imod	**M**	
00	1	CPS, CPSID, CPSIE - Change mode variant
01	x	Unallocated.
10	x	CPS, CPSID, CPSIE - Interrupt enable and change mode variant
11	x	CPS, CPSID, CPSIE - Interrupt disable and change mode variant

F3.1.16.3 Miscellaneous system

This section describes the encoding of the Miscellaneous system instruction class. The encodings in this section are decoded from Branches and miscellaneous control.

15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0	15 14 13 12	11 10 9 8	7 4	3 0
1 1 1 1	0 0 1 1	1 0 1 1	(1)(1)(1)(1)	1 0 (0) 0	(1)(1)(1)(1)	opc	option

Table F3-73

Decode fields		Instruction page	Feature
opc	**option**		
000x	xxxx	Unallocated.	-
0010	xxxx	CLREX	-
0011	xxxx	Unallocated.	-
0100	!= 0x00	DSB	-
0100	0000	SSBB	-
0100	0100	PSSBB	-
0101	xxxx	DMB	-
0110	xxxx	ISB	-
0111	xxxx	SB	FEAT_SB
1xxx	xxxx	Unallocated.	-

F3.1.16.4 Exception return

This section describes the encoding of the Exception return instruction class. The encodings in this section are decoded from Branches and miscellaneous control.

15 14 13 12	11 10 9 8	7 6 5 4	3 0	15 14 13 12	11 10 9 8	7 0
1 1 1 1	0 0 1 1	1 1 0 1	Rn	1 0 (0) 0	(1)(1)(1)(1)	imm8

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F3-74

Decode fields Rn:imm8	Instruction page
`!= 111000000000`	SUB, SUBS (immediate)
`111000000000`	ERET

F3.1.16.5 DCPS

This section describes the encoding of the DCPS instruction class. The encodings in this section are decoded from Branches and miscellaneous control.

15	14	13	12	11	10	9	8	7	6	5	4	3 … 0	15	14	13	12	11 … 2	1	0
1	1	1	1	0	1	1	1	1	0	0	0	imm4	1	0	0	0	imm10	opt	

Table F3-75

Decode fields imm4	imm10	opt	Instruction page
`!= 1111`	`xxxxxxxxxx`	`xx`	Unallocated.
`1111`	`!= 0000000000`	`xx`	Unallocated.
`1111`	`0000000000`	`00`	Unallocated.
`1111`	`0000000000`	`01`	DCPS1
`1111`	`0000000000`	`10`	DCPS2
`1111`	`0000000000`	`11`	DCPS3

F3.1.16.6 Exception generation

This section describes the encoding of the Exception generation instruction class. The encodings in this section are decoded from Branches and miscellaneous control.

15	14	13	12	11	10	9	8	7	6	5	4	3 … 0	15	14	13	12	11 … 0
1	1	1	1	0	1	1	1	1	1	1	o1	imm4	1	0	o2	0	imm12

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F3-76

Decode fields		Instruction page
o1	**o2**	
0	0	HVC
0	1	Unallocated.
1	0	SMC
1	1	UDF

F3.1.17 Data-processing (plain binary immediate)

This section describes the encoding of the Data-processing (plain binary immediate) group. The encodings in this section are decoded from 32-bit.

Table F3-77 Encoding table for the Data-processing (plain binary immediate) group

Decode fields		Decode group or instruction page
op0	**op1**	
0	0x	Data-processing (simple immediate)
0	10	Move Wide (16-bit immediate)
0	11	Unallocated.
1	xx	Saturate, Bitfield

F3.1.17.1 Data-processing (simple immediate)

This section describes the encoding of the Data-processing (simple immediate) instruction class. The encodings in this section are decoded from Data-processing (plain binary immediate).

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F3-78

Decode fields			Instruction page
o1	**o2**	**Rn**	
0	0	!= 11x1	ADD, ADDS (immediate)
0	0	1101	ADD, ADDS (SP plus immediate)
0	0	1111	ADR - T3
0	1	xxxx	Unallocated.
1	0	xxxx	Unallocated.
1	1	!= 11x1	SUB, SUBS (immediate)
1	1	1101	SUB, SUBS (SP minus immediate)
1	1	1111	ADR - T2

F3.1.17.2 Move Wide (16-bit immediate)

This section describes the encoding of the Move Wide (16-bit immediate) instruction class. The encodings in this section are decoded from Data-processing (plain binary immediate).

15 14 13 12	11 10 9 8	7 6 5 4	3 0	15 14 12	11 8	7 0
1 1 1 1 0	i 1 0	o1 1 0 0	imm4	0 imm3	Rd	imm8

Table F3-79

Decode fields	Instruction page
o1	
0	MOV, MOVS (immediate)
1	MOVT

F3.1.17.3 Saturate, Bitfield

This section describes the encoding of the Saturate, Bitfield instruction class. The encodings in this section are decoded from Data-processing (plain binary immediate).

15 14 13 12	11 10 9 8	7 5 4	3 0	15 14 12	11 8	7 6 5 4	0
1 1 1 1 0	(0) 1 1	op1 0	Rn	0 imm3	Rd	imm2 (0)	widthm1

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F3-80

Decode fields			Instruction page
op1	Rn	imm3:imm2	
000	xxxx	xxxxx	SSAT - Logical shift left variant
001	xxxx	!= 00000	SSAT - Arithmetic shift right variant
001	xxxx	00000	SSAT16
010	xxxx	xxxxx	SBFX
011	!= 1111	xxxxx	BFI
011	1111	xxxxx	BFC
100	xxxx	xxxxx	USAT - Logical shift left variant
101	xxxx	!= 00000	USAT - Arithmetic shift right variant
101	xxxx	00000	USAT16
110	xxxx	xxxxx	UBFX
111	xxxx	xxxxx	Unallocated.

F3.1.18 Advanced SIMD element or structure load/store

This section describes the encoding of the Advanced SIMD element or structure load/store group. The encodings in this section are decoded from 32-bit.

This group has encodings in both the T32 and A32 instruction sets. For information about mappings between the encodings of this group, see About the T32 Advanced SIMD and floating-point instructions and their encoding

Table F3-81 Encoding table for the Advanced SIMD element or structure load/store group

Decode fields		Decode group or instruction page
op0	op1	
0	xx	Advanced SIMD load/store multiple structures
1	11	Advanced SIMD load single structure to all lanes
1	!= 11	Advanced SIMD load/store single structure to one lane

F3.1.18.1 Advanced SIMD load/store multiple structures

This section describes the encoding of the Advanced SIMD load/store multiple structures instruction class. The encodings in this section are decoded from Advanced SIMD element or structure load/store.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

15	14	13	12	11	10	9	8	7	6	5	4	3 … 0	15 … 12	11 … 8	7	6	5	4	3 … 0
1	1	1	1	1	0	0	1	0	D	L	0	Rn	Vd	itype	size	align	align	align	Rm

Table F3-82

Decode fields			Instruction page
L	**itype**	**Rm**	
0	000x	!= 11x1	VST4 (multiple 4-element structures)
0	000x	1101	VST4 (multiple 4-element structures)
0	000x	1111	VST4 (multiple 4-element structures)
0	0010	!= 11x1	VST1 (multiple single elements)
0	0010	1101	VST1 (multiple single elements)
0	0010	1111	VST1 (multiple single elements)
0	0011	!= 11x1	VST2 (multiple 2-element structures)
0	0011	1101	VST2 (multiple 2-element structures)
0	0011	1111	VST2 (multiple 2-element structures)
0	010x	!= 11x1	VST3 (multiple 3-element structures)
0	010x	1101	VST3 (multiple 3-element structures)
0	010x	1111	VST3 (multiple 3-element structures)
0	0110	!= 11x1	VST1 (multiple single elements)
0	0110	1101	VST1 (multiple single elements)
0	0110	1111	VST1 (multiple single elements)
0	0111	!= 11x1	VST1 (multiple single elements)
0	0111	1101	VST1 (multiple single elements)
0	0111	1111	VST1 (multiple single elements)
0	100x	!= 11x1	VST2 (multiple 2-element structures)
0	100x	1101	VST2 (multiple 2-element structures)
0	100x	1111	VST2 (multiple 2-element structures)
0	1010	!= 11x1	VST1 (multiple single elements)
0	1010	1101	VST1 (multiple single elements)
0	1010	1111	VST1 (multiple single elements)
1	000x	!= 11x1	VLD4 (multiple 4-element structures)
1	000x	1101	VLD4 (multiple 4-element structures)
1	000x	1111	VLD4 (multiple 4-element structures)
1	0010	!= 11x1	VLD1 (multiple single elements)
1	0010	1101	VLD1 (multiple single elements)
1	0010	1111	VLD1 (multiple single elements)

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields			Instruction page
L	itype	Rm	
1	0011	!= 11x1	VLD2 (multiple 2-element structures)
1	0011	1101	VLD2 (multiple 2-element structures)
1	0011	1111	VLD2 (multiple 2-element structures)
1	010x	!= 11x1	VLD3 (multiple 3-element structures)
1	010x	1101	VLD3 (multiple 3-element structures)
1	010x	1111	VLD3 (multiple 3-element structures)
x	1011	xxxx	Unallocated.
1	0110	!= 11x1	VLD1 (multiple single elements)
1	0110	1101	VLD1 (multiple single elements)
1	0110	1111	VLD1 (multiple single elements)
1	0111	!= 11x1	VLD1 (multiple single elements)
1	0111	1101	VLD1 (multiple single elements)
1	0111	1111	VLD1 (multiple single elements)
x	11xx	xxxx	Unallocated.
1	100x	!= 11x1	VLD2 (multiple 2-element structures)
1	100x	1101	VLD2 (multiple 2-element structures)
1	100x	1111	VLD2 (multiple 2-element structures)
1	1010	!= 11x1	VLD1 (multiple single elements)
1	1010	1101	VLD1 (multiple single elements)
1	1010	1111	VLD1 (multiple single elements)

F3.1.18.2 Advanced SIMD load single structure to all lanes

This section describes the encoding of the Advanced SIMD load single structure to all lanes instruction class. The encodings in this section are decoded from Advanced SIMD element or structure load/store.

15 14 13 12	11 10 9 8	7 6 5 4	3 0	15 12	11 10 9 8	7 6 5 4	3 0
1 1 1 1	1 0 0 1 1	D L 0	Rn	Vd	1 1 N	size T a	Rm

Table F3-83

Decode fields				Instruction page
L	N	a	Rm	
0	x	x	xxxx	Unallocated.
1	00	x	!= 11x1	VLD1 (single element to all lanes)
1	00	x	1101	VLD1 (single element to all lanes)

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields				Instruction page
L	N	a	Rm	
1	00	x	1111	VLD1 (single element to all lanes)
1	01	x	!= 11x1	VLD2 (single 2-element structure to all lanes)
1	01	x	1101	VLD2 (single 2-element structure to all lanes)
1	01	x	1111	VLD2 (single 2-element structure to all lanes)
1	10	0	!= 11x1	VLD3 (single 3-element structure to all lanes)
1	10	0	1101	VLD3 (single 3-element structure to all lanes)
1	10	0	1111	VLD3 (single 3-element structure to all lanes)
1	10	1	xxxx	Unallocated.
1	11	x	!= 11x1	VLD4 (single 4-element structure to all lanes)
1	11	x	1101	VLD4 (single 4-element structure to all lanes)
1	11	x	1111	VLD4 (single 4-element structure to all lanes)

F3.1.18.3 Advanced SIMD load/store single structure to one lane

This section describes the encoding of the Advanced SIMD load/store single structure to one lane instruction class. The encodings in this section are decoded from Advanced SIMD element or structure load/store.

15 14 13 12	11 10 9 8	7 6 5 4	3 0	15 12	11 10 9 8	7 4	3 0
1 1 1 1	1 0 0 1 1	D L 0	Rn	Vd	!=11 size	N	index_align	Rm

Table F3-84

Decode fields				Instruction page
L	size	N	Rm	
0	00	00	!= 11x1	VST1 (single element from one lane)
0	00	00	1101	VST1 (single element from one lane)
0	00	00	1111	VST1 (single element from one lane)
0	00	01	!= 11x1	VST2 (single 2-element structure from one lane)
0	00	01	1101	VST2 (single 2-element structure from one lane)
0	00	01	1111	VST2 (single 2-element structure from one lane)
0	00	10	!= 11x1	VST3 (single 3-element structure from one lane)
0	00	10	1101	VST3 (single 3-element structure from one lane)
0	00	10	1111	VST3 (single 3-element structure from one lane)
0	00	11	!= 11x1	VST4 (single 4-element structure from one lane)
0	00	11	1101	VST4 (single 4-element structure from one lane)
0	00	11	1111	VST4 (single 4-element structure from one lane)

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields				Instruction page
L	size	N	Rm	
0	01	00	!= 11x1	VST1 (single element from one lane)
0	01	00	1101	VST1 (single element from one lane)
0	01	00	1111	VST1 (single element from one lane)
0	01	01	!= 11x1	VST2 (single 2-element structure from one lane)
0	01	01	1101	VST2 (single 2-element structure from one lane)
0	01	01	1111	VST2 (single 2-element structure from one lane)
0	01	10	!= 11x1	VST3 (single 3-element structure from one lane)
0	01	10	1101	VST3 (single 3-element structure from one lane)
0	01	10	1111	VST3 (single 3-element structure from one lane)
0	01	11	!= 11x1	VST4 (single 4-element structure from one lane)
0	01	11	1101	VST4 (single 4-element structure from one lane)
0	01	11	1111	VST4 (single 4-element structure from one lane)
0	10	00	!= 11x1	VST1 (single element from one lane)
0	10	00	1101	VST1 (single element from one lane)
0	10	00	1111	VST1 (single element from one lane)
0	10	01	!= 11x1	VST2 (single 2-element structure from one lane)
0	10	01	1101	VST2 (single 2-element structure from one lane)
0	10	01	1111	VST2 (single 2-element structure from one lane)
0	10	10	!= 11x1	VST3 (single 3-element structure from one lane)
0	10	10	1101	VST3 (single 3-element structure from one lane)
0	10	10	1111	VST3 (single 3-element structure from one lane)
0	10	11	!= 11x1	VST4 (single 4-element structure from one lane)
0	10	11	1101	VST4 (single 4-element structure from one lane)
0	10	11	1111	VST4 (single 4-element structure from one lane)
1	00	00	!= 11x1	VLD1 (single element to one lane)
1	00	00	1101	VLD1 (single element to one lane)
1	00	00	1111	VLD1 (single element to one lane)
1	00	01	!= 11x1	VLD2 (single 2-element structure to one lane)
1	00	01	1101	VLD2 (single 2-element structure to one lane)
1	00	01	1111	VLD2 (single 2-element structure to one lane)
1	00	10	!= 11x1	VLD3 (single 3-element structure to one lane)
1	00	10	1101	VLD3 (single 3-element structure to one lane)
1	00	10	1111	VLD3 (single 3-element structure to one lane)
1	00	11	!= 11x1	VLD4 (single 4-element structure to one lane)
1	00	11	1101	VLD4 (single 4-element structure to one lane)
1	00	11	1111	VLD4 (single 4-element structure to one lane)
1	01	00	!= 11x1	VLD1 (single element to one lane)
1	01	00	1101	VLD1 (single element to one lane)
1	01	00	1111	VLD1 (single element to one lane)

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields				Instruction page
L	size	N	Rm	
1	01	01	!= 11x1	VLD2 (single 2-element structure to one lane)
1	01	01	1101	VLD2 (single 2-element structure to one lane)
1	01	01	1111	VLD2 (single 2-element structure to one lane)
1	01	10	!= 11x1	VLD3 (single 3-element structure to one lane)
1	01	10	1101	VLD3 (single 3-element structure to one lane)
1	01	10	1111	VLD3 (single 3-element structure to one lane)
1	01	11	!= 11x1	VLD4 (single 4-element structure to one lane)
1	01	11	1101	VLD4 (single 4-element structure to one lane)
1	01	11	1111	VLD4 (single 4-element structure to one lane)
1	10	00	!= 11x1	VLD1 (single element to one lane)
1	10	00	1101	VLD1 (single element to one lane)
1	10	00	1111	VLD1 (single element to one lane)
1	10	01	!= 11x1	VLD2 (single 2-element structure to one lane)
1	10	01	1101	VLD2 (single 2-element structure to one lane)
1	10	01	1111	VLD2 (single 2-element structure to one lane)
1	10	10	!= 11x1	VLD3 (single 3-element structure to one lane)
1	10	10	1101	VLD3 (single 3-element structure to one lane)
1	10	10	1111	VLD3 (single 3-element structure to one lane)
1	10	11	!= 11x1	VLD4 (single 4-element structure to one lane)
1	10	11	1101	VLD4 (single 4-element structure to one lane)
1	10	11	1111	VLD4 (single 4-element structure to one lane)

F3.1.19 Load/store single

This section describes the encoding of the Load/store single group. The encodings in this section are decoded from 32-bit.

This decode also imposes the constraint:

- `op0<1>:op1 != 10`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F3-85 Encoding table for the Load/store single group

Decode fields				Decode group or instruction page
op0	op1	op2	op3	
00	x	!= 1111	000000	Load/store, unsigned (register offset)
00	x	!= 1111	000001	Unallocated.
00	x	!= 1111	00001x	Unallocated.
00	x	!= 1111	0001xx	Unallocated.
00	x	!= 1111	001xxx	Unallocated.
00	x	!= 1111	01xxxx	Unallocated.
00	x	!= 1111	10x0xx	Unallocated.
00	x	!= 1111	10x1xx	Load/store, unsigned (immediate, post-indexed)
00	x	!= 1111	1100xx	Load/store, unsigned (negative immediate)
00	x	!= 1111	1110xx	Load/store, unsigned (unprivileged)
00	x	!= 1111	11x1xx	Load/store, unsigned (immediate, pre-indexed)
01	x	!= 1111	xxxxxx	Load/store, unsigned (positive immediate)
0x	x	1111	xxxxxx	Load, unsigned (literal)
10	1	!= 1111	000000	Load/store, signed (register offset)
10	1	!= 1111	000001	Unallocated.
10	1	!= 1111	00001x	Unallocated.
10	1	!= 1111	0001xx	Unallocated.
10	1	!= 1111	001xxx	Unallocated.
10	1	!= 1111	01xxxx	Unallocated.
10	1	!= 1111	10x0xx	Unallocated.
10	1	!= 1111	10x1xx	Load/store, signed (immediate, post-indexed)
10	1	!= 1111	1100xx	Load/store, signed (negative immediate)
10	1	!= 1111	1110xx	Load/store, signed (unprivileged)
10	1	!= 1111	11x1xx	Load/store, signed (immediate, pre-indexed)
11	1	!= 1111	xxxxxx	Load/store, signed (positive immediate)
1x	1	1111	xxxxxx	Load, signed (literal)

F3.1.19.1 Load/store, unsigned (register offset)

This section describes the encoding of the Load/store, unsigned (register offset) instruction class. The encodings in this section are decoded from Load/store single.

15 14 13 12	11 10 9 8	7 6 5 4	3 0	15 12	11 10 9 8	7 6 5 4	3 0
1 1 1 1	1 0 0 0 0	size	L	!=1111	Rt	0 0 0 0 0 0	imm2	Rm
				Rn				

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F3-86

Decode fields			Instruction page
size	**L**	**Rt**	
00	0	xxxx	STRB (register)
00	1	!= 1111	LDRB (register)
00	1	1111	PLD, PLDW (register) - Preload read variant
01	0	xxxx	STRH (register)
01	1	!= 1111	LDRH (register)
01	1	1111	PLD, PLDW (register) - Preload write variant
10	0	xxxx	STR (register)
10	1	xxxx	LDR (register)
11	x	xxxx	Unallocated.

F3.1.19.2 Load/store, unsigned (immediate, post-indexed)

This section describes the encoding of the Load/store, unsigned (immediate, post-indexed) instruction class. The encodings in this section are decoded from Load/store single.

15 14 13 12	11 10 9 8	7 6 5 4	3 0	15 12	11 10	9	8	7 0
1 1 1 1	1 0 0 0	0	size	L	!=1111	Rt	1 0	U	1	imm8
					Rn					

Table F3-87

Decode fields		Instruction page
size	**L**	
00	0	STRB (immediate)
00	1	LDRB (immediate)
01	0	STRH (immediate)
01	1	LDRH (immediate)
10	0	STR (immediate)
10	1	LDR (immediate)
11	x	Unallocated.

F3.1.19.3 Load/store, unsigned (negative immediate)

This section describes the encoding of the Load/store, unsigned (negative immediate) instruction class. The encodings in this section are decoded from Load/store single.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

15 14 13 12	11 10 9 8	7 6 5 4	3 0	15 12	11 10 9 8	7 0
1 1 1 1	1 0 0 0	0 size	L	!=1111 Rn	Rt	1 1 0 0	imm8

Table F3-88

Decode fields			Instruction page
size	**L**	**Rt**	
00	0	xxxx	STRB (immediate)
00	1	!= 1111	LDRB (immediate)
00	1	1111	PLD, PLDW (immediate) - Preload read variant
01	0	xxxx	STRH (immediate)
01	1	!= 1111	LDRH (immediate)
01	1	1111	PLD, PLDW (immediate) - Preload write variant
10	0	xxxx	STR (immediate)
10	1	xxxx	LDR (immediate)
11	x	xxxx	Unallocated.

F3.1.19.4 Load/store, unsigned (unprivileged)

This section describes the encoding of the Load/store, unsigned (unprivileged) instruction class. The encodings in this section are decoded from Load/store single.

15 14 13 12	11 10 9 8	7 6 5 4	3 0	15 12	11 10 9 8	7 0
1 1 1 1	1 0 0 0	0 size	L	!=1111 Rn	Rt	1 1 1 0	imm8

Table F3-89

Decode fields		Instruction page
size	**L**	
00	0	STRBT
00	1	LDRBT
01	0	STRHT
01	1	LDRHT
10	0	STRT

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields		Instruction page
size	L	
10	1	LDRT
11	x	Unallocated.

F3.1.19.5 Load/store, unsigned (immediate, pre-indexed)

This section describes the encoding of the Load/store, unsigned (immediate, pre-indexed) instruction class. The encodings in this section are decoded from Load/store single.

15 14 13 12	11 10 9 8	7 6 5 4	3 0	15 12	11 10	9	8	7 0
1 1 1 1	1 0 0 0	0 size	L	!=1111	Rt	1 1	U	1	imm8
			Rn					

Table F3-90

Decode fields		Instruction page
size	L	
00	0	STRB (immediate)
00	1	LDRB (immediate)
01	0	STRH (immediate)
01	1	LDRH (immediate)
10	0	STR (immediate)
10	1	LDR (immediate)
11	x	Unallocated.

F3.1.19.6 Load/store, unsigned (positive immediate)

This section describes the encoding of the Load/store, unsigned (positive immediate) instruction class. The encodings in this section are decoded from Load/store single.

15 14 13 12	11 10 9 8	7 6 5 4	3 0	15 12	11 0
1 1 1 1	1 0 0 0	1 size	L	!=1111	Rt	imm12
			Rn		

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F3-91

Decode fields			Instruction page
size	**L**	**Rt**	
00	0	xxxx	STRB (immediate)
00	1	!= 1111	LDRB (immediate)
00	1	1111	PLD, PLDW (immediate) - Preload read variant
01	0	xxxx	STRH (immediate)
01	1	!= 1111	LDRH (immediate)
01	1	1111	PLD, PLDW (immediate) - Preload write variant
10	0	xxxx	STR (immediate)
10	1	xxxx	LDR (immediate)

F3.1.19.7 Load, unsigned (literal)

This section describes the encoding of the Load, unsigned (literal) instruction class. The encodings in this section are decoded from Load/store single.

15 14 13 12	11 10 9 8	7	6 5 4	3 2 1 0	15 12	11 0
1 1 1 1	1 0 0 0	U	size L	1 1 1 1	Rt	imm12

Table F3-92

Decode fields			Instruction page
size	**L**	**Rt**	
0x	1	1111	PLD (literal)
00	1	!= 1111	LDRB (literal)
01	1	!= 1111	LDRH (literal)
10	1	xxxx	LDR (literal)
11	x	xxxx	Unallocated.

F3.1.19.8 Load/store, signed (register offset)

This section describes the encoding of the Load/store, signed (register offset) instruction class. The encodings in this section are decoded from Load/store single.

15 14 13 12	11 10 9 8	7 6 5 4	3 0	15 12	11 10 9 8	7 6 5 4	3 0
1 1 1 1	1 0 0 1 0	size	1	!=1111	0 0 0 0	0 0 imm2	Rm
				Rn			

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F3-93

Decode fields		Instruction page
size	**Rt**	
`00`	`!= 1111`	LDRSB (register)
`00`	`1111`	PLI (register)
`01`	`!= 1111`	LDRSH (register)
`01`	`1111`	Reserved hint, behaves as NOP.
`1x`	`xxxx`	Unallocated.

F3.1.19.9 Load/store, signed (immediate, post-indexed)

This section describes the encoding of the Load/store, signed (immediate, post-indexed) instruction class. The encodings in this section are decoded from Load/store single.

15 14 13 12	11 10 9 8	7 6 5 4	3 0	15 12	11 10	9	8	7 0
1 1 1 1	1 0 0 1 0	size	1	!=1111	Rt	1 0	U	1	imm8
				Rn				

Table F3-94

Decode fields	Instruction page
size	
`00`	LDRSB (immediate)
`01`	LDRSH (immediate)
`1x`	Unallocated.

F3.1.19.10 Load/store, signed (negative immediate)

This section describes the encoding of the Load/store, signed (negative immediate) instruction class. The encodings in this section are decoded from Load/store single.

15 14 13 12	11 10 9 8	7 6 5 4	3 0	15 12	11 10 9 8	7 0
1 1 1 1	1 0 0 1 0	size	1	!=1111	Rt	1 1 0 0	imm8
				Rn		

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F3-95

Decode fields		Instruction page
size	**Rt**	
`00`	`!= 1111`	LDRSB (immediate)
`00`	`1111`	PLI (immediate, literal)
`01`	`!= 1111`	LDRSH (immediate)
`01`	`1111`	Reserved hint, behaves as NOP.
`1x`	`xxxx`	Unallocated.

F3.1.19.11 Load/store, signed (unprivileged)

This section describes the encoding of the Load/store, signed (unprivileged) instruction class. The encodings in this section are decoded from Load/store single.

15 14 13 12	11 10 9 8	7 6 5 4	3 0	15 12	11 10 9 8	7 0
1 1 1 1	1 0 0 1 0	size	1	!=1111	Rt	1 1 1 0	imm8
				Rn		

Table F3-96

Decode fields	Instruction page
size	
`00`	LDRSBT
`01`	LDRSHT
`1x`	Unallocated.

F3.1.19.12 Load/store, signed (immediate, pre-indexed)

This section describes the encoding of the Load/store, signed (immediate, pre-indexed) instruction class. The encodings in this section are decoded from Load/store single.

15 14 13 12	11 10 9 8	7 6 5 4	3 0	15 12	11 10 9 8	7 0
1 1 1 1	1 0 0 1 0	size	1	!=1111	Rt	1 1 U 1	imm8
				Rn		

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F3-97

Decode fields size	Instruction page
`00`	LDRSB (immediate)
`01`	LDRSH (immediate)
`1x`	Unallocated.

F3.1.19.13 Load/store, signed (positive immediate)

This section describes the encoding of the Load/store, signed (positive immediate) instruction class. The encodings in this section are decoded from Load/store single.

15 14 13 12	11 10 9 8	7 6 5 4	3 0	15 12	11 0
1 1 1 1	1 0 0 1 1	size	1	!=1111	Rt	imm12
				Rn	

Table F3-98

Decode fields size	Rt	Instruction page
`00`	`!= 1111`	LDRSB (immediate)
`00`	`1111`	PLI (immediate, literal)
`01`	`!= 1111`	LDRSH (immediate)
`01`	`1111`	Reserved hint, behaves as NOP.

F3.1.19.14 Load, signed (literal)

This section describes the encoding of the Load, signed (literal) instruction class. The encodings in this section are decoded from Load/store single.

15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0	15 12	11 0
1 1 1 1	1 0 0 1	U	size	1 1 1 1 1	Rt	imm12

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F3-99

Decode fields		Instruction page
size	**Rt**	
00	!= 1111	LDRSB (literal)
00	1111	PLI (immediate, literal)
01	!= 1111	LDRSH (literal)
01	1111	Reserved hint, behaves as NOP.
1x	xxxx	Unallocated.

F3.1.20 Data-processing (register)

This section describes the encoding of the Data-processing (register) group. The encodings in this section are decoded from 32-bit.

Table F3-100 Encoding table for the Data-processing (register) group

Decode fields			Decode group or instruction page
op0	**op1**	**op2**	
0	1111	0000	MOV, MOVS (register-shifted register) - Flag setting variant
0	1111	0001	Unallocated.
0	1111	001x	Unallocated.
0	1111	01xx	Unallocated.
0	1111	1xxx	Register extends
1	1111	0xxx	Parallel add-subtract
1	1111	10xx	Data-processing (two source registers)
1	1111	11xx	Unallocated.
x	!= 1111	xxxx	Unallocated.

F3.1.20.1 Register extends

This section describes the encoding of the Register extends instruction class. The encodings in this section are decoded from Data-processing (register).

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

15 14 13 12	11 10 9 8	7 6 5 4	3 0	15 14 13 12	11 8	7	6	5 4	3 0
1 1 1 1	1 0 1 0	0 op1	U Rn	1 1 1 1	Rd	1	(0)	rotate	Rm

Table F3-101

Decode fields			Instruction page
op1	U	Rn	
00	0	!= 1111	SXTAH
00	0	1111	SXTH
00	1	!= 1111	UXTAH
00	1	1111	UXTH
01	0	!= 1111	SXTAB16
01	0	1111	SXTB16
01	1	!= 1111	UXTAB16
01	1	1111	UXTB16
10	0	!= 1111	SXTAB
10	0	1111	SXTB
10	1	!= 1111	UXTAB
10	1	1111	UXTB
11	x	xxxx	Unallocated.

F3.1.20.2 Parallel add-subtract

This section describes the encoding of the Parallel add-subtract instruction class. The encodings in this section are decoded from Data-processing (register).

15 14 13 12	11 10 9 8	7 6 4	3 0	15 14 13 12	11 8	7	6	5	4	3 0
1 1 1 1	1 0 1 0	1 op1	Rn	1 1 1 1	Rd	0	U	H	S	Rm

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F3-102

Decode fields				Instruction page
op1	U	H	S	
000	0	0	0	SADD8
000	0	0	1	QADD8
000	0	1	0	SHADD8
000	0	1	1	Unallocated.
000	1	0	0	UADD8
000	1	0	1	UQADD8
000	1	1	0	UHADD8
000	1	1	1	Unallocated.
001	0	0	0	SADD16
001	0	0	1	QADD16
001	0	1	0	SHADD16
001	0	1	1	Unallocated.
001	1	0	0	UADD16
001	1	0	1	UQADD16
001	1	1	0	UHADD16
001	1	1	1	Unallocated.
010	0	0	0	SASX
010	0	0	1	QASX
010	0	1	0	SHASX
010	0	1	1	Unallocated.
010	1	0	0	UASX
010	1	0	1	UQASX
010	1	1	0	UHASX
010	1	1	1	Unallocated.
100	0	0	0	SSUB8
100	0	0	1	QSUB8
100	0	1	0	SHSUB8
100	0	1	1	Unallocated.
100	1	0	0	USUB8
100	1	0	1	UQSUB8
100	1	1	0	UHSUB8
100	1	1	1	Unallocated.
101	0	0	0	SSUB16
101	0	0	1	QSUB16
101	0	1	0	SHSUB16
101	0	1	1	Unallocated.
101	1	0	0	USUB16

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields				Instruction page
op1	U	H	S	
101	1	0	1	UQSUB16
101	1	1	0	UHSUB16
101	1	1	1	Unallocated.
110	0	0	0	SSAX
110	0	0	1	QSAX
110	0	1	0	SHSAX
110	0	1	1	Unallocated.
110	1	0	0	USAX
110	1	0	1	UQSAX
110	1	1	0	UHSAX
110	1	1	1	Unallocated.
111	x	x	x	Unallocated.

F3.1.20.3 Data-processing (two source registers)

This section describes the encoding of the Data-processing (two source registers) instruction class. The encodings in this section are decoded from Data-processing (register).

15 14 13 12	11 10 9 8	7 6 4	3 0	15 14 13 12	11 8	7 6	5 4	3 0
1 1 1 1	1 0 1 0 1	op1	Rn	1 1 1 1	Rd	1 0	op2	Rm

Table F3-103

Decode fields		Instruction page	Feature
op1	op2		
000	00	QADD	-
000	01	QDADD	-
000	10	QSUB	-
000	11	QDSUB	-
001	00	REV	-
001	01	REV16	-
001	10	RBIT	-
001	11	REVSH	-
010	00	SEL	-
010	01	Unallocated.	-
010	1x	Unallocated.	-

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields		Instruction page	Feature
op1	op2		
011	00	CLZ	-
011	01	Unallocated.	-
011	1x	Unallocated.	-
100	00	CRC32 - CRC32B variant	FEAT_CRC32
100	01	CRC32 - CRC32H variant	FEAT_CRC32
100	10	CRC32 - CRC32W variant	FEAT_CRC32
100	11	CONSTRAINED UNPREDICTABLE	-
101	00	CRC32C - CRC32CB variant	FEAT_CRC32
101	01	CRC32C - CRC32CH variant	FEAT_CRC32
101	10	CRC32C - CRC32CW variant	FEAT_CRC32
101	11	CONSTRAINED UNPREDICTABLE	-
11x	xx	Unallocated.	-

The behavior of the CONSTRAINED UNPREDICTABLE encodings in this table is described in CONSTRAINED UNPREDICTABLE behavior for T32 and A32 instruction encodings.

F3.1.21 Multiply, multiply accumulate, and absolute difference

This section describes the encoding of the Multiply, multiply accumulate, and absolute difference group. The encodings in this section are decoded from 32-bit.

Table F3-104 Encoding table for the Multiply, multiply accumulate, and absolute difference group

Decode fields	Decode group or instruction page
op0	
00	Multiply and absolute difference
01	Unallocated.
1x	Unallocated.

F3.1.21.1 Multiply and absolute difference

This section describes the encoding of the Multiply and absolute difference instruction class. The encodings in this section are decoded from Multiply, multiply accumulate, and absolute difference.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

15 14 13 12	11 10 9 8	7 6 4	3 0	15 12	11 8	7 6	5 4	3 0
1 1 1 1	1 0 1 1	0	op1	Rn	Ra	Rd	0 0	op2	Rm

Table F3-105

Decode fields			Instruction page
op1	Ra	op2	
000	!= 1111	00	MLA, MLAS
000	xxxx	01	MLS
000	xxxx	1x	Unallocated.
000	1111	00	MUL, MULS
001	!= 1111	00	SMLABB, SMLABT, SMLATB, SMLATT - SMLABB variant
001	!= 1111	01	SMLABB, SMLABT, SMLATB, SMLATT - SMLABT variant
001	!= 1111	10	SMLABB, SMLABT, SMLATB, SMLATT - SMLATB variant
001	!= 1111	11	SMLABB, SMLABT, SMLATB, SMLATT - SMLATT variant
001	1111	00	SMULBB, SMULBT, SMULTB, SMULTT - SMULBB variant
001	1111	01	SMULBB, SMULBT, SMULTB, SMULTT - SMULBT variant
001	1111	10	SMULBB, SMULBT, SMULTB, SMULTT - SMULTB variant
001	1111	11	SMULBB, SMULBT, SMULTB, SMULTT - SMULTT variant
010	!= 1111	00	SMLAD, SMLADX - SMLAD variant
010	!= 1111	01	SMLAD, SMLADX - SMLADX variant
010	xxxx	1x	Unallocated.
010	1111	00	SMUAD, SMUADX - SMUAD variant
010	1111	01	SMUAD, SMUADX - SMUADX variant
011	!= 1111	00	SMLAWB, SMLAWT - SMLAWB variant
011	!= 1111	01	SMLAWB, SMLAWT - SMLAWT variant
011	xxxx	1x	Unallocated.
011	1111	00	SMULWB, SMULWT - SMULWB variant
011	1111	01	SMULWB, SMULWT - SMULWT variant
100	!= 1111	00	SMLSD, SMLSDX - SMLSD variant
100	!= 1111	01	SMLSD, SMLSDX - SMLSDX variant
100	xxxx	1x	Unallocated.
100	1111	00	SMUSD, SMUSDX - SMUSD variant
100	1111	01	SMUSD, SMUSDX - SMUSDX variant
101	!= 1111	00	SMMLA, SMMLAR - SMMLA variant
101	!= 1111	01	SMMLA, SMMLAR - SMMLAR variant
101	xxxx	1x	Unallocated.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields			Instruction page
op1	Ra	op2	
101	1111	00	SMMUL, SMMULR - SMMUL variant
101	1111	01	SMMUL, SMMULR - SMMULR variant
110	xxxx	00	SMMLS, SMMLSR - SMMLS variant
110	xxxx	01	SMMLS, SMMLSR - SMMLSR variant
110	xxxx	1x	Unallocated.
111	!= 1111	00	USADA8
111	xxxx	01	Unallocated.
111	xxxx	1x	Unallocated.
111	1111	00	USAD8

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F3.2 About the T32 Advanced SIMD and floating-point instructions and their encoding

The Advanced SIMD and floating-point instructions are common to the T32 and A32 instruction sets. These instructions perform Advanced SIMD and floating-point operations on a common register file, the SIMD&FP register file. This means:

- In general, the instructions that load or store registers in this file, or move data between general-purpose registers and this register file, are common to the Advanced SIMD and floating-point instructions.
- There are distinct Advanced SIMD data-processing instructions and floating-point data-processing instructions.

All T32 Advanced SIMD and floating-point instructions have 32-bit encodings. Different groups of these instructions are decoded from different points in the 32-bit T32 instruction decode structure. Table F3-106 shows these instruction groups, and where each group is decoded from the overall T32 decode structure:

Table F3-106 Advanced SIMD and floating-point instructions in the T32 decode structure

Advanced SIMD and floating-point instruction group	T32 decode is from
Advanced SIMD and System register load/store and 64-bit move	System register access, Advanced SIMD, and floating-point
Floating-point data-processing	System register access, Advanced SIMD, and floating-point
Advanced SIMD and System register 32-bit move	System register access, Advanced SIMD, and floating-point
Advanced SIMD data-processing	System register access, Advanced SIMD, and floating-point
Advanced SIMD element or structure load/store	32-bit

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Chapter F4
A32 Instruction Set Encoding

This chapter describes the encoding of the A32 instruction set. It contains the following sections:

- A32 instruction set encoding.
- About the A32 Advanced SIMD and floating-point instructions and their encoding.

In this chapter:

- In the decode tables, an entry of `x` for a field value means the value of the field does not affect the decoding.
- In the decode diagrams, a shaded field indicates that the bits in that field are not used in that level of decode.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F4.1 A32 instruction set encoding

The A32 instruction stream is a sequence of 32-bit word-aligned words. Each instruction is a single word in the stream.

Most A32 instructions can be conditional, with a condition determined by bits[31:28] of the instruction, the `cond` field. For more information see The Condition code field in A32 instruction encodings. This applies to all instructions except those with the `cond` field equal to `0b111`.

The behavior of an attempt to execute an unallocated instruction is described in UNDEFINED, UNPREDICTABLE, and CONSTRAINED UNPREDICTABLE instruction set space.

For more information on A32 instruction encodings see About the T32 and A32 Instruction Descriptions.

The A32 instruction encoding is:

Table F4-1 Main encoding table for the A32 instruction set

Decode fields			Decode group or instruction page
cond	**op0**	**op1**	
`!= 1111`	`00x`	`x`	Data-processing and miscellaneous instructions
`!= 1111`	`010`	`x`	Load/Store Word, Unsigned Byte (immediate, literal)
`!= 1111`	`011`	`0`	Load/Store Word, Unsigned Byte (register)
`!= 1111`	`011`	`1`	Media instructions
`xxxx`	`10x`	`x`	Branch, branch with link, and block data transfer
`xxxx`	`11x`	`x`	System register access, Advanced SIMD, floating-point, and Supervisor call
`1111`	`0xx`	`x`	Unconditional instructions

F4.1.1 Data-processing and miscellaneous instructions

This section describes the encoding of the Data-processing and miscellaneous instructions group. The encodings in this section are decoded from A32 instruction set encoding.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F4-2 Encoding table for the Data-processing and miscellaneous instructions group

Decode fields					Decode group or instruction page
op0	op1	op2	op3	op4	
0	xxxxx	1	!= 00	1	Extra load/store
0	0xxxx	1	00	1	Multiply and Accumulate
0	1xxxx	1	00	1	Synchronization primitives and Load-Acquire/Store-Release
0	10xx0	0	xx	x	Miscellaneous
0	10xx0	1	xx	0	Halfword Multiply and Accumulate
0	!= 10xx0	x	xx	0	Data-processing register (immediate shift)
0	!= 10xx0	0	xx	1	Data-processing register (register shift)
1	xxxxx	x	xx	x	Data-processing immediate

F4.1.1.1 Multiply and Accumulate

This section describes the encoding of the Multiply and Accumulate instruction class. The encodings in this section are decoded from Data-processing and miscellaneous instructions.

31–28	27 26 25 24	23–21	20	19–16	15–12	11–8	7 6 5 4	3–0
!=1111	0 0 0 0	opc	S	RdHi	RdLo	Rm	1 0 0 1	Rn
cond								

Table F4-3

Decode fields		Instruction page
opc	S	
000	x	MUL, MULS
001	x	MLA, MLAS
010	0	UMAAL
010	1	Unallocated.
011	0	MLS
011	1	Unallocated.
100	x	UMULL, UMULLS
101	x	UMLAL, UMLALS
110	x	SMULL, SMULLS
111	x	SMLAL, SMLALS

F4.1.1.2 Halfword Multiply and Accumulate

This section describes the encoding of the Halfword Multiply and Accumulate instruction class. The encodings in this section are decoded from Data-processing and miscellaneous instructions.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

31–28	27 26 25 24	23 22 21	20	19–16	15–12	11–8	7	6	5	4	3–0
!=1111 (cond)	0 0 0 1 0	opc	0	Rd	Ra	Rm	1	M	N	0	Rn

Table F4-4

Decode fields			Instruction page
opc	**M**	**N**	
00	x	x	SMLABB, SMLABT, SMLATB, SMLATT
01	0	0	SMLAWB, SMLAWT - SMLAWB variant
01	0	1	SMULWB, SMULWT - SMULWB variant
01	1	0	SMLAWB, SMLAWT - SMLAWT variant
01	1	1	SMULWB, SMULWT - SMULWT variant
10	x	x	SMLALBB, SMLALBT, SMLALTB, SMLALTT
11	x	x	SMULBB, SMULBT, SMULTB, SMULTT

F4.1.2 Extra load/store

This section describes the encoding of the Extra load/store group. The encodings in this section are decoded from Data-processing and miscellaneous instructions.

31–28	27–25	24	23	22 21	8	7	6 5	4	3–0
!=1111	000		op0			1	!=00	1	

Table F4-5 Encoding table for the Extra load/store group

Decode fields	Decode group or instruction page
op0	
0	Load/Store Dual, Half, Signed Byte (register)
1	Load/Store Dual, Half, Signed Byte (immediate, literal)

F4.1.2.1 Load/Store Dual, Half, Signed Byte (register)

This section describes the encoding of the Load/Store Dual, Half, Signed Byte (register) instruction class. The encodings in this section are decoded from Extra load/store.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

31–28	27	26	25	24	23	22	21	20	19–16	15–12	11	10	9	8	7	6	5	4	3–0
!=1111	0	0	0	P	U	0	W	o1	Rn	Rt	(0)	(0)	(0)	(0)	1	!=00		1	Rm
cond																op2			

Table F4-6

Decode fields				Instruction page
P	**W**	**o1**	**op2**	
0	0	0	01	STRH (register) - Post-indexed variant
0	0	0	10	LDRD (register) - Post-indexed variant
0	0	0	11	STRD (register) - Post-indexed variant
0	0	1	01	LDRH (register) - Post-indexed variant
0	0	1	10	LDRSB (register) - Post-indexed variant
0	0	1	11	LDRSH (register) - Post-indexed variant
0	1	0	01	STRHT
0	1	0	10	Unallocated.
0	1	0	11	Unallocated.
0	1	1	01	LDRHT
0	1	1	10	LDRSBT
0	1	1	11	LDRSHT
1	x	0	01	STRH (register) - Pre-indexed variant
1	x	0	10	LDRD (register) - Pre-indexed variant
1	x	0	11	STRD (register) - Pre-indexed variant
1	x	1	01	LDRH (register) - Pre-indexed variant
1	x	1	10	LDRSB (register) - Pre-indexed variant
1	x	1	11	LDRSH (register) - Pre-indexed variant

F4.1.2.2 Load/Store Dual, Half, Signed Byte (immediate, literal)

This section describes the encoding of the Load/Store Dual, Half, Signed Byte (immediate, literal) instruction class. The encodings in this section are decoded from Extra load/store.

31–28	27	26	25	24	23	22	21	20	19–16	15–12	11–8	7	6	5	4	3–0
!=1111	0	0	0	P	U	1	W	o1	Rn	Rt	imm4H	1	!=00		1	imm4L
cond													op2			

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F4-7

Decode fields				Instruction page
P:W	**o1**	**Rn**	**op2**	
xx	0	1111	10	LDRD (literal)
!= 01	1	1111	01	LDRH (literal)
!= 01	1	1111	10	LDRSB (literal)
!= 01	1	1111	11	LDRSH (literal)
00	0	!= 1111	10	LDRD (immediate) - Post-indexed variant
00	0	xxxx	01	STRH (immediate) - Post-indexed variant
00	0	xxxx	11	STRD (immediate) - Post-indexed variant
00	1	!= 1111	01	LDRH (immediate) - Post-indexed variant
00	1	!= 1111	10	LDRSB (immediate) - Post-indexed variant
00	1	!= 1111	11	LDRSH (immediate) - Post-indexed variant
01	0	!= 1111	10	Unallocated.
01	0	xxxx	01	STRHT
01	0	xxxx	11	Unallocated.
01	1	xxxx	01	LDRHT
01	1	xxxx	10	LDRSBT
01	1	xxxx	11	LDRSHT
10	0	!= 1111	10	LDRD (immediate) - Offset variant
10	0	xxxx	01	STRH (immediate) - Offset variant
10	0	xxxx	11	STRD (immediate) - Offset variant
10	1	!= 1111	01	LDRH (immediate) - Offset variant
10	1	!= 1111	10	LDRSB (immediate) - Offset variant
10	1	!= 1111	11	LDRSH (immediate) - Offset variant
11	0	!= 1111	10	LDRD (immediate) - Pre-indexed variant
11	0	xxxx	01	STRH (immediate) - Pre-indexed variant
11	0	xxxx	11	STRD (immediate) - Pre-indexed variant
11	1	!= 1111	01	LDRH (immediate) - Pre-indexed variant
11	1	!= 1111	10	LDRSB (immediate) - Pre-indexed variant
11	1	!= 1111	11	LDRSH (immediate) - Pre-indexed variant

F4.1.3 Synchronization primitives and Load-Acquire/Store-Release

This section describes the encoding of the Synchronization primitives and Load-Acquire/Store-Release group. The encodings in this section are decoded from Data-processing and miscellaneous instructions.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

31–28	27–24	23	22–12	11	10	9	8	7–4	3–0
!=1111	0001	op0		11				1001	

Table F4-8 Encoding table for the Synchronization primitives and Load-Acquire/Store-Release group

Decode fields op0	Decode group or instruction page
0	Unallocated.
1	Load/Store Exclusive and Load-Acquire/Store-Release

F4.1.3.1 Load/Store Exclusive and Load-Acquire/Store-Release

This section describes the encoding of the Load/Store Exclusive and Load-Acquire/Store-Release instruction class. The encodings in this section are decoded from Synchronization primitives and Load-Acquire/Store-Release.

31–28	27 26 25 24	23 22 21 20	19–16	15–12	11	10	9	8	7 6 5 4	3–0
!=1111 cond	0 0 0 1 1	size L	Rn	xRd	(1)	(1)	ex	ord	1 0 0 1	xRt

Table F4-9

Decode fields size	L	ex	ord	Instruction page
00	0	0	0	STL
00	0	0	1	Unallocated.
00	0	1	0	STLEX
00	0	1	1	STREX
00	1	0	0	LDA
00	1	0	1	Unallocated.
00	1	1	0	LDAEX
00	1	1	1	LDREX
01	0	0	x	Unallocated.
01	0	1	0	STLEXD
01	0	1	1	STREXD

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields				Instruction page
size	L	ex	ord	
01	1	0	x	Unallocated.
01	1	1	0	LDAEXD
01	1	1	1	LDREXD
10	0	0	0	STLB
10	0	0	1	Unallocated.
10	0	1	0	STLEXB
10	0	1	1	STREXB
10	1	0	0	LDAB
10	1	0	1	Unallocated.
10	1	1	0	LDAEXB
10	1	1	1	LDREXB
11	0	0	0	STLH
11	0	0	1	Unallocated.
11	0	1	0	STLEXH
11	0	1	1	STREXH
11	1	0	0	LDAH
11	1	0	1	Unallocated.
11	1	1	0	LDAEXH
11	1	1	1	LDREXH

F4.1.4 Miscellaneous

This section describes the encoding of the Miscellaneous group. The encodings in this section are decoded from Data-processing and miscellaneous instructions.

31–28	27–23	22 21 20	19–8	7	6–4	3–0
!=1111	00010	op0	0	0	op1	

Table F4-10 Encoding table for the Miscellaneous group

Decode fields		Decode group or instruction page
op0	op1	
00	001	Unallocated.
00	010	Unallocated.
00	011	Unallocated.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields		Decode group or instruction page
op0	op1	
00	110	Unallocated.
01	001	BX
01	010	BXJ
01	011	BLX (register)
01	110	Unallocated.
10	001	Unallocated.
10	010	Unallocated.
10	011	Unallocated.
10	110	Unallocated.
11	001	CLZ
11	010	Unallocated.
11	011	Unallocated.
11	110	ERET
xx	111	Exception Generation
xx	000	Move special register (register)
xx	100	Cyclic Redundancy Check
xx	101	Integer Saturating Arithmetic

F4.1.4.1 Exception Generation

This section describes the encoding of the Exception Generation instruction class. The encodings in this section are decoded from Miscellaneous.

31 28	27 26 25 24	23 22 21 20	19 8	7 6 5 4	3 0
!=1111	0 0 0 1 0	opc	0	imm12	0 1 1 1	imm4
cond					

Table F4-11

Decode fields	Instruction page
opc	
00	HLT
01	BKPT
10	HVC
11	SMC

F4.1.4.2 Move special register (register)

This section describes the encoding of the Move special register (register) instruction class. The encodings in this section are decoded from Miscellaneous.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

31–28	27	26	25	24	23	22	21	20	19–16	15–12	11	10	9	8	7	6	5	4	3–0
!=1111	0	0	0	1	0	opc		0	mask	Rd	(0)	(0)	B	m	0	0	0	0	Rn
cond																			

Table F4-12

Decode fields		Instruction page
opc	**B**	
`x0`	`0`	MRS
`x0`	`1`	MRS (Banked register)
`x1`	`0`	MSR (register)
`x1`	`1`	MSR (Banked register)

F4.1.4.3 Cyclic Redundancy Check

This section describes the encoding of the Cyclic Redundancy Check instruction class. The encodings in this section are decoded from Miscellaneous.

31–28	27	26	25	24	23	22	21	20	19–16	15–12	11	10	9	8	7	6	5	4	3–0
!=1111	0	0	0	1	0	sz		0	Rn	Rd	(0)	(0)	C	(0)	0	1	0	0	Rm
cond																			

Table F4-13

Decode fields		Instruction page	Feature
sz	**C**		
`00`	`0`	CRC32 - CRC32B variant	FEAT_CRC32
`00`	`1`	CRC32C - CRC32CB variant	FEAT_CRC32
`01`	`0`	CRC32 - CRC32H variant	FEAT_CRC32
`01`	`1`	CRC32C - CRC32CH variant	FEAT_CRC32
`10`	`0`	CRC32 - CRC32W variant	FEAT_CRC32
`10`	`1`	CRC32C - CRC32CW variant	FEAT_CRC32
`11`	`x`	CONSTRAINED UNPREDICTABLE	-

The behavior of the CONSTRAINED UNPREDICTABLE encodings in this table is described in CONSTRAINED UNPREDICTABLE behavior for T32 and A32 instruction encodings.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F4.1.4.4 Integer Saturating Arithmetic

This section describes the encoding of the Integer Saturating Arithmetic instruction class. The encodings in this section are decoded from Miscellaneous.

Table F4-14

Decode fields opc	Instruction page
00	QADD
01	QSUB
10	QDADD
11	QDSUB

F4.1.5 Data-processing register (immediate shift)

This section describes the encoding of the Data-processing register (immediate shift) group. The encodings in this section are decoded from Data-processing and miscellaneous instructions.

This decode also imposes the constraint:

- `op0:op1 != 100`.

Table F4-15 Encoding table for the Data-processing register (immediate shift) group

Decode fields op0	op1	Decode group or instruction page
0x	x	Integer Data Processing (three register, immediate shift)
10	1	Integer Test and Compare (two register, immediate shift)

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields		Decode group or instruction page
op0	op1	
11	x	Logical Arithmetic (three register, immediate shift)

F4.1.5.1 Integer Data Processing (three register, immediate shift)

This section describes the encoding of the Integer Data Processing (three register, immediate shift) instruction class. The encodings in this section are decoded from Data-processing register (immediate shift).

31–28	27 26 25 24	23–21	20	19–16	15–12	11–7	6 5	4	3–0
!=1111 (cond)	0 0 0 0	opc	S	Rn	Rd	imm5	stype	0	Rm

Table F4-16

Decode fields				Instruction page
opc	S	Rn	imm5:stype	
000	x	xxxx	!= 0000011	AND, ANDS (register) - ANDS, shift or rotate by value variant
000	x	xxxx	0000011	AND, ANDS (register) - ANDS, rotate right with extend variant
001	x	xxxx	!= 0000011	EOR, EORS (register) - EORS, shift or rotate by value variant
001	x	xxxx	0000011	EOR, EORS (register) - EORS, rotate right with extend variant
010	0	!= 1101	!= 0000011	SUB, SUBS (register) - SUB, shift or rotate by value variant
010	0	!= 1101	0000011	SUB, SUBS (register) - SUB, rotate right with extend variant
010	0	1101	!= 0000011	SUB, SUBS (SP minus register) - SUB, shift or rotate by value variant
010	0	1101	0000011	SUB, SUBS (SP minus register) - SUB, rotate right with extend variant
010	1	!= 1101	!= 0000011	SUB, SUBS (register) - SUBS, shift or rotate by value variant
010	1	!= 1101	0000011	SUB, SUBS (register) - SUBS, rotate right with extend variant
010	1	1101	!= 0000011	SUB, SUBS (SP minus register) - SUBS, shift or rotate by value variant
010	1	1101	0000011	SUB, SUBS (SP minus register) - SUBS, rotate right with extend variant
011	x	xxxx	!= 0000011	RSB, RSBS (register) - RSBS, shift or rotate by value variant
011	x	xxxx	0000011	RSB, RSBS (register) - RSBS, rotate right with extend variant
100	0	!= 1101	!= 0000011	ADD, ADDS (register) - ADD, shift or rotate by value variant
100	0	!= 1101	0000011	ADD, ADDS (register) - ADD, rotate right with extend variant
100	0	1101	!= 0000011	ADD, ADDS (SP plus register) - ADD, shift or rotate by value variant
100	0	1101	0000011	ADD, ADDS (SP plus register) - ADD, rotate right with extend variant
100	1	!= 1101	!= 0000011	ADD, ADDS (register) - ADDS, shift or rotate by value variant
100	1	!= 1101	0000011	ADD, ADDS (register) - ADDS, rotate right with extend variant
100	1	1101	!= 0000011	ADD, ADDS (SP plus register) - ADDS, shift or rotate by value variant
100	1	1101	0000011	ADD, ADDS (SP plus register) - ADDS, rotate right with extend variant

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields				Instruction page
opc	S	Rn	imm5:stype	
101	x	xxxx	!= 0000011	ADC, ADCS (register) - ADCS, shift or rotate by value variant
101	x	xxxx	0000011	ADC, ADCS (register) - ADCS, rotate right with extend variant
110	x	xxxx	!= 0000011	SBC, SBCS (register) - SBCS, shift or rotate by value variant
110	x	xxxx	0000011	SBC, SBCS (register) - SBCS, rotate right with extend variant
111	x	xxxx	!= 0000011	RSC, RSCS (register) - RSCS, shift or rotate by value variant
111	x	xxxx	0000011	RSC, RSCS (register) - RSCS, rotate right with extend variant

F4.1.5.2 Integer Test and Compare (two register, immediate shift)

This section describes the encoding of the Integer Test and Compare (two register, immediate shift) instruction class. The encodings in this section are decoded from Data-processing register (immediate shift).

31–28	27–24	23–20	19–16	15	14	13	12	11–7	6–5	4	3–0
!=1111 (cond)	0 0 0 1 0	opc	Rn	(0)	(0)	(0)	(0)	imm5	stype	0	Rm

Table F4-17

Decode fields		Instruction page
opc	imm5:stype	
00	!= 0000011	TST (register) - Shift or rotate by value variant
00	0000011	TST (register) - Rotate right with extend variant
01	!= 0000011	TEQ (register) - Shift or rotate by value variant
01	0000011	TEQ (register) - Rotate right with extend variant
10	!= 0000011	CMP (register) - Shift or rotate by value variant
10	0000011	CMP (register) - Rotate right with extend variant
11	!= 0000011	CMN (register) - Shift or rotate by value variant
11	0000011	CMN (register) - Rotate right with extend variant

F4.1.5.3 Logical Arithmetic (three register, immediate shift)

This section describes the encoding of the Logical Arithmetic (three register, immediate shift) instruction class. The encodings in this section are decoded from Data-processing register (immediate shift).

31–28	27–24	23–20	19–16	15–12	11–7	6–5	4	3–0
!=1111 (cond)	0 0 0 1 1	opc	S	Rn	Rd	imm5	stype	0	Rm

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F4-18

Decode fields		Instruction page
opc	**imm5:stype**	
00	!= 0000011	ORR, ORRS (register) - ORRS, shift or rotate by value variant
00	0000011	ORR, ORRS (register) - ORRS, rotate right with extend variant
01	!= 0000011	MOV, MOVS (register) - MOVS, shift or rotate by value variant
01	0000011	MOV, MOVS (register) - MOVS, rotate right with extend variant
10	!= 0000011	BIC, BICS (register) - BICS, shift or rotate by value variant
10	0000011	BIC, BICS (register) - BICS, rotate right with extend variant
11	!= 0000011	MVN, MVNS (register) - MVNS, shift or rotate by value variant
11	0000011	MVN, MVNS (register) - MVNS, rotate right with extend variant

F4.1.6 Data-processing register (register shift)

This section describes the encoding of the Data-processing register (register shift) group. The encodings in this section are decoded from Data-processing and miscellaneous instructions.

This decode also imposes the constraint:

- `op0:op1 != 100`.

Table F4-19 Encoding table for the Data-processing register (register shift) group

Decode fields		Decode group or instruction page
op0	**op1**	
0x	x	Integer Data Processing (three register, register shift)
10	1	Integer Test and Compare (two register, register shift)
11	x	Logical Arithmetic (three register, register shift)

F4.1.6.1 Integer Data Processing (three register, register shift)

This section describes the encoding of the Integer Data Processing (three register, register shift) instruction class. The encodings in this section are decoded from Data-processing register (register shift).

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

31–28	27 26 25 24	23–21	20	19–16	15–12	11–8	7	6 5	4	3–0
!=1111	0 0 0 0	opc	S	Rn	Rd	Rs	0	stype	1	Rm
cond										

Table F4-20

Decode fields opc	Instruction page
000	AND, ANDS (register-shifted register)
001	EOR, EORS (register-shifted register)
010	SUB, SUBS (register-shifted register)
011	RSB, RSBS (register-shifted register)
100	ADD, ADDS (register-shifted register)
101	ADC, ADCS (register-shifted register)
110	SBC, SBCS (register-shifted register)
111	RSC, RSCS (register-shifted register)

F4.1.6.2 Integer Test and Compare (two register, register shift)

This section describes the encoding of the Integer Test and Compare (two register, register shift) instruction class. The encodings in this section are decoded from Data-processing register (register shift).

31–28	27 26 25 24	23 22 21	20	19–16	15 14 13 12	11–8	7	6 5	4	3–0
!=1111	0 0 0 1 0	opc	1	Rn	(0)(0)(0)(0)	Rs	0	stype	1	Rm
cond										

Table F4-21

Decode fields opc	Instruction page
00	TST (register-shifted register)
01	TEQ (register-shifted register)
10	CMP (register-shifted register)
11	CMN (register-shifted register)

F4.1.6.3 Logical Arithmetic (three register, register shift)

This section describes the encoding of the Logical Arithmetic (three register, register shift) instruction class. The encodings in this section are decoded from Data-processing register (register shift).

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

31–28	27 26 25 24	23 22 21 20	19–16	15–12	11–8	7	6 5	4	3–0
!=1111	0 0 0 1 1	opc	S	Rn	Rd	Rs	0	stype	1	Rm

cond

Table F4-22

Decode fields	Instruction page
opc	
00	ORR, ORRS (register-shifted register)
01	MOV, MOVS (register-shifted register)
10	BIC, BICS (register-shifted register)
11	MVN, MVNS (register-shifted register)

F4.1.7 Data-processing immediate

This section describes the encoding of the Data-processing immediate group. The encodings in this section are decoded from Data-processing and miscellaneous instructions.

31–28	27–25	24	23	22 21 20	19–0
!=1111	001	op0		op1	

Table F4-23 Encoding table for the Data-processing immediate group

Decode fields		Decode group or instruction page
op0	**op1**	
0x	xx	Integer Data Processing (two register and immediate)
10	00	Move Halfword (immediate)
10	10	Move Special Register and Hints (immediate)
10	x1	Integer Test and Compare (one register and immediate)
11	xx	Logical Arithmetic (two register and immediate)

F4.1.7.1 Integer Data Processing (two register and immediate)

This section describes the encoding of the Integer Data Processing (two register and immediate) instruction class. The encodings in this section are decoded from Data-processing immediate.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

31–28	27 26 25 24	23–21	20	19–16	15–12	11–0
!=1111	0 0 1 0	opc	S	Rn	Rd	imm12
cond						

Table F4-24

Decode fields			Instruction page
opc	**S**	**Rn**	
000	x	xxxx	AND, ANDS (immediate)
001	x	xxxx	EOR, EORS (immediate)
010	0	!= 11x1	SUB, SUBS (immediate) - SUB variant
010	0	1101	SUB, SUBS (SP minus immediate) - SUB variant
010	0	1111	ADR - A2
010	1	!= 1101	SUB, SUBS (immediate) - SUBS variant
010	1	1101	SUB, SUBS (SP minus immediate) - SUBS variant
011	x	xxxx	RSB, RSBS (immediate)
100	0	!= 11x1	ADD, ADDS (immediate) - ADD variant
100	0	1101	ADD, ADDS (SP plus immediate) - ADD variant
100	0	1111	ADR - A1
100	1	!= 1101	ADD, ADDS (immediate) - ADDS variant
100	1	1101	ADD, ADDS (SP plus immediate) - ADDS variant
101	x	xxxx	ADC, ADCS (immediate)
110	x	xxxx	SBC, SBCS (immediate)
111	x	xxxx	RSC, RSCS (immediate)

F4.1.7.2 Move Halfword (immediate)

This section describes the encoding of the Move Halfword (immediate) instruction class. The encodings in this section are decoded from Data-processing immediate.

31–28	27 26 25 24	23 22	21 20	19–16	15–12	11–0
!=1111	0 0 1 1 0	H	0 0	imm4	Rd	imm12
cond						

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F4-25

Decode fields	Instruction page
H	
0	MOV, MOVS (immediate)
1	MOVT

F4.1.7.3 Move Special Register and Hints (immediate)

This section describes the encoding of the Move Special Register and Hints (immediate) instruction class. The encodings in this section are decoded from Data-processing immediate.

31 28	27 26 25 24	23	22	21 20	19 16	15	14	13	12	11 0
!=1111	0 0 1 1 0	R	1	0	imm4	(1)	(1)	(1)	(1)	imm12
cond										

Table F4-26

Decode fields		Instruction page	Feature
R:imm4	**imm12**		
!= 00000	xxxxxxxxxxxx	MSR (immediate)	-
00000	xxxx00000000	NOP	-
00000	xxxx00000001	YIELD	-
00000	xxxx00000010	WFE	-
00000	xxxx00000011	WFI	-
00000	xxxx00000100	SEV	-
00000	xxxx00000101	SEVL	-
00000	xxxx0000011x	Reserved hint, behaves as NOP.	-
00000	xxxx00001xxx	Reserved hint, behaves as NOP.	-
00000	xxxx00010000	ESB	FEAT_RAS
00000	xxxx00010001	Reserved hint, behaves as NOP.	-
00000	xxxx00010010	TSB	FEAT_TRF
00000	xxxx00010011	Reserved hint, behaves as NOP.	-
00000	xxxx00010100	CSDB	-
00000	xxxx00010101	Reserved hint, behaves as NOP.	-
00000	xxxx00010110	CLRBHB	FEAT_CLRBHB
00000	xxxx00010111	Reserved hint, behaves as NOP.	-
00000	xxxx00011xxx	Reserved hint, behaves as NOP.	-
00000	xxxx001xxxxx	Reserved hint, behaves as NOP.	-
00000	xxxx01xxxxxx	Reserved hint, behaves as NOP.	-

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields		Instruction page	Feature
R:imm4	imm12		
00000	xxxx10xxxxxx	Reserved hint, behaves as NOP.	-
00000	xxxx110xxxxx	Reserved hint, behaves as NOP.	-
00000	xxxx1110xxxx	Reserved hint, behaves as NOP.	-
00000	xxxx1111xxxx	DBG	-

F4.1.7.4 Integer Test and Compare (one register and immediate)

This section describes the encoding of the Integer Test and Compare (one register and immediate) instruction class. The encodings in this section are decoded from Data-processing immediate.

31–28	27 26 25 24	23 22 21 20	19–16	15 14 13 12	11–0
!=1111	0 0 1 1 0	opc	1	Rn	(0)(0)(0)(0)	imm12
cond					

Table F4-27

Decode fields	Instruction page
opc	
00	TST (immediate)
01	TEQ (immediate)
10	CMP (immediate)
11	CMN (immediate)

F4.1.7.5 Logical Arithmetic (two register and immediate)

This section describes the encoding of the Logical Arithmetic (two register and immediate) instruction class. The encodings in this section are decoded from Data-processing immediate.

31–28	27 26 25 24	23 22 21 20	19–16	15–12	11–0
!=1111	0 0 1 1 1	opc	S	Rn	Rd	imm12
cond					

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F4-28

Decode fields opc	Instruction page
00	ORR, ORRS (immediate)
01	MOV, MOVS (immediate)
10	BIC, BICS (immediate)
11	MVN, MVNS (immediate)

F4.1.8 Load/Store Word, Unsigned Byte (immediate, literal)

This section describes the encoding of the Load/Store Word, Unsigned Byte (immediate, literal) instruction class. The encodings in this section are decoded from A32 instruction set encoding.

31 28	27 26 25	24	23	22	21	20	19 16	15 12	11 0
!=1111 cond	0 1 0	P	U	o2	W	o1	Rn	Rt	imm12

Table F4-29

Decode fields P:W	o2	o1	Rn	Instruction page
!= 01	0	1	1111	LDR (literal)
!= 01	1	1	1111	LDRB (literal)
00	0	0	xxxx	STR (immediate) - Post-indexed variant
00	0	1	!= 1111	LDR (immediate) - Post-indexed variant
00	1	0	xxxx	STRB (immediate) - Post-indexed variant
00	1	1	!= 1111	LDRB (immediate) - Post-indexed variant
01	0	0	xxxx	STRT
01	0	1	xxxx	LDRT
01	1	0	xxxx	STRBT
01	1	1	xxxx	LDRBT
10	0	0	xxxx	STR (immediate) - Offset variant
10	0	1	!= 1111	LDR (immediate) - Offset variant
10	1	0	xxxx	STRB (immediate) - Offset variant
10	1	1	!= 1111	LDRB (immediate) - Offset variant
11	0	0	xxxx	STR (immediate) - Pre-indexed variant
11	0	1	!= 1111	LDR (immediate) - Pre-indexed variant

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields				Instruction page
P:W	o2	o1	Rn	
11	1	0	xxxx	STRB (immediate) - Pre-indexed variant
11	1	1	!= 1111	LDRB (immediate) - Pre-indexed variant

F4.1.9 Load/Store Word, Unsigned Byte (register)

This section describes the encoding of the Load/Store Word, Unsigned Byte (register) instruction class. The encodings in this section are decoded from A32 instruction set encoding.

31–28	27	26	25	24	23	22	21	20	19–16	15–12	11–7	6–5	4	3–0
!=1111	0	1	1	P	U	o2	W	o1	Rn	Rt	imm5	stype	0	Rm
cond														

Table F4-30

Decode fields				Instruction page
P	o2	W	o1	
0	0	0	0	STR (register) - Post-indexed variant
0	0	0	1	LDR (register) - Post-indexed variant
0	0	1	0	STRT
0	0	1	1	LDRT
0	1	0	0	STRB (register) - Post-indexed variant
0	1	0	1	LDRB (register) - Post-indexed variant
0	1	1	0	STRBT
0	1	1	1	LDRBT
1	0	x	0	STR (register) - Pre-indexed variant
1	0	x	1	LDR (register) - Pre-indexed variant
1	1	x	0	STRB (register) - Pre-indexed variant
1	1	x	1	LDRB (register) - Pre-indexed variant

F4.1.10 Media instructions

This section describes the encoding of the Media instructions group. The encodings in this section are decoded from A32 instruction set encoding.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

31–28	27–25	24–20	19–8	7–5	4	3–0
!=1111	011	op0		op1	1	

Table F4-31 Encoding table for the Media instructions group

Decode fields		Decode group or instruction page
op0	op1	
00xxx	xxx	Parallel Arithmetic
01000	101	SEL
01000	001	Unallocated.
01000	xx0	PKHBT, PKHTB
01001	x01	Unallocated.
01001	xx0	Unallocated.
0110x	x01	Unallocated.
0110x	xx0	Unallocated.
01x10	001	Saturate 16-bit
01x10	101	Unallocated.
01x11	x01	Reverse Bit/Byte
01x1x	xx0	Saturate 32-bit
01xxx	111	Unallocated.
01xxx	011	Extend and Add
10xxx	xxx	Signed multiply, Divide
11000	000	Unsigned Sum of Absolute Differences
11000	100	Unallocated.
11001	x00	Unallocated.
1101x	x00	Unallocated.
110xx	111	Unallocated.
1110x	111	Unallocated.
1110x	x00	Bitfield Insert
11110	111	Unallocated.
11111	111	Permanently UNDEFINED
1111x	x00	Unallocated.
11x0x	x10	Unallocated.
11x1x	x10	Bitfield Extract
11xxx	011	Unallocated.
11xxx	x01	Unallocated.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F4.1.10.1 Parallel Arithmetic

This section describes the encoding of the Parallel Arithmetic instruction class. The encodings in this section are decoded from Media instructions.

31–28	27 26 25 24	23 22–20	19–16	15–12	11	10	9	8	7	6 5	4	3–0
!=1111	0 1 1 0	0	op1	Rn	Rd	(1)	(1)	(1)	(1)	B	op2	1	Rm
cond												

Table F4-32

Decode fields			Instruction page
op1	B	op2	
000	x	xx	Unallocated.
001	0	00	SADD16
001	0	01	SASX
001	0	10	SSAX
001	0	11	SSUB16
001	1	00	SADD8
001	1	01	Unallocated.
001	1	10	Unallocated.
001	1	11	SSUB8
010	0	00	QADD16
010	0	01	QASX
010	0	10	QSAX
010	0	11	QSUB16
010	1	00	QADD8
010	1	01	Unallocated.
010	1	10	Unallocated.
010	1	11	QSUB8
011	0	00	SHADD16
011	0	01	SHASX
011	0	10	SHSAX
011	0	11	SHSUB16
011	1	00	SHADD8
011	1	01	Unallocated.
011	1	10	Unallocated.
011	1	11	SHSUB8
100	x	xx	Unallocated.
101	0	00	UADD16

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields			Instruction page
op1	B	op2	
101	0	01	UASX
101	0	10	USAX
101	0	11	USUB16
101	1	00	UADD8
101	1	01	Unallocated.
101	1	10	Unallocated.
101	1	11	USUB8
110	0	00	UQADD16
110	0	01	UQASX
110	0	10	UQSAX
110	0	11	UQSUB16
110	1	00	UQADD8
110	1	01	Unallocated.
110	1	10	Unallocated.
110	1	11	UQSUB8
111	0	00	UHADD16
111	0	01	UHASX
111	0	10	UHSAX
111	0	11	UHSUB16
111	1	00	UHADD8
111	1	01	Unallocated.
111	1	10	Unallocated.
111	1	11	UHSUB8

F4.1.10.2 Saturate 16-bit

This section describes the encoding of the Saturate 16-bit instruction class. The encodings in this section are decoded from Media instructions.

31 28	27 26 25 24	23	22	21 20	19 16	15 12	11	10	9	8	7 6 5 4	3 0
!=1111	0 1 1 0 1	U		1 0	sat_imm	Rd	(1)	(1)	(1)	(1)	0 0 1 1	Rn
cond												

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F4-33

Decode fields	Instruction page
U	
0	SSAT16
1	USAT16

F4.1.10.3 Reverse Bit/Byte

This section describes the encoding of the Reverse Bit/Byte instruction class. The encodings in this section are decoded from Media instructions.

31 28	27 26 25 24	23	22	21 20	19 18 17 16	15 12	11 10 9 8	7	6 5 4	3 0
!=1111	0 1 1 0	1	o1	1 1	(1)(1)(1)(1)	Rd	(1)(1)(1)(1)	o2	0 1 1	Rm
cond										

Table F4-34

Decode fields		Instruction page
o1	**o2**	
0	0	REV
0	1	REV16
1	0	RBIT
1	1	REVSH

F4.1.10.4 Saturate 32-bit

This section describes the encoding of the Saturate 32-bit instruction class. The encodings in this section are decoded from Media instructions.

31 28	27 26 25 24	23	22	21	20 16	15 12	11 7	6	5 4	3 0
!=1111	0 1 1 0	1	U	1	sat_imm	Rd	imm5	sh	0 1	Rn
cond										

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F4-35

Decode fields	Instruction page
U	
0	SSAT
1	USAT

F4.1.10.5 Extend and Add

This section describes the encoding of the Extend and Add instruction class. The encodings in this section are decoded from Media instructions.

31–28	27 26 25 24	23	22	21 20	19–16	15–12	11 10	9	8	7 6 5 4	3–0
!=1111	0 1 1 0	1	U	op	Rn	Rd	rotate	(0)	(0)	0 1 1 1	Rm
cond											

Table F4-36

Decode fields			Instruction page
U	**op**	**Rn**	
0	00	!= 1111	SXTAB16
0	00	1111	SXTB16
0	10	!= 1111	SXTAB
0	10	1111	SXTB
0	11	!= 1111	SXTAH
0	11	1111	SXTH
1	00	!= 1111	UXTAB16
1	00	1111	UXTB16
1	10	!= 1111	UXTAB
1	10	1111	UXTB
1	11	!= 1111	UXTAH
1	11	1111	UXTH

F4.1.10.6 Signed multiply, Divide

This section describes the encoding of the Signed multiply, Divide instruction class. The encodings in this section are decoded from Media instructions.

31–28	27 26 25 24	23 22 20	19–16	15–12	11–8	7–5	4	3–0
!=1111	0 1 1 1	0 op1	Rd	Ra	Rm	op2	1	Rn
cond								

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F4-37

Decode fields			Instruction page
op1	Ra	op2	
000	!= 1111	000	SMLAD, SMLADX - SMLAD variant
000	!= 1111	001	SMLAD, SMLADX - SMLADX variant
000	!= 1111	010	SMLSD, SMLSDX - SMLSD variant
000	!= 1111	011	SMLSD, SMLSDX - SMLSDX variant
000	xxxx	1xx	Unallocated.
000	1111	000	SMUAD, SMUADX - SMUAD variant
000	1111	001	SMUAD, SMUADX - SMUADX variant
000	1111	010	SMUSD, SMUSDX - SMUSD variant
000	1111	011	SMUSD, SMUSDX - SMUSDX variant
001	xxxx	000	SDIV
001	xxxx	!= 000	Unallocated.
010	xxxx	xxx	Unallocated.
011	xxxx	000	UDIV
011	xxxx	!= 000	Unallocated.
100	xxxx	000	SMLALD, SMLALDX - SMLALD variant
100	xxxx	001	SMLALD, SMLALDX - SMLALDX variant
100	xxxx	010	SMLSLD, SMLSLDX - SMLSLD variant
100	xxxx	011	SMLSLD, SMLSLDX - SMLSLDX variant
100	xxxx	1xx	Unallocated.
101	!= 1111	000	SMMLA, SMMLAR - SMMLA variant
101	!= 1111	001	SMMLA, SMMLAR - SMMLAR variant
101	xxxx	01x	Unallocated.
101	xxxx	10x	Unallocated.
101	xxxx	110	SMMLS, SMMLSR - SMMLS variant
101	xxxx	111	SMMLS, SMMLSR - SMMLSR variant
101	1111	000	SMMUL, SMMULR - SMMUL variant
101	1111	001	SMMUL, SMMULR - SMMULR variant
11x	xxxx	xxx	Unallocated.

F4.1.10.7 Unsigned Sum of Absolute Differences

This section describes the encoding of the Unsigned Sum of Absolute Differences instruction class. The encodings in this section are decoded from Media instructions.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

31–28	27 26 25 24	23 22 21 20	19–16	15–12	11–8	7 6 5 4	3–0
!=1111	0 1 1 1	1 0 0 0	Rd	Ra	Rm	0 0 0 1	Rn

cond

Table F4-38

Decode fields Ra	Instruction page
`!= 1111`	USADA8
`1111`	USAD8

F4.1.10.8 Bitfield Insert

This section describes the encoding of the Bitfield Insert instruction class. The encodings in this section are decoded from Media instructions.

31–28	27 26 25 24	23 22 21 20	16	15–12	11–	7 6 5 4	3–0
!=1111	0 1 1 1	1 1 0	msb	Rd	lsb	0 0 1	Rn

cond

Table F4-39

Decode fields Rn	Instruction page
`!= 1111`	BFI
`1111`	BFC

F4.1.10.9 Permanently UNDEFINED

This section describes the encoding of the Permanently UNDEFINED instruction class. The encodings in this section are decoded from Media instructions.

31–28	27 26 25 24	23 22 21 20	19–	–8	7 6 5 4	3–0
!=1111	0 1 1 1	1 1 1 1	imm12		1 1 1 1	imm4

cond

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F4-40

Decode fields cond	Instruction page
`0xxx`	Unallocated.
`10xx`	Unallocated.
`110x`	Unallocated.
`1110`	UDF

F4.1.10.10 Bitfield Extract

This section describes the encoding of the Bitfield Extract instruction class. The encodings in this section are decoded from Media instructions.

Table F4-41

Decode fields U	Instruction page
`0`	SBFX
`1`	UBFX

F4.1.11 Branch, branch with link, and block data transfer

This section describes the encoding of the Branch, branch with link, and block data transfer group. The encodings in this section are decoded from A32 instruction set encoding.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F4-42 Encoding table for the Branch, branch with link, and block data transfer group

Decode fields		Decode group or instruction page
cond	**op0**	
1111	0	Exception Save/Restore
!= 1111	0	Load/Store Multiple
xxxx	1	Branch (immediate)

F4.1.11.1 Exception Save/Restore

This section describes the encoding of the Exception Save/Restore instruction class. The encodings in this section are decoded from Branch, branch with link, and block data transfer.

31 30 29 28	27 26 25	24	23	22	21	20	19 16	15 5	4 0
1 1 1 1	1 0 0	P	U	S	W	L	Rn	op	mode

Table F4-43

Decode fields				Instruction page
P	**U**	**S**	**L**	
x	x	0	0	Unallocated.
0	0	0	1	RFE, RFEDA, RFEDB, RFEIA, RFEIB - Decrement After variant
0	0	1	0	SRS, SRSDA, SRSDB, SRSIA, SRSIB - Decrement After variant
0	1	0	1	RFE, RFEDA, RFEDB, RFEIA, RFEIB - Increment After variant
0	1	1	0	SRS, SRSDA, SRSDB, SRSIA, SRSIB - Increment After variant
1	0	0	1	RFE, RFEDA, RFEDB, RFEIA, RFEIB - Decrement Before variant
1	0	1	0	SRS, SRSDA, SRSDB, SRSIA, SRSIB - Decrement Before variant
x	x	1	1	Unallocated.
1	1	0	1	RFE, RFEDA, RFEDB, RFEIA, RFEIB - Increment Before variant
1	1	1	0	SRS, SRSDA, SRSDB, SRSIA, SRSIB - Increment Before variant

F4.1.11.2 Load/Store Multiple

This section describes the encoding of the Load/Store Multiple instruction class. The encodings in this section are decoded from Branch, branch with link, and block data transfer.

31 28	27 26 25	24	23	22	21	20	19 16	15 0
!=1111	1 0 0	P	U	op	W	L	Rn	register_list
cond								

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F4-44

Decode fields					Instruction page
P	**U**	**op**	**L**	**register_list**	
0	0	0	0	xxxxxxxxxxxxxxxx	STMDA, STMED
0	0	0	1	xxxxxxxxxxxxxxxx	LDMDA, LDMFA
0	1	0	0	xxxxxxxxxxxxxxxx	STM, STMIA, STMEA
0	1	0	1	xxxxxxxxxxxxxxxx	LDM, LDMIA, LDMFD
x	x	1	0	xxxxxxxxxxxxxxxx	STM (User registers)
1	0	0	0	xxxxxxxxxxxxxxxx	STMDB, STMFD
1	0	0	1	xxxxxxxxxxxxxxxx	LDMDB, LDMEA
x	x	1	1	0xxxxxxxxxxxxxxx	LDM (User registers)
1	1	0	0	xxxxxxxxxxxxxxxx	STMIB, STMFA
1	1	0	1	xxxxxxxxxxxxxxxx	LDMIB, LDMED
x	x	1	1	1xxxxxxxxxxxxxxx	LDM (exception return)

F4.1.11.3 Branch (immediate)

This section describes the encoding of the Branch (immediate) instruction class. The encodings in this section are decoded from Branch, branch with link, and block data transfer.

31 28	27 26 25	24	23 0
cond	1 0 1	H	imm24

Table F4-45

Decode fields		Instruction page
cond	**H**	
!= 1111	0	B
!= 1111	1	BL, BLX (immediate) - A1
1111	x	BL, BLX (immediate) - A2

F4.1.12 System register access, Advanced SIMD, floating-point, and Supervisor call

This section describes the encoding of the System register access, Advanced SIMD, floating-point, and Supervisor call group. The encodings in this section are decoded from A32 instruction set encoding.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F4-46 Encoding table for the System register access, Advanced SIMD, floating-point, and Supervisor call group

Decode fields				Decode group or instruction page
cond	**op0**	**op1**	**op2**	
`xxxx`	`0x`	`0x`	`x`	Unallocated.
`xxxx`	`10`	`0x`	`x`	Unallocated.
`xxxx`	`11`	`xx`	`x`	Supervisor call
`1111`	`!= 11`	`1x`	`x`	Unconditional Advanced SIMD and floating-point instructions
`!= 1111`	`0x`	`1x`	`x`	Advanced SIMD and System register load/store and 64-bit move
`!= 1111`	`10`	`1x`	`1`	Advanced SIMD and System register 32-bit move
`!= 1111`	`10`	`10`	`0`	Floating-point data-processing
`!= 1111`	`10`	`11`	`0`	Unallocated.

F4.1.13 Supervisor call

This section describes the encoding of the Supervisor call group. The encodings in this section are decoded from System register access, Advanced SIMD, floating-point, and Supervisor call.

Table F4-47 Encoding table for the Supervisor call group

Decode fields	Decode group or instruction page
cond	
`1111`	Unallocated.
`!= 1111`	SVC

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F4.1.14 Unconditional Advanced SIMD and floating-point instructions

This section describes the encoding of the Unconditional Advanced SIMD and floating-point instructions group. The encodings in this section are decoded from System register access, Advanced SIMD, floating-point, and Supervisor call.

This decode also imposes the constraint:

- `op0<2:1> != 11`.

Table F4-48 Encoding table for the Unconditional Advanced SIMD and floating-point instructions group

Decode fields						Decode group or instruction page
op0	op1	op2	op3	op4	op5	
0xx	xxxxxx	x	0x	x	x	Advanced SIMD three registers of the same length extension
100	xxxxxx	0	!= 00	0	0	Floating-point conditional select
101	00xxxx	0	!= 00	x	0	Floating-point minNum/maxNum
101	110000	0	!= 00	1	0	Floating-point extraction and insertion
101	111xxx	0	!= 00	1	0	Floating-point directed convert to integer
10x	xxxxxx	0	00	x	x	Advanced SIMD and floating-point multiply with accumulate
10x	xxxxxx	1	0x	x	x	Advanced SIMD and floating-point dot product

F4.1.14.1 Advanced SIMD three registers of the same length extension

This section describes the encoding of the Advanced SIMD three registers of the same length extension instruction class. The encodings in this section are decoded from Unconditional Advanced SIMD and floating-point instructions.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F4-49

Decode fields						Instruction page	Feature
op1	op2	op3	op4	Q	U		
x1	0x	0	0	0	0	VCADD - 64-bit SIMD vector variant	FEAT_FCMA
x1	0x	0	0	0	1	Unallocated.	-
x1	0x	0	0	1	0	VCADD - 128-bit SIMD vector variant	FEAT_FCMA
x1	0x	0	0	1	1	Unallocated.	-
00	0x	0	0	x	x	Unallocated.	-
00	0x	0	1	x	x	Unallocated.	-
00	00	1	0	0	0	Unallocated.	-
00	00	1	0	0	1	Unallocated.	-
00	00	1	0	1	0	VMMLA	FEAT_AA32BF16
00	00	1	0	1	1	Unallocated.	-
00	00	1	1	0	0	VDOT (vector) - 64-bit SIMD vector variant	FEAT_AA32BF16
00	00	1	1	0	1	Unallocated.	-
00	00	1	1	1	0	VDOT (vector) - 128-bit SIMD vector variant	FEAT_AA32BF16
00	00	1	1	1	1	Unallocated.	-
00	01	1	0	x	x	Unallocated.	-
00	01	1	1	x	x	Unallocated.	-
00	10	0	0	x	1	VFMAL (vector)	FEAT_FHM
00	10	0	1	x	x	Unallocated.	-
00	10	1	0	0	x	Unallocated.	-
00	10	1	0	1	0	VSMMLA	FEAT_AA32I8MM
00	10	1	0	1	1	VUMMLA	FEAT_AA32I8MM
00	10	1	1	0	0	VSDOT (vector) - 64-bit SIMD vector variant	FEAT_DotProd
00	10	1	1	0	1	VUDOT (vector) - 64-bit SIMD vector variant	FEAT_DotProd
00	10	1	1	1	0	VSDOT (vector) - 128-bit SIMD vector variant	FEAT_DotProd
00	10	1	1	1	1	VUDOT (vector) - 128-bit SIMD vector variant	FEAT_DotProd
00	11	0	0	x	1	VFMAB, VFMAT (BFloat16, vector)	FEAT_AA32BF16
00	11	0	1	x	x	Unallocated.	-
00	11	1	0	x	x	Unallocated.	-
00	11	1	1	x	x	Unallocated.	-
01	10	0	0	x	1	VFMSL (vector)	FEAT_FHM
01	10	0	1	x	x	Unallocated.	-
01	10	1	0	0	x	Unallocated.	-
01	10	1	0	1	0	VUSMMLA	FEAT_AA32I8MM
01	10	1	0	1	1	Unallocated.	-
01	10	1	1	0	0	VUSDOT (vector) - 64-bit SIMD vector variant	FEAT_AA32I8MM
01	10	1	1	x	1	Unallocated.	-
01	10	1	1	1	0	VUSDOT (vector) - 128-bit SIMD vector variant	FEAT_AA32I8MM

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields						Instruction page	Feature
op1	op2	op3	op4	Q	U		
01	11	0	1	x	x	Unallocated.	-
01	11	1	0	x	x	Unallocated.	-
01	11	1	1	x	x	Unallocated.	-
xx	1x	0	0	x	0	VCMLA	FEAT_FCMA
10	11	0	1	x	x	Unallocated.	-
10	11	1	0	x	x	Unallocated.	-
10	11	1	1	x	x	Unallocated.	-
11	11	0	1	x	x	Unallocated.	-
11	11	1	0	x	x	Unallocated.	-
11	11	1	1	x	x	Unallocated.	-

F4.1.14.2 Floating-point conditional select

This section describes the encoding of the Floating-point conditional select instruction class. The encodings in this section are decoded from Unconditional Advanced SIMD and floating-point instructions.

31 30 29 28	27 26 25 24	23	22	21 20	19 16	15 12	11 10	9 8	7	6	5	4	3 0
1 1 1 1	1 1 1 0	0	D	cc	Vn	Vd	1 0	!=00	N	0	M	0	Vm
								size					

Table F4-50

Decode fields	Instruction page	Feature
size		
01	VSELEQ, VSELGE, VSELGT, VSELVS - Greater than, half-precision scalar variant	FEAT_FP16
10	VSELEQ, VSELGE, VSELGT, VSELVS - Greater than, single-precision scalar variant	-
11	VSELEQ, VSELGE, VSELGT, VSELVS - Greater than, double-precision scalar variant	-

F4.1.14.3 Floating-point minNum/maxNum

This section describes the encoding of the Floating-point minNum/maxNum instruction class. The encodings in this section are decoded from Unconditional Advanced SIMD and floating-point instructions.

31 30 29 28	27 26 25 24	23	22	21 20	19 16	15 12	11 10	9 8	7	6	5	4	3 0
1 1 1 1	1 1 1 0	1	D	0 0	Vn	Vd	1 0	!=00	N	op	M	0	Vm
								size					

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F4-51

Decode fields		Instruction page	Feature
size	op		
01	0	VMAXNM - Half-precision scalar variant	FEAT_FP16
01	1	VMINNM - Half-precision scalar variant	FEAT_FP16
10	0	VMAXNM - Single-precision scalar variant	-
10	1	VMINNM - Single-precision scalar variant	-
11	0	VMAXNM - Double-precision scalar variant	-
11	1	VMINNM - Double-precision scalar variant	-

F4.1.14.4 Floating-point extraction and insertion

This section describes the encoding of the Floating-point extraction and insertion instruction class. The encodings in this section are decoded from Unconditional Advanced SIMD and floating-point instructions.

31 30 29 28	27 26 25 24	23	22	21 20	19 18 17 16	15 12	11 10	9 8	7	6	5	4	3 0
1 1 1 1	1 1 1 0	1	D	1 1	0 0 0 0	Vd	1 0	!=00	op	1	M	0	Vm
								size					

Table F4-52

Decode fields		Instruction page	Feature
size	op		
01	x	Unallocated.	-
10	0	VMOVX	FEAT_FP16
10	1	VINS	FEAT_FP16
11	x	Unallocated.	-

F4.1.14.5 Floating-point directed convert to integer

This section describes the encoding of the Floating-point directed convert to integer instruction class. The encodings in this section are decoded from Unconditional Advanced SIMD and floating-point instructions.

31 30 29 28	27 26 25 24	23	22	21 20	19	18	17 16	15 12	11 10	9 8	7	6	5	4	3 0
1 1 1 1	1 1 1 0	1	D	1 1	1	o1	RM	Vd	1 0	!=00	op	1	M	0	Vm
										size					

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F4-53

Decode fields				Instruction page	Feature
o1	RM	size	op		
0	xx	!= 00	1	Unallocated.	-
0	00	01	0	VRINTA (floating-point) - Half-precision scalar variant	FEAT_FP16
0	00	10	0	VRINTA (floating-point) - Single-precision scalar variant	-
0	00	11	0	VRINTA (floating-point) - Double-precision scalar variant	-
0	01	01	0	VRINTN (floating-point) - Half-precision scalar variant	FEAT_FP16
0	01	10	0	VRINTN (floating-point) - Single-precision scalar variant	-
0	01	11	0	VRINTN (floating-point) - Double-precision scalar variant	-
0	10	01	0	VRINTP (floating-point) - Half-precision scalar variant	FEAT_FP16
0	10	10	0	VRINTP (floating-point) - Single-precision scalar variant	-
0	10	11	0	VRINTP (floating-point) - Double-precision scalar variant	-
0	11	01	0	VRINTM (floating-point) - Half-precision scalar variant	FEAT_FP16
0	11	10	0	VRINTM (floating-point) - Single-precision scalar variant	-
0	11	11	0	VRINTM (floating-point) - Double-precision scalar variant	-
1	00	01	x	VCVTA (floating-point) - Half-precision scalar variant	FEAT_FP16
1	00	10	x	VCVTA (floating-point) - Single-precision scalar variant	-
1	00	11	x	VCVTA (floating-point) - Double-precision scalar variant	-
1	01	01	x	VCVTN (floating-point) - Half-precision scalar variant	FEAT_FP16
1	01	10	x	VCVTN (floating-point) - Single-precision scalar variant	-
1	01	11	x	VCVTN (floating-point) - Double-precision scalar variant	-
1	10	01	x	VCVTP (floating-point) - Half-precision scalar variant	FEAT_FP16
1	10	10	x	VCVTP (floating-point) - Single-precision scalar variant	-
1	10	11	x	VCVTP (floating-point) - Double-precision scalar variant	-
1	11	01	x	VCVTM (floating-point) - Half-precision scalar variant	FEAT_FP16
1	11	10	x	VCVTM (floating-point) - Single-precision scalar variant	-
1	11	11	x	VCVTM (floating-point) - Double-precision scalar variant	-

F4.1.14.6 Advanced SIMD and floating-point multiply with accumulate

This section describes the encoding of the Advanced SIMD and floating-point multiply with accumulate instruction class. The encodings in this section are decoded from Unconditional Advanced SIMD and floating-point instructions.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F4-54

Decode fields				Instruction page	Feature
op1	op2	Q	U		
0	xx	x	0	VCMLA (by element) - 128-bit SIMD vector of half-precision floating-point variant	FEAT_FCMA
0	00	x	1	VFMAL (by scalar)	FEAT_FHM
0	01	x	1	VFMSL (by scalar)	FEAT_FHM
0	10	x	1	Unallocated.	-
0	11	x	1	VFMAB, VFMAT (BFloat16, by scalar)	FEAT_AA32BF16
1	xx	0	0	VCMLA (by element) - 64-bit SIMD vector of single-precision floating-point variant	FEAT_FCMA
1	xx	x	1	Unallocated.	-
1	xx	1	0	VCMLA (by element) - 128-bit SIMD vector of single-precision floating-point variant	FEAT_FCMA

F4.1.14.7 Advanced SIMD and floating-point dot product

This section describes the encoding of the Advanced SIMD and floating-point dot product instruction class. The encodings in this section are decoded from Unconditional Advanced SIMD and floating-point instructions.

Table F4-55

Decode fields					Instruction page	Feature
op1	op2	op4	Q	U		
0	00	0	x	x	Unallocated.	-
0	00	1	0	0	VDOT (by element) - 64-bit SIMD vector variant	FEAT_AA32BF16
0	00	1	x	1	Unallocated.	-
0	00	1	1	0	VDOT (by element) - 128-bit SIMD vector variant	FEAT_AA32BF16
0	01	0	x	x	Unallocated.	-
0	10	0	x	x	Unallocated.	-
0	10	1	0	0	VSDOT (by element) - 64-bit SIMD vector variant	FEAT_DotProd
0	10	1	0	1	VUDOT (by element) - 64-bit SIMD vector variant	FEAT_DotProd
0	10	1	1	0	VSDOT (by element) - 128-bit SIMD vector variant	FEAT_DotProd
0	10	1	1	1	VUDOT (by element) - 128-bit SIMD vector variant	FEAT_DotProd
0	11	x	x	x	Unallocated.	-

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields					Instruction page	Feature
op1	op2	op4	Q	U		
1	xx	0	x	x	Unallocated.	-
1	00	1	0	0	VUSDOT (by element) - 64-bit SIMD vector variant	FEAT_AA32I8MM
1	00	1	0	1	VSUDOT (by element) - 64-bit SIMD vector variant	FEAT_AA32I8MM
1	00	1	1	0	VUSDOT (by element) - 128-bit SIMD vector variant	FEAT_AA32I8MM
1	00	1	1	1	VSUDOT (by element) - 128-bit SIMD vector variant	FEAT_AA32I8MM
1	01	1	x	x	Unallocated.	-
1	1x	1	x	x	Unallocated.	-

F4.1.15 Advanced SIMD and System register load/store and 64-bit move

This section describes the encoding of the Advanced SIMD and System register load/store and 64-bit move group. The encodings in this section are decoded from System register access, Advanced SIMD, floating-point, and Supervisor call.

This group has encodings in both the T32 and A32 instruction sets. For information about mappings between the encodings of this group, see About the A32 Advanced SIMD and floating-point instructions and their encoding

31–28	27–25	24–21	20–12	11	10–9	8–0
!=1111	110	op0		1	op1	

Table F4-56 Encoding table for the Advanced SIMD and System register load/store and 64-bit move group

Decode fields		Decode group or instruction page
op0	op1	
00x0	0x	Advanced SIMD and floating-point 64-bit move
00x0	11	System register 64-bit move
!= 00x0	0x	Advanced SIMD and floating-point load/store
!= 00x0	11	System register load/store
xxxx	10	Unallocated.

F4.1.15.1 Advanced SIMD and floating-point 64-bit move

This section describes the encoding of the Advanced SIMD and floating-point 64-bit move instruction class. The encodings in this section are decoded from Advanced SIMD and System register load/store and 64-bit move.

31–28	27	26	25	24	23	22	21	20	19–16	15–12	11	10	9–8	7	6	5	4	3–0
!=1111	1	1	0	0	0	D	0	op	Rt2	Rt	1	0	size	opc2		M	o3	Vm
cond																		

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F4-57

Decode fields					Instruction page
D	**op**	**size**	**opc2**	**o3**	
0	x	xx	xx	x	Unallocated.
1	x	xx	xx	0	Unallocated.
1	x	0x	00	1	Unallocated.
1	x	xx	01	x	Unallocated.
1	0	10	00	1	VMOV (between two general-purpose registers and two single-precision registers) - From general-purpose registers variant
1	0	11	00	1	VMOV (between two general-purpose registers and a doubleword floating-point register) - From general-purpose registers variant
1	x	xx	1x	x	Unallocated.
1	1	10	00	1	VMOV (between two general-purpose registers and two single-precision registers) - To general-purpose registers variant
1	1	11	00	1	VMOV (between two general-purpose registers and a doubleword floating-point register) - To general-purpose registers variant

F4.1.15.2 System register 64-bit move

This section describes the encoding of the System register 64-bit move instruction class. The encodings in this section are decoded from Advanced SIMD and System register load/store and 64-bit move.

Table F4-58

Decode fields		Instruction page
D	**L**	
0	x	Unallocated.
1	0	MCRR
1	1	MRRC

F4.1.15.3 Advanced SIMD and floating-point load/store

This section describes the encoding of the Advanced SIMD and floating-point load/store instruction class. The encodings in this section are decoded from Advanced SIMD and System register load/store and 64-bit move.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

31–28	27	26	25	24	23	22	21	20	19–16	15–12	11	10	9–8	7–0
!=1111	1	1	0	P	U	D	W	L	Rn	Vd	1	0	size	imm8
cond														

Table F4-59

Decode fields							Instruction page	Feature
P	**U**	**W**	**L**	**Rn**	**size**	**imm8**		
0	0	1	x	xxxx	xx	xxxxxxxx	Unallocated.	-
0	1	x	x	xxxx	0x	xxxxxxxx	Unallocated.	-
0	1	x	0	xxxx	10	xxxxxxxx	VSTM, VSTMDB, VSTMIA - Increment After variant	-
0	1	x	0	xxxx	11	xxxxxxx0	VSTM, VSTMDB, VSTMIA - Increment After variant	-
0	1	x	0	xxxx	11	xxxxxxx1	FSTMDBX, FSTMIAX - Increment After variant	-
0	1	x	1	xxxx	10	xxxxxxxx	VLDM, VLDMDB, VLDMIA - Increment After variant	-
0	1	x	1	xxxx	11	xxxxxxx0	VLDM, VLDMDB, VLDMIA - Increment After variant	-
0	1	x	1	xxxx	11	xxxxxxx1	FLDM*X (FLDMDBX, FLDMIAX) - Increment After variant	-
1	x	0	0	xxxx	01	xxxxxxxx	VSTR - Half-precision scalar variant	FEAT_FP16
1	x	0	0	xxxx	10	xxxxxxxx	VSTR - Single-precision scalar variant	-
1	x	0	0	xxxx	11	xxxxxxxx	VSTR - Double-precision scalar variant	-
1	x	0	1	!= 1111	01	xxxxxxxx	VLDR (immediate) - Half-precision scalar variant	FEAT_FP16
1	x	0	1	!= 1111	10	xxxxxxxx	VLDR (immediate) - Single-precision scalar variant	-
1	x	0	1	!= 1111	11	xxxxxxxx	VLDR (immediate) - Double-precision scalar variant	-
1	0	1	x	xxxx	0x	xxxxxxxx	Unallocated.	-
1	0	1	0	xxxx	10	xxxxxxxx	VSTM, VSTMDB, VSTMIA - Decrement Before variant	-
1	0	1	0	xxxx	11	xxxxxxx0	VSTM, VSTMDB, VSTMIA - Decrement Before variant	-
1	0	1	0	xxxx	11	xxxxxxx1	FSTMDBX, FSTMIAX - Decrement Before variant	-
1	0	1	1	xxxx	10	xxxxxxxx	VLDM, VLDMDB, VLDMIA - Decrement Before variant	-
1	0	1	1	xxxx	11	xxxxxxx0	VLDM, VLDMDB, VLDMIA - Decrement Before variant	-
1	0	1	1	xxxx	11	xxxxxxx1	FLDM*X (FLDMDBX, FLDMIAX) - Decrement Before variant	-
1	x	0	1	1111	01	xxxxxxxx	VLDR (literal) - Half-precision scalar variant	FEAT_FP16
1	x	0	1	1111	10	xxxxxxxx	VLDR (literal) - Single-precision scalar variant	-
1	x	0	1	1111	11	xxxxxxxx	VLDR (literal) - Double-precision scalar variant	-

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields							Instruction page	Feature
P	U	W	L	Rn	size	imm8		
1	1	1	x	xxxx	xx	xxxxxxxx	Unallocated.	-

F4.1.15.4 System register load/store

This section describes the encoding of the System register load/store instruction class. The encodings in this section are decoded from Advanced SIMD and System register load/store and 64-bit move.

Table F4-60

Decode fields						Instruction page
P:U:W	D	L	Rn	CRd	cp15	
!= 000	0	x	xxxx	!= 0101	0	Unallocated.
!= 000	0	1	1111	0101	0	LDC (literal)
!= 000	x	x	xxxx	xxxx	1	Unallocated.
!= 000	1	x	xxxx	0101	0	Unallocated.
0x1	0	0	xxxx	0101	0	STC - Post-indexed variant
0x1	0	1	!= 1111	0101	0	LDC (immediate) - Post-indexed variant
010	0	0	xxxx	0101	0	STC - Unindexed variant
010	0	1	!= 1111	0101	0	LDC (immediate) - Unindexed variant
1x0	0	0	xxxx	0101	0	STC - Offset variant
1x0	0	1	!= 1111	0101	0	LDC (immediate) - Offset variant
1x1	0	0	xxxx	0101	0	STC - Pre-indexed variant
1x1	0	1	!= 1111	0101	0	LDC (immediate) - Pre-indexed variant

F4.1.16 Advanced SIMD and System register 32-bit move

This section describes the encoding of the Advanced SIMD and System register 32-bit move group. The encodings in this section are decoded from System register access, Advanced SIMD, floating-point, and Supervisor call.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F4-61 Encoding table for the Advanced SIMD and System register 32-bit move group

Decode fields		Decode group or instruction page	Feature
op0	op1		
000	000	Unallocated.	-
000	001	VMOV (between general-purpose register and half-precision)	FEAT_FP16
000	010	VMOV (between general-purpose register and single-precision)	-
001	010	Unallocated.	-
01x	010	Unallocated.	-
10x	010	Unallocated.	-
110	010	Unallocated.	-
111	010	Floating-point move special register	-
xxx	011	Advanced SIMD 8/16/32-bit element move/duplicate	-
xxx	10x	Unallocated.	-
xxx	11x	System register 32-bit move	-

F4.1.16.1 Floating-point move special register

This section describes the encoding of the Floating-point move special register instruction class. The encodings in this section are decoded from Advanced SIMD and System register 32-bit move.

31 28	27 26 25 24	23 22 21 20	19 16	15 12	11 10 9 8	7	6	5	4	3	2	1	0
!=1111	1 1 1 0	1 1 1 L	reg	Rt	1 0 1 0	(0)	(0)	(0)	1	(0)	(0)	(0)	(0)
cond													

Table F4-62

Decode fields	Instruction page
L	
0	VMSR
1	VMRS

F4.1.16.2 Advanced SIMD 8/16/32-bit element move/duplicate

This section describes the encoding of the Advanced SIMD 8/16/32-bit element move/duplicate instruction class. The encodings in this section are decoded from Advanced SIMD and System register 32-bit move.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

31–28	27 26 25 24	23–21	20	19–16	15–12	11 10 9 8	7	6 5	4	3	2	1	0
!=1111 (cond)	1 1 1 0	opc1	L	Vn	Rt	1 0 1 1	N	opc2	1	(0)	(0)	(0)	(0)

Table F4-63

Decode fields			Instruction page
opc1	**L**	**opc2**	
`0xx`	`0`	`xx`	VMOV (general-purpose register to scalar)
`xxx`	`1`	`xx`	VMOV (scalar to general-purpose register)
`1xx`	`0`	`0x`	VDUP (general-purpose register)
`1xx`	`0`	`1x`	Unallocated.

F4.1.16.3 System register 32-bit move

This section describes the encoding of the System register 32-bit move instruction class. The encodings in this section are decoded from Advanced SIMD and System register 32-bit move.

31–28	27 26 25 24	23–21	20	19–16	15–12	11 10 9	8	7–5	4	3–0
!=1111 (cond)	1 1 1 0	opc1	L	CRn	Rt	1 1 1		opc2	1	CRm

cp15

Table F4-64

Decode fields	Instruction page
L	
`0`	MCR
`1`	MRC

F4.1.17 Floating-point data-processing

This section describes the encoding of the Floating-point data-processing group. The encodings in this section are decoded from System register access, Advanced SIMD, floating-point, and Supervisor call.

This group has encodings in both the T32 and A32 instruction sets. For information about mappings between the encodings of this group, see About the A32 Advanced SIMD and floating-point instructions and their encoding

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

31–28	27–24	23–20	19–12	11 10	9–7	6	5	4	3–0
!=1111	1110	op0		10				0	

op1

Table F4-65 Encoding table for the Floating-point data-processing group

Decode fields		Decode group or instruction page
op0	op1	
1x11	1	Floating-point data-processing (two registers)
1x11	0	Floating-point move immediate
!= 1x11	x	Floating-point data-processing (three registers)

F4.1.17.1 Floating-point data-processing (two registers)

This section describes the encoding of the Floating-point data-processing (two registers) instruction class. The encodings in this section are decoded from Floating-point data-processing.

31–28	27 26 25 24	23	22	21 20	19	18–16	15–12	11 10	9 8	7	6	5	4	3–0
!=1111	1 1 1 0	1	D	1 1	o1	opc2	Vd	1 0	size	o3	1	M	0	Vm
cond														

Table F4-66

Decode fields				Instruction page	Feature
o1	opc2	size	o3		
x	xxx	00	x	Unallocated.	-
0	000	01	0	Unallocated.	-
0	000	01	1	VABS - Half-precision scalar variant	FEAT_FP16
0	000	10	0	VMOV (register) - Single-precision scalar variant	-
0	000	10	1	VABS - Single-precision scalar variant	-
0	000	11	0	VMOV (register) - Double-precision scalar variant	-
0	000	11	1	VABS - Double-precision scalar variant	-
0	001	01	0	VNEG - Half-precision scalar variant	FEAT_FP16
0	001	01	1	VSQRT - Half-precision scalar variant	FEAT_FP16
0	001	10	0	VNEG - Single-precision scalar variant	-

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields				Instruction page	Feature
o1	opc2	size	o3		
0	001	10	1	VSQRT - Single-precision scalar variant	-
0	001	11	0	VNEG - Double-precision scalar variant	-
0	001	11	1	VSQRT - Double-precision scalar variant	-
0	010	01	x	Unallocated.	-
0	010	10	0	VCVTB - Half-precision to single-precision variant	-
0	010	10	1	VCVTT - Half-precision to single-precision variant	-
0	010	11	0	VCVTB - Half-precision to double-precision variant	-
0	010	11	1	VCVTT - Half-precision to double-precision variant	-
0	011	01	0	VCVTB (BFloat16)	FEAT_AA32BF16
0	011	01	1	VCVTT (BFloat16)	FEAT_AA32BF16
0	011	10	0	VCVTB - Single-precision to half-precision variant	-
0	011	10	1	VCVTT - Single-precision to half-precision variant	-
0	011	11	0	VCVTB - Double-precision to half-precision variant	-
0	011	11	1	VCVTT - Double-precision to half-precision variant	-
0	100	01	0	VCMP	FEAT_FP16
0	100	01	1	VCMPE	FEAT_FP16
0	100	10	0	VCMP	-
0	100	10	1	VCMPE	-
0	100	11	0	VCMP	-
0	100	11	1	VCMPE	-
0	101	01	0	VCMP	FEAT_FP16
0	101	01	1	VCMPE	FEAT_FP16
0	101	10	0	VCMP	-
0	101	10	1	VCMPE	-
0	101	11	0	VCMP	-
0	101	11	1	VCMPE	-
0	110	01	0	VRINTR - Half-precision scalar variant	FEAT_FP16
0	110	01	1	VRINTZ (floating-point) - Half-precision scalar variant	FEAT_FP16
0	110	10	0	VRINTR - Single-precision scalar variant	-
0	110	10	1	VRINTZ (floating-point) - Single-precision scalar variant	-
0	110	11	0	VRINTR - Double-precision scalar variant	-
0	110	11	1	VRINTZ (floating-point) - Double-precision scalar variant	-
0	111	01	0	VRINTX (floating-point) - Half-precision scalar variant	FEAT_FP16
0	111	01	1	Unallocated.	-
0	111	10	0	VRINTX (floating-point) - Single-precision scalar variant	-
0	111	10	1	VCVT (between double-precision and single-precision) - Single-precision to double-precision variant	-
0	111	11	0	VRINTX (floating-point) - Double-precision scalar variant	-
0	111	11	1	VCVT (between double-precision and single-precision) - Double-precision to single-precision variant	-

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields				Instruction page	Feature
o1	opc2	size	o3		
1	000	01	x	VCVT (integer to floating-point, floating-point) - Half-precision scalar variant	FEAT_FP16
1	000	10	x	VCVT (integer to floating-point, floating-point) - Single-precision scalar variant	-
1	000	11	x	VCVT (integer to floating-point, floating-point) - Double-precision scalar variant	-
1	001	01	x	Unallocated.	-
1	001	10	x	Unallocated.	-
1	001	11	0	Unallocated.	-
1	001	11	1	VJCVT	FEAT_JSCVT
1	01x	01	x	VCVT (between floating-point and fixed-point, floating-point)	FEAT_FP16
1	01x	10	x	VCVT (between floating-point and fixed-point, floating-point)	-
1	01x	11	x	VCVT (between floating-point and fixed-point, floating-point)	-
1	100	01	0	VCVTR	FEAT_FP16
1	100	01	1	VCVT (floating-point to integer, floating-point)	FEAT_FP16
1	100	10	0	VCVTR	-
1	100	10	1	VCVT (floating-point to integer, floating-point)	-
1	100	11	0	VCVTR	-
1	100	11	1	VCVT (floating-point to integer, floating-point)	-
1	101	01	0	VCVTR	FEAT_FP16
1	101	01	1	VCVT (floating-point to integer, floating-point)	FEAT_FP16
1	101	10	0	VCVTR	-
1	101	10	1	VCVT (floating-point to integer, floating-point)	-
1	101	11	0	VCVTR	-
1	101	11	1	VCVT (floating-point to integer, floating-point)	-
1	11x	01	x	VCVT (between floating-point and fixed-point, floating-point)	FEAT_FP16
1	11x	10	x	VCVT (between floating-point and fixed-point, floating-point)	-
1	11x	11	x	VCVT (between floating-point and fixed-point, floating-point)	-

F4.1.17.2 Floating-point move immediate

This section describes the encoding of the Floating-point move immediate instruction class. The encodings in this section are decoded from Floating-point data-processing.

31–28	27	26	25	24	23	22	21	20	19–16	15–12	11	10	9	8	7	6	5	4	3–0
!=1111	1	1	1	0	1	D	1	1	imm4H	Vd	1	0	size		(0)	0	(0)	0	imm4L
cond																			

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F4-67

Decode fields size	Instruction page	Feature
00	Unallocated.	-
01	VMOV (immediate) - Half-precision scalar variant	FEAT_FP16
10	VMOV (immediate) - Single-precision scalar variant	-
11	VMOV (immediate) - Double-precision scalar variant	-

F4.1.17.3 Floating-point data-processing (three registers)

This section describes the encoding of the Floating-point data-processing (three registers) instruction class. The encodings in this section are decoded from Floating-point data-processing.

31 28	27 26 25 24	23	22	21 20	19 16	15 12	11 10	9 8	7	6	5	4	3 0
!=1111	1 1 1 0	o0	D	o1	Vn	Vd	1 0	size	N	o2	M	0	Vm
cond													

Table F4-68

Decode fields o0:o1	size	o2	Instruction page	Feature
!= 111	00	x	Unallocated.	-
000	01	0	VMLA (floating-point) - Half-precision scalar variant	FEAT_FP16
000	01	1	VMLS (floating-point) - Half-precision scalar variant	FEAT_FP16
000	10	0	VMLA (floating-point) - Single-precision scalar variant	-
000	10	1	VMLS (floating-point) - Single-precision scalar variant	-
000	11	0	VMLA (floating-point) - Double-precision scalar variant	-
000	11	1	VMLS (floating-point) - Double-precision scalar variant	-
001	01	0	VNMLS - Half-precision scalar variant	FEAT_FP16
001	01	1	VNMLA - Half-precision scalar variant	FEAT_FP16
001	10	0	VNMLS - Single-precision scalar variant	-
001	10	1	VNMLA - Single-precision scalar variant	-
001	11	0	VNMLS - Double-precision scalar variant	-
001	11	1	VNMLA - Double-precision scalar variant	-
010	01	0	VMUL (floating-point) - Half-precision scalar variant	FEAT_FP16
010	01	1	VNMUL - Half-precision scalar variant	FEAT_FP16
010	10	0	VMUL (floating-point) - Single-precision scalar variant	-
010	10	1	VNMUL - Single-precision scalar variant	-
010	11	0	VMUL (floating-point) - Double-precision scalar variant	-

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields			Instruction page	Feature
o0:o1	size	o2		
010	11	1	VNMUL - Double-precision scalar variant	-
011	01	0	VADD (floating-point) - Half-precision scalar variant	FEAT_FP16
011	01	1	VSUB (floating-point) - Half-precision scalar variant	FEAT_FP16
011	10	0	VADD (floating-point) - Single-precision scalar variant	-
011	10	1	VSUB (floating-point) - Single-precision scalar variant	-
011	11	0	VADD (floating-point) - Double-precision scalar variant	-
011	11	1	VSUB (floating-point) - Double-precision scalar variant	-
100	01	0	VDIV - Half-precision scalar variant	FEAT_FP16
100	10	0	VDIV - Single-precision scalar variant	-
100	11	0	VDIV - Double-precision scalar variant	-
101	01	0	VFNMS - Half-precision scalar variant	FEAT_FP16
101	01	1	VFNMA - Half-precision scalar variant	FEAT_FP16
101	10	0	VFNMS - Single-precision scalar variant	-
101	10	1	VFNMA - Single-precision scalar variant	-
101	11	0	VFNMS - Double-precision scalar variant	-
101	11	1	VFNMA - Double-precision scalar variant	-
110	01	0	VFMA - Half-precision scalar variant	FEAT_FP16
110	01	1	VFMS - Half-precision scalar variant	FEAT_FP16
110	10	0	VFMA - Single-precision scalar variant	-
110	10	1	VFMS - Single-precision scalar variant	-
110	11	0	VFMA - Double-precision scalar variant	-
110	11	1	VFMS - Double-precision scalar variant	-

F4.1.18 Unconditional instructions

This section describes the encoding of the Unconditional instructions group. The encodings in this section are decoded from A32 instruction set encoding.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F4-69 Encoding table for the Unconditional instructions group

Decode fields		Decode group or instruction page
op0	op1	
00x	x	Miscellaneous
01x	x	Advanced SIMD data-processing
1xx	1	Memory hints and barriers
100	0	Advanced SIMD element or structure load/store
101	0	Unallocated.
11x	0	Unallocated.

F4.1.19 Miscellaneous

This section describes the encoding of the Miscellaneous group. The encodings in this section are decoded from Unconditional instructions.

Table F4-70 Encoding table for the Miscellaneous group

Decode fields		Decode group or instruction page	Feature
op0	op1		
0xxxx	xxxx	Unallocated.	-
10000	xx0x	Change Process State	-
10001	1000	Unallocated.	-
10001	x100	Unallocated.	-
10001	xx01	Unallocated.	-
10001	0000	SETPAN	FEAT_PAN
1000x	0111	Unallocated.	-
10010	0111	CONSTRAINED UNPREDICTABLE	-
10011	0111	Unallocated.	-
1001x	xx0x	Unallocated.	-
100xx	0011	Unallocated.	-
100xx	0x10	Unallocated.	-
100xx	1x1x	Unallocated.	-
101xx	xxxx	Unallocated.	-

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

11xxx	xxxx	Unallocated.	-

The behavior of the CONSTRAINED UNPREDICTABLE encodings in this table is described in CONSTRAINED UNPREDICTABLE behavior for T32 and A32 instruction encodings.

F4.1.19.1 Change Process State

This section describes the encoding of the Change Process State instruction class. The encodings in this section are decoded from Miscellaneous.

31	30	29	28	27	26	25	24	23	22	21	20	19 18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	0
1	1	1	1	0	0	0	1	0	0	0	0	imod	M	op	(0)	(0)	(0)	(0)	(0)	(0)	E	A	I	F	0	mode	

Table F4-71

Decode fields						Instruction page
imod	**M**	**op**	**I**	**F**	**mode**	
xx	x	1	0	0	0xxxx	SETEND
00	1	0	x	x	xxxxx	CPS, CPSID, CPSIE - Change mode variant
10	x	0	x	x	xxxxx	CPS, CPSID, CPSIE - Interrupt enable and change mode variant
xx	x	1	0	0	1xxxx	Unallocated.
xx	x	1	0	1	xxxxx	Unallocated.
xx	x	1	1	x	xxxxx	Unallocated.
11	x	0	x	x	xxxxx	CPS, CPSID, CPSIE - Interrupt disable and change mode variant

F4.1.20 Advanced SIMD data-processing

This section describes the encoding of the Advanced SIMD data-processing group. The encodings in this section are decoded from Unconditional instructions.

This group has encodings in both the T32 and A32 instruction sets. For information about mappings between the encodings of this group, see About the A32 Advanced SIMD and floating-point instructions and their encoding

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F4-72 Encoding table for the Advanced SIMD data-processing group

Decode fields		Decode group or instruction page
op0	op1	
0	x	Advanced SIMD three registers of the same length
1	0	Advanced SIMD two registers, or three registers of different lengths
1	1	Advanced SIMD shifts and immediate generation

F4.1.20.1 Advanced SIMD three registers of the same length

This section describes the encoding of the Advanced SIMD three registers of the same length instruction class. The encodings in this section are decoded from Advanced SIMD data-processing.

31 30 29 28	27 26 25	24	23	22	21 20	19 16	15 12	11 8	7	6	5	4	3 0
1 1 1 1	0 0 1	U	0	D	size	Vn	Vd	opc	N	Q	M	o1	Vm

Table F4-73

Decode fields					Instruction page	Feature
U	size	opc	Q	o1		
0	0x	1100	x	1	VFMA	-
0	0x	1101	x	0	VADD (floating-point)	-
0	0x	1101	x	1	VMLA (floating-point)	-
0	0x	1110	x	0	VCEQ (register) - A2	-
0	0x	1111	x	0	VMAX (floating-point)	-
0	0x	1111	x	1	VRECPS	-
x	xx	0000	x	0	VHADD	-
0	00	0001	x	1	VAND (register)	-
x	xx	0000	x	1	VQADD	-
x	xx	0001	x	0	VRHADD	-
0	00	1100	x	0	SHA1C	FEAT_SHA1
x	xx	0010	x	0	VHSUB	-
0	01	0001	x	1	VBIC (register)	-
x	xx	0010	x	1	VQSUB	-
x	xx	0011	x	0	VCGT (register) - A1	-
x	xx	0011	x	1	VCGE (register) - A1	-
0	01	1100	x	0	SHA1P	FEAT_SHA1
0	1x	1100	x	1	VFMS	-
0	1x	1101	x	0	VSUB (floating-point)	-

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields					Instruction page	Feature
U	size	opc	Q	o1		
0	1x	1101	x	1	VMLS (floating-point)	-
0	1x	1110	x	0	Unallocated.	-
0	1x	1111	x	0	VMIN (floating-point)	-
0	1x	1111	x	1	VRSQRTS	-
x	xx	0100	x	0	VSHL (register)	-
0	xx	1000	x	0	VADD (integer)	-
0	10	0001	x	1	VORR (register)	-
0	xx	1000	x	1	VTST	-
x	xx	0100	x	1	VQSHL (register)	-
0	xx	1001	x	0	VMLA (integer)	-
x	xx	0101	x	0	VRSHL	-
x	xx	0101	x	1	VQRSHL	-
0	xx	1011	x	0	VQDMULH	-
0	10	1100	x	0	SHA1M	FEAT_SHA1
0	xx	1011	x	1	VPADD (integer)	-
x	xx	0110	x	0	VMAX (integer)	-
0	11	0001	x	1	VORN (register)	-
x	xx	0110	x	1	VMIN (integer)	-
x	xx	0111	x	0	VABD (integer)	-
x	xx	0111	x	1	VABA	-
0	11	1100	x	0	SHA1SU0	FEAT_SHA1
1	0x	1101	x	0	VPADD (floating-point)	-
1	0x	1101	x	1	VMUL (floating-point)	-
1	0x	1110	x	0	VCGE (register) - A2	-
1	0x	1110	x	1	VACGE	-
1	0x	1111	0	0	VPMAX (floating-point)	-
1	0x	1111	x	1	VMAXNM	-
1	00	0001	x	1	VEOR	-
x	xx	1001	x	1	VMUL (integer and polynomial)	-
1	00	1100	x	0	SHA256H	FEAT_SHA256
x	xx	1010	0	0	VPMAX (integer)	-
1	01	0001	x	1	VBSL	-
x	xx	1010	0	1	VPMIN (integer)	-
x	xx	1010	1	x	Unallocated.	-
1	01	1100	x	0	SHA256H2	FEAT_SHA256
1	1x	1101	x	0	VABD (floating-point)	-
1	1x	1110	x	0	VCGT (register) - A2	-
1	1x	1110	x	1	VACGT	-
1	1x	1111	0	0	VPMIN (floating-point)	-

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields					Instruction page	Feature
U	size	opc	Q	o1		
1	1x	1111	x	1	VMINNM	-
1	xx	1000	x	0	VSUB (integer)	-
1	10	0001	x	1	VBIT	-
1	xx	1000	x	1	VCEQ (register) - A1	-
1	xx	1001	x	0	VMLS (integer)	-
1	xx	1011	x	0	VQRDMULH	-
1	10	1100	x	0	SHA256SU1	FEAT_SHA256
1	xx	1011	x	1	VQRDMLAH	FEAT_RDM
1	11	0001	x	1	VBIF	-
1	xx	1100	x	1	VQRDMLSH	FEAT_RDM
1	xx	1111	1	0	Unallocated.	-

F4.1.21 Advanced SIMD two registers, or three registers of different lengths

This section describes the encoding of the Advanced SIMD two registers, or three registers of different lengths group. The encodings in this section are decoded from Advanced SIMD data-processing.

Table F4-74 Encoding table for the Advanced SIMD two registers, or three registers of different lengths group

Decode fields				Decode group or instruction page
op0	op1	op2	op3	
0	11	xx	x	VEXT (byte elements)
1	11	0x	x	Advanced SIMD two registers misc
1	11	10	x	VTBL, VTBX
1	11	11	x	Advanced SIMD duplicate (scalar)
x	!= 11	xx	0	Advanced SIMD three registers of different lengths
x	!= 11	xx	1	Advanced SIMD two registers and a scalar

F4.1.21.1 Advanced SIMD two registers misc

This section describes the encoding of the Advanced SIMD two registers misc instruction class. The encodings in this section are decoded from Advanced SIMD two registers, or three registers of different lengths.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 12	11 10	7 6 5 4	3 0
1 1 1 1	0 0 1 1 1	D 1 1	size	opc1	Vd	0	opc2	Q M 0	Vm

Table F4-75

Decode fields				Instruction page	Feature
size	opc1	opc2	Q		
xx	00	0000	x	VREV64	-
xx	00	0001	x	VREV32	-
xx	00	0010	x	VREV16	-
xx	00	0011	x	Unallocated.	-
xx	00	010x	x	VPADDL	-
xx	00	0110	0	AESE	FEAT_AES
xx	00	0110	1	AESD	FEAT_AES
xx	00	0111	0	AESMC	FEAT_AES
xx	00	0111	1	AESIMC	FEAT_AES
xx	00	1000	x	VCLS	-
00	10	0000	x	VSWP	-
xx	00	1001	x	VCLZ	-
xx	00	1010	x	VCNT	-
xx	00	1011	x	VMVN (register)	-
00	10	1100	1	Unallocated.	-
xx	00	110x	x	VPADAL	-
xx	00	1110	x	VQABS	-
xx	00	1111	x	VQNEG	-
xx	01	x000	x	VCGT (immediate #0)	-
xx	01	x001	x	VCGE (immediate #0)	-
xx	01	x010	x	VCEQ (immediate #0)	-
xx	01	x011	x	VCLE (immediate #0)	-
xx	01	x100	x	VCLT (immediate #0)	-
xx	01	x110	x	VABS	-
xx	01	x111	x	VNEG	-
xx	01	0101	1	SHA1H	FEAT_SHA1
01	10	1100	1	VCVT (from single-precision to BFloat16, Advanced SIMD)	FEAT_AA32BF16
xx	10	0001	x	VTRN	-
xx	10	0010	x	VUZP	-
xx	10	0011	x	VZIP	-

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields				Instruction page	Feature
size	opc1	opc2	Q		
xx	10	0100	0	VMOVN	-
xx	10	0100	1	VQMOVN, VQMOVUN - Unsigned result variant	-
xx	10	0101	x	VQMOVN, VQMOVUN - Signed result variant	-
xx	10	0110	0	VSHLL	-
xx	10	0111	0	SHA1SU1	FEAT_SHA1
xx	10	0111	1	SHA256SU0	FEAT_SHA256
xx	10	1000	x	VRINTN (Advanced SIMD)	-
xx	10	1001	x	VRINTX (Advanced SIMD)	-
xx	10	1010	x	VRINTA (Advanced SIMD)	-
xx	10	1011	x	VRINTZ (Advanced SIMD)	-
10	10	1100	1	Unallocated.	-
xx	10	1100	0	VCVT (between half-precision and single-precision, Advanced SIMD) - Single-precision to half-precision variant	-
xx	10	1101	x	VRINTM (Advanced SIMD)	-
xx	10	1110	0	VCVT (between half-precision and single-precision, Advanced SIMD) - Half-precision to single-precision variant	-
xx	10	1110	1	Unallocated.	-
xx	10	1111	x	VRINTP (Advanced SIMD)	-
xx	11	000x	x	VCVTA (Advanced SIMD)	-
xx	11	001x	x	VCVTN (Advanced SIMD)	-
xx	11	010x	x	VCVTP (Advanced SIMD)	-
xx	11	011x	x	VCVTM (Advanced SIMD)	-
xx	11	10x0	x	VRECPE	-
xx	11	10x1	x	VRSQRTE	-
11	10	1100	1	Unallocated.	-
xx	11	11xx	x	VCVT (between floating-point and integer, Advanced SIMD)	-

F4.1.21.2 Advanced SIMD duplicate (scalar)

This section describes the encoding of the Advanced SIMD duplicate (scalar) instruction class. The encodings in this section are decoded from Advanced SIMD two registers, or three registers of different lengths.

31 30 29 28	27 26 25 24	23 22	21 20	19 16	15 12	11 10	9 7	6	5	4	3 0
1 1 1 1	0 0 1 1 1	D	1 1	imm4	Vd	1 1	opc	Q	M	0	Vm

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F4-76

Decode fields opc	Instruction page
000	VDUP (scalar)
001	Unallocated.
01x	Unallocated.
1xx	Unallocated.

F4.1.21.3 Advanced SIMD three registers of different lengths

This section describes the encoding of the Advanced SIMD three registers of different lengths instruction class. The encodings in this section are decoded from Advanced SIMD two registers, or three registers of different lengths.

31 30 29 28	27 26 25	24	23	22	21 20	19 16	15 12	11 8	7	6	5	4	3 0
1 1 1 1	0 0 1	U	1	D	!=11 size	Vn	Vd	opc	N	0	M	0	Vm

Table F4-77

Decode fields U	 opc	Instruction page
x	0000	VADDL
x	0001	VADDW
x	0010	VSUBL
0	0100	VADDHN
x	0011	VSUBW
0	0110	VSUBHN
0	1001	VQDMLAL
x	0101	VABAL
0	1011	VQDMLSL
0	1101	VQDMULL
x	0111	VABDL (integer)
x	1000	VMLAL (integer)
x	1010	VMLSL (integer)
1	0100	VRADDHN
1	0110	VRSUBHN
x	11x0	VMULL (integer and polynomial)
1	1001	Unallocated.
1	1011	Unallocated.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields		Instruction page
U	opc	
1	1101	Unallocated.
x	1111	Unallocated.

F4.1.21.4 Advanced SIMD two registers and a scalar

This section describes the encoding of the Advanced SIMD two registers and a scalar instruction class. The encodings in this section are decoded from Advanced SIMD two registers, or three registers of different lengths.

31	30	29	28	27	26	25	24	23	22	21	20	19 16	15 12	11 8	7	6	5	4	3 0
1	1	1	1	0	0	1	Q	1	D	!=11		Vn	Vd	opc	N	1	M	0	Vm
										size									

Table F4-78

Decode fields		Instruction page	Feature
Q	opc		
x	000x	VMLA (by scalar)	-
0	0011	VQDMLAL	-
x	0010	VMLAL (by scalar)	-
0	0111	VQDMLSL	-
x	010x	VMLS (by scalar)	-
0	1011	VQDMULL	-
x	0110	VMLSL (by scalar)	-
x	100x	VMUL (by scalar)	-
1	0011	Unallocated.	-
x	1010	VMULL (by scalar)	-
1	0111	Unallocated.	-
x	1100	VQDMULH	-
x	1101	VQRDMULH	-
1	1011	Unallocated.	-
x	1110	VQRDMLAH	FEAT_RDM
x	1111	VQRDMLSH	FEAT_RDM

F4.1.22 Advanced SIMD shifts and immediate generation

This section describes the encoding of the Advanced SIMD shifts and immediate generation group. The encodings in this section are decoded from Advanced SIMD data-processing.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

31 … 25	24	23	22	21 … 7	6	5	4	3 … 0
1111001		1		op0			1	

Table F4-79 Encoding table for the Advanced SIMD shifts and immediate generation group

Decode fields op0	Decode group or instruction page
`000xxxxxxxxxxx0`	Advanced SIMD one register and modified immediate
`!= 000xxxxxxxxxxx0`	Advanced SIMD two registers and shift amount

F4.1.22.1 Advanced SIMD one register and modified immediate

This section describes the encoding of the Advanced SIMD one register and modified immediate instruction class. The encodings in this section are decoded from Advanced SIMD shifts and immediate generation.

31 30 29 28	27 26 25	24	23	22	21 20 19	18 … 16	15 … 12	11 … 8	7	6	5	4	3 … 0
1 1 1 1	0 0 1	i	1	D	0 0 0	imm3	Vd	cmode	0	Q	op	1	imm4

Table F4-80

Decode fields cmode	op	Instruction page
`0xx0`	`0`	VMOV (immediate) - A1
`0xx0`	`1`	VMVN (immediate) - A1
`0xx1`	`0`	VORR (immediate) - A1
`0xx1`	`1`	VBIC (immediate) - A1
`10x0`	`0`	VMOV (immediate) - A3
`10x0`	`1`	VMVN (immediate) - A2
`10x1`	`0`	VORR (immediate) - A2
`10x1`	`1`	VBIC (immediate) - A2
`11xx`	`0`	VMOV (immediate) - A4
`110x`	`1`	VMVN (immediate) - A3
`1110`	`1`	VMOV (immediate) - A5
`1111`	`1`	Unallocated.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F4.1.22.2 Advanced SIMD two registers and shift amount

This section describes the encoding of the Advanced SIMD two registers and shift amount instruction class. The encodings in this section are decoded from Advanced SIMD shifts and immediate generation.

31 30 29 28	27 26 25 24	23	22	21	19 18 16	15 12	11 8	7	6	5	4	3 0
1 1 1 1	0 0 1 U	1	D	imm3H	imm3L	Vd	opc	L	Q	M	1	Vm

Table F4-81

Decode fields						Instruction page
U	**imm3H:L**	**imm3L**	**opc**	**L**	**Q**	
x	!= 0000	xxx	0000	x	x	VSHR
x	!= 0000	xxx	0001	x	x	VSRA
x	!= 0000	000	1010	x	0	VMOVL
x	!= 0000	xxx	0010	x	x	VRSHR
x	!= 0000	xxx	0011	x	x	VRSRA
x	!= 0000	xxx	0111	x	x	VQSHL, VQSHLU (immediate) - 128-bit SIMD vector, signed result variant
x	!= 0000	xxx	1001	0	0	VQSHRN, VQSHRUN - Signed result variant
x	!= 0000	xxx	1001	0	1	VQRSHRN, VQRSHRUN - Signed result variant
x	!= 0000	xxx	1010	0	0	VSHLL
x	!= 0000	xxx	11xx	0	x	VCVT (between floating-point and fixed-point, Advanced SIMD)
0	!= 0000	xxx	0101	x	x	VSHL (immediate)
0	!= 0000	xxx	1000	0	0	VSHRN
0	!= 0000	xxx	1000	0	1	VRSHRN
1	!= 0000	xxx	0100	x	x	VSRI
1	!= 0000	xxx	0101	x	x	VSLI
1	!= 0000	xxx	0110	x	x	VQSHL, VQSHLU (immediate) - 128-bit SIMD vector, unsigned result variant
1	!= 0000	xxx	1000	0	0	VQSHRN, VQSHRUN - Unsigned result variant
1	!= 0000	xxx	1000	0	1	VQRSHRN, VQRSHRUN - Unsigned result variant

F4.1.23 Memory hints and barriers

This section describes the encoding of the Memory hints and barriers group. The encodings in this section are decoded from Unconditional instructions.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F4-82 Encoding table for the Memory hints and barriers group

Decode fields		Decode group or instruction page
op0	**op1**	
`00xx1`	`x`	CONSTRAINED UNPREDICTABLE
`01001`	`x`	CONSTRAINED UNPREDICTABLE
`01011`	`x`	Barriers
`011x1`	`x`	CONSTRAINED UNPREDICTABLE
`0xxx0`	`x`	Preload (immediate)
`1xxx0`	`0`	Preload (register)
`1xxx1`	`0`	CONSTRAINED UNPREDICTABLE
`1xxxx`	`1`	Unallocated.

The behavior of the CONSTRAINED UNPREDICTABLE encodings in this table is described in CONSTRAINED UNPREDICTABLE behavior for T32 and A32 instruction encodings.

F4.1.23.1 Barriers

This section describes the encoding of the Barriers instruction class. The encodings in this section are decoded from Memory hints and barriers.

Table F4-83

Decode fields		Instruction page	Feature
opcode	**option**		
`0000`	`xxxx`	CONSTRAINED UNPREDICTABLE	-
`0001`	`xxxx`	CLREX	-
`001x`	`xxxx`	CONSTRAINED UNPREDICTABLE	-

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields		Instruction page	Feature
opcode	option		
0100	!= 0x00	DSB	-
0100	0000	SSBB	-
0100	0100	PSSBB	-
0101	xxxx	DMB	-
0110	xxxx	ISB	-
0111	xxxx	SB	FEAT_SB
1xxx	xxxx	CONSTRAINED UNPREDICTABLE	-

The behavior of the CONSTRAINED UNPREDICTABLE encodings in this table is described in CONSTRAINED UNPREDICTABLE behavior for T32 and A32 instruction encodings.

F4.1.23.2 Preload (immediate)

This section describes the encoding of the Preload (immediate) instruction class. The encodings in this section are decoded from Memory hints and barriers.

31	30	29	28	27	26	25	24	23	22	21	20	19 … 16	15	14	13	12	11 … 0
1	1	1	1	0	1	0	D	U	R	0	1	Rn	(1)	(1)	(1)	(1)	imm12

Table F4-84

Decode fields			Instruction page
D	R	Rn	
0	0	xxxx	Reserved hint, behaves as NOP.
0	1	xxxx	PLI (immediate, literal)
1	x	1111	PLD (literal)
1	0	!= 1111	PLD, PLDW (immediate) - Preload write variant
1	1	!= 1111	PLD, PLDW (immediate) - Preload read variant

F4.1.23.3 Preload (register)

This section describes the encoding of the Preload (register) instruction class. The encodings in this section are decoded from Memory hints and barriers.

31	30	29	28	27	26	25	24	23	22	21	20	19 … 16	15	14	13	12	11 … 7	6	5	4	3 … 0
1	1	1	1	0	1	1	D	U	o2	0	1	Rn	(1)	(1)	(1)	(1)	imm5	stype		0	Rm

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Table F4-85

Decode fields			Instruction page
D	**o2**	**imm5:stype**	
0	0	xxxxxxx	Reserved hint, behaves as NOP.
0	1	!= 0000011	PLI (register) - Shift or rotate by value variant
0	1	0000011	PLI (register) - Rotate right with extend variant
1	0	!= 0000011	PLD, PLDW (register) - Preload write, optional shift or rotate variant
1	0	0000011	PLD, PLDW (register) - Preload write, rotate right with extend variant
1	1	!= 0000011	PLD, PLDW (register) - Preload read, optional shift or rotate variant
1	1	0000011	PLD, PLDW (register) - Preload read, rotate right with extend variant

F4.1.24 Advanced SIMD element or structure load/store

This section describes the encoding of the Advanced SIMD element or structure load/store group. The encodings in this section are decoded from Unconditional instructions.

This group has encodings in both the T32 and A32 instruction sets. For information about mappings between the encodings of this group, see About the A32 Advanced SIMD and floating-point instructions and their encoding

Table F4-86 Encoding table for the Advanced SIMD element or structure load/store group

Decode fields		Decode group or instruction page
op0	**op1**	
0	xx	Advanced SIMD load/store multiple structures
1	11	Advanced SIMD load single structure to all lanes
1	!= 11	Advanced SIMD load/store single structure to one lane

F4.1.24.1 Advanced SIMD load/store multiple structures

This section describes the encoding of the Advanced SIMD load/store multiple structures instruction class. The encodings in this section are decoded from Advanced SIMD element or structure load/store.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

31 30 29 28	27 26 25 24	23 22 21 20	19 16	15 12	11 8	7 6	5 4	3 0
1 1 1 1	0 1 0 0	0 D L 0	Rn	Vd	itype	size	align	Rm

Table F4-87

Decode fields			Instruction page
L	**itype**	**Rm**	
0	000x	!= 11x1	VST4 (multiple 4-element structures)
0	000x	1101	VST4 (multiple 4-element structures)
0	000x	1111	VST4 (multiple 4-element structures)
0	0010	!= 11x1	VST1 (multiple single elements)
0	0010	1101	VST1 (multiple single elements)
0	0010	1111	VST1 (multiple single elements)
0	0011	!= 11x1	VST2 (multiple 2-element structures)
0	0011	1101	VST2 (multiple 2-element structures)
0	0011	1111	VST2 (multiple 2-element structures)
0	010x	!= 11x1	VST3 (multiple 3-element structures)
0	010x	1101	VST3 (multiple 3-element structures)
0	010x	1111	VST3 (multiple 3-element structures)
0	0110	!= 11x1	VST1 (multiple single elements)
0	0110	1101	VST1 (multiple single elements)
0	0110	1111	VST1 (multiple single elements)
0	0111	!= 11x1	VST1 (multiple single elements)
0	0111	1101	VST1 (multiple single elements)
0	0111	1111	VST1 (multiple single elements)
0	100x	!= 11x1	VST2 (multiple 2-element structures)
0	100x	1101	VST2 (multiple 2-element structures)
0	100x	1111	VST2 (multiple 2-element structures)
0	1010	!= 11x1	VST1 (multiple single elements)
0	1010	1101	VST1 (multiple single elements)
0	1010	1111	VST1 (multiple single elements)
1	000x	!= 11x1	VLD4 (multiple 4-element structures)
1	000x	1101	VLD4 (multiple 4-element structures)
1	000x	1111	VLD4 (multiple 4-element structures)
1	0010	!= 11x1	VLD1 (multiple single elements)
1	0010	1101	VLD1 (multiple single elements)
1	0010	1111	VLD1 (multiple single elements)

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields			Instruction page
L	itype	Rm	
1	0011	!= 11x1	VLD2 (multiple 2-element structures)
1	0011	1101	VLD2 (multiple 2-element structures)
1	0011	1111	VLD2 (multiple 2-element structures)
1	010x	!= 11x1	VLD3 (multiple 3-element structures)
1	010x	1101	VLD3 (multiple 3-element structures)
1	010x	1111	VLD3 (multiple 3-element structures)
x	1011	xxxx	Unallocated.
1	0110	!= 11x1	VLD1 (multiple single elements)
1	0110	1101	VLD1 (multiple single elements)
1	0110	1111	VLD1 (multiple single elements)
1	0111	!= 11x1	VLD1 (multiple single elements)
1	0111	1101	VLD1 (multiple single elements)
1	0111	1111	VLD1 (multiple single elements)
x	11xx	xxxx	Unallocated.
1	100x	!= 11x1	VLD2 (multiple 2-element structures)
1	100x	1101	VLD2 (multiple 2-element structures)
1	100x	1111	VLD2 (multiple 2-element structures)
1	1010	!= 11x1	VLD1 (multiple single elements)
1	1010	1101	VLD1 (multiple single elements)
1	1010	1111	VLD1 (multiple single elements)

F4.1.24.2 Advanced SIMD load single structure to all lanes

This section describes the encoding of the Advanced SIMD load single structure to all lanes instruction class. The encodings in this section are decoded from Advanced SIMD element or structure load/store.

31 30 29 28	27 26 25 24	23	22	21	20	19 16	15 12	11 10	9 8	7 6	5	4	3 0
1 1 1 1	0 1 0 0	1	D	L	0	Rn	Vd	1 1	N	size	T	a	Rm

Table F4-88

Decode fields				Instruction page
L	N	a	Rm	
0	xx	x	xxxx	Unallocated.
1	00	x	!= 11x1	VLD1 (single element to all lanes)
1	00	x	1101	VLD1 (single element to all lanes)

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields				Instruction page
L	N	a	Rm	
1	00	x	1111	VLD1 (single element to all lanes)
1	01	x	!= 11x1	VLD2 (single 2-element structure to all lanes)
1	01	x	1101	VLD2 (single 2-element structure to all lanes)
1	01	x	1111	VLD2 (single 2-element structure to all lanes)
1	10	0	!= 11x1	VLD3 (single 3-element structure to all lanes)
1	10	0	1101	VLD3 (single 3-element structure to all lanes)
1	10	0	1111	VLD3 (single 3-element structure to all lanes)
1	10	1	xxxx	Unallocated.
1	11	x	!= 11x1	VLD4 (single 4-element structure to all lanes)
1	11	x	1101	VLD4 (single 4-element structure to all lanes)
1	11	x	1111	VLD4 (single 4-element structure to all lanes)

F4.1.24.3 Advanced SIMD load/store single structure to one lane

This section describes the encoding of the Advanced SIMD load/store single structure to one lane instruction class. The encodings in this section are decoded from Advanced SIMD element or structure load/store.

31 30 29 28	27 26 25 24	23	22	21	20	19 16	15 12	11 10	9 8	7 4	3 0
1 1 1 1	0 1 0 0	1	D	L	0	Rn	Vd	!=11 size	N	index_align	Rm

Table F4-89

Decode fields				Instruction page
L	size	N	Rm	
0	00	00	!= 11x1	VST1 (single element from one lane)
0	00	00	1101	VST1 (single element from one lane)
0	00	00	1111	VST1 (single element from one lane)
0	00	01	!= 11x1	VST2 (single 2-element structure from one lane)
0	00	01	1101	VST2 (single 2-element structure from one lane)
0	00	01	1111	VST2 (single 2-element structure from one lane)
0	00	10	!= 11x1	VST3 (single 3-element structure from one lane)
0	00	10	1101	VST3 (single 3-element structure from one lane)
0	00	10	1111	VST3 (single 3-element structure from one lane)
0	00	11	!= 11x1	VST4 (single 4-element structure from one lane)
0	00	11	1101	VST4 (single 4-element structure from one lane)
0	00	11	1111	VST4 (single 4-element structure from one lane)

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields				Instruction page
L	size	N	Rm	
0	01	00	!= 11x1	VST1 (single element from one lane)
0	01	00	1101	VST1 (single element from one lane)
0	01	00	1111	VST1 (single element from one lane)
0	01	01	!= 11x1	VST2 (single 2-element structure from one lane)
0	01	01	1101	VST2 (single 2-element structure from one lane)
0	01	01	1111	VST2 (single 2-element structure from one lane)
0	01	10	!= 11x1	VST3 (single 3-element structure from one lane)
0	01	10	1101	VST3 (single 3-element structure from one lane)
0	01	10	1111	VST3 (single 3-element structure from one lane)
0	01	11	!= 11x1	VST4 (single 4-element structure from one lane)
0	01	11	1101	VST4 (single 4-element structure from one lane)
0	01	11	1111	VST4 (single 4-element structure from one lane)
0	10	00	!= 11x1	VST1 (single element from one lane)
0	10	00	1101	VST1 (single element from one lane)
0	10	00	1111	VST1 (single element from one lane)
0	10	01	!= 11x1	VST2 (single 2-element structure from one lane)
0	10	01	1101	VST2 (single 2-element structure from one lane)
0	10	01	1111	VST2 (single 2-element structure from one lane)
0	10	10	!= 11x1	VST3 (single 3-element structure from one lane)
0	10	10	1101	VST3 (single 3-element structure from one lane)
0	10	10	1111	VST3 (single 3-element structure from one lane)
0	10	11	!= 11x1	VST4 (single 4-element structure from one lane)
0	10	11	1101	VST4 (single 4-element structure from one lane)
0	10	11	1111	VST4 (single 4-element structure from one lane)
1	00	00	!= 11x1	VLD1 (single element to one lane)
1	00	00	1101	VLD1 (single element to one lane)
1	00	00	1111	VLD1 (single element to one lane)
1	00	01	!= 11x1	VLD2 (single 2-element structure to one lane)
1	00	01	1101	VLD2 (single 2-element structure to one lane)
1	00	01	1111	VLD2 (single 2-element structure to one lane)
1	00	10	!= 11x1	VLD3 (single 3-element structure to one lane)
1	00	10	1101	VLD3 (single 3-element structure to one lane)
1	00	10	1111	VLD3 (single 3-element structure to one lane)
1	00	11	!= 11x1	VLD4 (single 4-element structure to one lane)
1	00	11	1101	VLD4 (single 4-element structure to one lane)
1	00	11	1111	VLD4 (single 4-element structure to one lane)
1	01	00	!= 11x1	VLD1 (single element to one lane)
1	01	00	1101	VLD1 (single element to one lane)
1	01	00	1111	VLD1 (single element to one lane)

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode fields				Instruction page
L	size	N	Rm	
1	01	01	!= 11x1	VLD2 (single 2-element structure to one lane)
1	01	01	1101	VLD2 (single 2-element structure to one lane)
1	01	01	1111	VLD2 (single 2-element structure to one lane)
1	01	10	!= 11x1	VLD3 (single 3-element structure to one lane)
1	01	10	1101	VLD3 (single 3-element structure to one lane)
1	01	10	1111	VLD3 (single 3-element structure to one lane)
1	01	11	!= 11x1	VLD4 (single 4-element structure to one lane)
1	01	11	1101	VLD4 (single 4-element structure to one lane)
1	01	11	1111	VLD4 (single 4-element structure to one lane)
1	10	00	!= 11x1	VLD1 (single element to one lane)
1	10	00	1101	VLD1 (single element to one lane)
1	10	00	1111	VLD1 (single element to one lane)
1	10	01	!= 11x1	VLD2 (single 2-element structure to one lane)
1	10	01	1101	VLD2 (single 2-element structure to one lane)
1	10	01	1111	VLD2 (single 2-element structure to one lane)
1	10	10	!= 11x1	VLD3 (single 3-element structure to one lane)
1	10	10	1101	VLD3 (single 3-element structure to one lane)
1	10	10	1111	VLD3 (single 3-element structure to one lane)
1	10	11	!= 11x1	VLD4 (single 4-element structure to one lane)
1	10	11	1101	VLD4 (single 4-element structure to one lane)
1	10	11	1111	VLD4 (single 4-element structure to one lane)

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F4.2 About the A32 Advanced SIMD and floating-point instructions and their encoding

The Advanced SIMD and floating-point instructions are common to the T32 and A32 instruction sets. These instructions perform Advanced SIMD and floating-point operations on a common register file, the SIMD&FP register file. This means:

- In general, the instructions that load or store registers in this file, or move data between general-purpose registers and this register file, are common to the Advanced SIMD and floating-point instructions.
- There are distinct Advanced SIMD data-processing instructions and floating-point data-processing instructions.

All A32 Advanced SIMD and floating-point instructions have 32-bit encodings. Different groups of these instructions are decoded from different points in the 32-bit A32 instruction decode structure. Table F4-90 shows these instruction groups, and where each group is decoded from the overall A32 decode structure:

Table F4-90 Advanced SIMD and floating-point instructions in the A32 decode structure

Advanced SIMD and floating-point instruction group	A32 decode is from
Advanced SIMD and System register load/store and 64-bit move	System register access, Advanced SIMD, floating-point, and Supervisor call
Floating-point data-processing	System register access, Advanced SIMD, floating-point, and Supervisor call
Advanced SIMD and System register 32-bit move	System register access, Advanced SIMD, floating-point, and Supervisor call
Advanced SIMD data-processing	Unconditional instructions
Advanced SIMD element or structure load/store	Unconditional instructions

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Chapter F5
T32 and A32 Base Instruction Set Instruction Descriptions

This chapter describes each instruction. It contains the following sections:

- Alphabetical list of T32 and A32 base instruction set instructions.
- Encoding and use of banked register transfer instructions.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1 Alphabetical list of T32 and A32 base instruction set instructions

This section lists every instruction in the T32 and A32 base instruction sets. For details of the format used see Format of instruction descriptions.

This section is formatted so that each instruction description starts on a new page.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.1 ADC, ADCS (immediate)

Add with Carry (immediate) adds an immediate value and the Carry flag value to a register value, and writes the result to the destination register.

If the destination register is not the PC, the ADCS variant of the instruction updates the condition flags based on the result.

The field descriptions for `<Rd>` identify the encodings where the PC is permitted as the destination register. Arm deprecates any use of these encodings. However, when the destination register is the PC:

- The ADC variant of the instruction is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- The ADCS variant of the instruction performs an exception return without the use of the stack. In this case:
  - The PE branches to the address written to the PC, and restores PSTATE from SPSR_<current_mode>.
  - The PE checks SPSR_<current_mode> for an illegal return event. See Illegal return events from AArch32 state.
  - The instruction is UNDEFINED in Hyp mode.
  - The instruction is CONSTRAINED UNPREDICTABLE in User mode and System mode.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding for the ADC variant

Applies when `(S == 0)`

```
ADC{<c>}{<q>}  {<Rd>, }<Rn>, #<const>
```

Encoding for the ADCS variant

Applies when `(S == 1)`

```
ADCS{<c>}{<q>}  {<Rd>, }<Rn>, #<const>
```

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant boolean setflags = (S == '1');
constant bits(32) imm32 = A32ExpandImm(imm12);
```

T1

Encoding for the ADC variant

Applies when `(S == 0)`

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
ADC{<c>}{<q>}  {<Rd>, }<Rn>, #<const>
```

Encoding for the ADCS variant

Applies when `(S == 1)`

```
ADCS{<c>}{<q>}  {<Rd>, }<Rn>, #<const>
```

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant boolean setflags = (S == '1');
constant bits(32) imm32 = T32ExpandImm(i:imm3:imm8);
// Armv8-A removes UNPREDICTABLE for R13
if d == 15 || n == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

For the “A1 ADC” and “A1 ADCS” variants: is the general-purpose destination register, encoded in the “Rd” field. If omitted, this register is the same as `<Rn>`. Arm deprecates using the PC as the destination register, but if the PC is used:

- For the ADC variant, the instruction is a branch to the address calculated by the operation. This is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- For the ADCS variant, the instruction performs an exception return, that restores PSTATE from SPSR_<current_mode>.

For the “T1 ADC” and “T1 ADCS” variants: is the general-purpose destination register, encoded in the “Rd” field. If omitted, this register is the same as `<Rn>`.

<Rn>

For the “A1 ADC” and “A1 ADCS” variants: is the general-purpose source register, encoded in the “Rn” field. The PC can be used, but this is deprecated.

For the “T1 ADC” and “T1 ADCS” variants: is the general-purpose source register, encoded in the “Rn” field.

<const>

For the “A1 ADC” and “A1 ADCS” variants: an immediate value. See Modified immediate constants in A32 instructions for the range of values.

For the “T1 ADC” and “T1 ADCS” variants: an immediate value. See Modified immediate constants in T32 instructions for the range of values.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    bits(32) result;
    bits(4) nzcv;
    (result, nzcv) = AddWithCarry(R[n], imm32, PSTATE.C);
    if d == 15 then          // Can only occur for A32 encoding
        if setflags then
            ALUExceptionReturn(result);
        else
            ALUWritePC(result);
    else
        R[d] = result;
        if setflags then
            PSTATE.<N,Z,C,V> = nzcv;
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.2 ADC, ADCS (register)

Add with Carry (register) adds a register value, the Carry flag value, and an optionally-shifted register value, and writes the result to the destination register.

If the destination register is not the PC, the ADCS variant of the instruction updates the condition flags based on the result.

The field descriptions for `<Rd>` identify the encodings where the PC is permitted as the destination register. Arm deprecates any use of these encodings. However, when the destination register is the PC:

- The ADC variant of the instruction is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- The ADCS variant of the instruction performs an exception return without the use of the stack. In this case:
  - The PE branches to the address written to the PC, and restores PSTATE from SPSR_<current_mode>.
  - The PE checks SPSR_<current_mode> for an illegal return event. See Illegal return events from AArch32 state.
  - The instruction is UNDEFINED in Hyp mode.
  - The instruction is CONSTRAINED UNPREDICTABLE in User mode and System mode.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1 and T2).

A1

31–28	27–24	23–21	20	19–16	15–12	11–7	6–5	4	3–0
!=1111	0 0 0 0	1 0 1	S	Rn	Rd	imm5	stype	0	Rm
cond		opc							

Encoding for the ADC, rotate right with extend variant

Applies when `(S == 0 && imm5 == 00000 && stype == 11)`

```
ADC{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, RRX
```

Encoding for the ADC, shift or rotate by value variant

Applies when `(S == 0 && !(imm5 == 00000 && stype == 11))`

```
ADC{<c>}{<q>}  {<Rd>, }<Rn>, <Rm> {, <shift> #<amount>}
```

Encoding for the ADCS, rotate right with extend variant

Applies when `(S == 1 && imm5 == 00000 && stype == 11)`

```
ADCS{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, RRX
```

Encoding for the ADCS, shift or rotate by value variant

Applies when `(S == 1 && !(imm5 == 00000 && stype == 11))`

```
ADCS{<c>}{<q>}  {<Rd>, }<Rn>, <Rm> {, <shift> #<amount>}
```

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean setflags = (S == '1');
SRType shift_t;
integer shift_n;
(shift_t, shift_n) = DecodeImmShift(stype, imm5);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T1

15–10	9–6	5–3	2–0
0 1 0 0 0 0	0 1 0 1	Rm	Rdn

op

Encoding

```
ADC<c>{<q>}  {<Rdn>, }<Rdn>, <Rm> // (InITBlock())
```

```
ADCS{<q>}  {<Rdn>, }<Rdn>, <Rm> // (Outside IT block)
```

Decode for this encoding

```
constant integer d = UInt(Rdn);
constant integer n = UInt(Rdn);
constant integer m = UInt(Rm);
constant boolean setflags = !InITBlock();
constant SRType shift_t = SRType_LSL;
constant integer shift_n = 0;
```

T2

15–9	8–5	4	3–0	15	14–12	11–8	7–6	5–4	3–0
1 1 1 0 1 0 1	1 0 1 0	S	Rn	(0)	imm3	Rd	imm2	stype	Rm

op1

Encoding for the ADC, rotate right with extend variant

Applies when `(S == 0 && imm3 == 000 && imm2 == 00 && stype == 11)`

```
ADC{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, RRX
```

Encoding for the ADC, shift or rotate by value variant

Applies when `(S == 0 && !(imm3 == 000 && imm2 == 00 && stype == 11))`

```
ADC{<c>}{<q>}  {<Rd>, }<Rn>, <Rm> {, <shift> #<amount>}
```

```
ADC<c>.W  {<Rd>, }<Rn>, <Rm> // (Inside IT block, and <Rd>, <Rn>, <Rm> can be represented in T1)
```

Encoding for the ADCS, rotate right with extend variant

Applies when `(S == 1 && imm3 == 000 && imm2 == 00 && stype == 11)`

```
ADCS{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, RRX
```

Encoding for the ADCS, shift or rotate by value variant

Applies when `(S == 1 && !(imm3 == 000 && imm2 == 00 && stype == 11))`

```
ADCS{<c>}{<q>}  {<Rd>, }<Rn>, <Rm> {, <shift> #<amount>}
```

```
ADCS.W  {<Rd>, }<Rn>, <Rm> // (Outside IT block, and <Rd>, <Rn>, <Rm> can be represented in T1)
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean setflags = (S == '1');
SRType shift_t;
integer shift_n;
(shift_t, shift_n) = DecodeImmShift(stype, imm3:imm2);
// Armv8-A removes UNPREDICTABLE for R13
if d == 15 || n == 15 || m == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

For the “A1 ADC, rotate right with extend”, “A1 ADC, shift or rotate by value”, “A1 ADCS, rotate right with extend”, and “A1 ADCS, shift or rotate by value” variants: is the general-purpose destination register, encoded in the “Rd” field. If omitted, this register is the same as `<Rn>`. Arm deprecates using the PC as the destination register, but if the PC is used:

- For the ADC variant, the instruction is a branch to the address calculated by the operation. This is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- For the ADCS variant, the instruction performs an exception return, that restores PSTATE from SPSR_<current_mode>.

For the “T2 ADC, rotate right with extend”, “T2 ADC, shift or rotate by value”, “T2 ADCS, rotate right with extend”, and “T2 ADCS, shift or rotate by value” variants: is the general-purpose destination register, encoded in the “Rd” field. If omitted, this register is the same as `<Rn>`.

<Rn>

For the “A1 ADC, rotate right with extend”, “A1 ADC, shift or rotate by value”, “A1 ADCS, rotate right with extend”, and “A1 ADCS, shift or rotate by value” variants: is the first general-purpose source register, encoded in the “Rn” field. The PC can be used, but this is deprecated.

For the “T2 ADC, rotate right with extend”, “T2 ADC, shift or rotate by value”, “T2 ADCS, rotate right with extend”, and “T2 ADCS, shift or rotate by value” variants: is the first general-purpose source register, encoded in the “Rn” field.

<Rm>

For the “A1 ADC, rotate right with extend”, “A1 ADC, shift or rotate by value”, “A1 ADCS, rotate right with extend”, and “A1 ADCS, shift or rotate by value” variants: is the second general-purpose source register, encoded in the “Rm” field. The PC can be used, but this is deprecated.

For the “T1”, “T2 ADC, rotate right with extend”, “T2 ADC, shift or rotate by value”, “T2 ADCS, rotate right with extend”, and “T2 ADCS, shift or rotate by value” variants: is the second general-purpose source register, encoded in the “Rm” field.

<shift>

Is the type of shift to be applied to the second source register, encoded in “stype”:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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

<amount>

For the “A1 ADC, shift or rotate by value” and “A1 ADCS, shift or rotate by value” variants: is the shift amount, in the range 1 to 31 (when <shift> = LSL or ROR) or 1 to 32 (when <shift> = LSR or ASR) encoded in the “imm5” field as <amount> modulo 32.

For the “T2 ADC, shift or rotate by value” and “T2 ADCS, shift or rotate by value” variants: is the shift amount, in the range 1 to 31 (when <shift> = LSL or ROR) or 1 to 32 (when <shift> = LSR or ASR), encoded in the “imm3:imm2” field as <amount> modulo 32.

<Rdn>

Is the first general-purpose source register and the destination register, encoded in the “Rdn” field.

In T32 assembly:

- Outside an IT block, if `ADCS <Rd>, <Rn>, <Rd>` has `<Rd>` and `<Rn>` both in the range R0-R7, it is assembled using encoding T1 as though `ADCS <Rd>, <Rn>` had been written.
- Inside an IT block, if `ADC<c> <Rd>, <Rn>, <Rd>` has `<Rd>` and `<Rn>` both in the range R0-R7, it is assembled using encoding T1 as though `ADC<c> <Rd>, <Rn>` had been written.

To prevent either of these happening, use the .W qualifier.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) shifted = Shift(R[m], shift_t, shift_n, PSTATE.C);
    bits(32) result;
    bits(4) nzcv;
    (result, nzcv) = AddWithCarry(R[n], shifted, PSTATE.C);
    if d == 15 then          // Can only occur for A32 encoding
        if setflags then
            ALUExceptionReturn(result);
        else
            ALUWritePC(result);
    else
        R[d] = result;
        if setflags then
            PSTATE.<N,Z,C,V> = nzcv;
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.3 ADC, ADCS (register-shifted register)

Add with Carry (register-shifted register) adds a register value, the Carry flag value, and a register-shifted register value. It writes the result to the destination register, and can optionally update the condition flags based on the result.

31–28	27–24	23–21	20	19–16	15–12	11–8	7	6–5	4	3–0
!=1111 (cond)	0 0 0 0	1 0 1 (opc)	S	Rn	Rd	Rs	0	stype	1	Rm

Encoding for the Flag setting variant

Applies when `(S == 1)`

```
ADCS{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, <shift> <Rs>
```

Encoding for the Not flag setting variant

Applies when `(S == 0)`

```
ADC{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, <shift> <Rs>
```

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant integer s = UInt(Rs);
constant boolean setflags = (S == '1');
constant SRType shift_t = DecodeRegShift(stype);
if d == 15 || n == 15 || m == 15 || s == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

Is the general-purpose destination register, encoded in the “Rd” field.

<Rn>

Is the first general-purpose source register, encoded in the “Rn” field.

<Rm>

Is the second general-purpose source register, encoded in the “Rm” field.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

<shift>

Is the type of shift to be applied to the second source register, encoded in “stype”:

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

<Rs>

Is the third general-purpose source register holding a shift amount in its bottom 8 bits, encoded in the “Rs” field.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant integer shift_n = UInt(R[s]<7:0>);
    constant bits(32) shifted = Shift(R[m], shift_t, shift_n, PSTATE.C);
    bits(32) result;
    bits(4) nzcv;
    (result, nzcv) = AddWithCarry(R[n], shifted, PSTATE.C);
    R[d] = result;
    if setflags then
        PSTATE.<N,Z,C,V> = nzcv;
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.4 ADD (immediate, to PC)

Add to PC adds an immediate value to the Align(PC, 4) value to form a PC-relative address, and writes the result to the destination register. Arm recommends that, where possible, software avoids using this alias.

This is a pseudo-instruction of ADR. This means:

- The encodings in this description are named to match the encodings of ADR.
- The assembler syntax is used only for assembly, and is not used on disassembly.
- The description of ADR gives the operational pseudocode, any CONSTRAINED UNPREDICTABLE behavior, and any operational information for this instruction.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1 and T3).

A1

Encoding for the A1 variant

```
ADD{<c>}{<q>}  <Rd>, PC, #<const>
```

is equivalent to

```
ADR{<c>}{<q>}  <Rd>, <label>
```

T1

Encoding for the T1 variant

```
ADD{<c>}{<q>}  <Rd>, PC, #<imm8>
```

is equivalent to

```
ADR{<c>}{<q>}  <Rd>, <label>
```

T3

Encoding for the T3 variant

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
ADD{<c>}{<q>}  <Rd>, PC, #<imm12>
```

```
ADDW{<c>}{<q>}  <Rd>, PC, #<imm12> // (UInt(Rd) < 8 && UInt(i :: imm3 :: imm8) < 256)
```

is equivalent to

```
ADR{<c>}{<q>}  <Rd>, <label>
```

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

For the “A1” variant: is the general-purpose destination register, encoded in the “Rd” field. If the PC is used, the instruction is a branch to the address calculated by the operation. This is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.

For the “T1” and “T3” variants: is the general-purpose destination register, encoded in the “Rd” field.

<const>

An immediate value. See Modified immediate constants in A32 instructions for the range of values.

<imm8>

Is an unsigned immediate, a multiple of 4, in the range 0 to 1020, encoded in the “imm8” field as <imm8>/4.

<imm12>

Is a 12-bit unsigned immediate, in the range 0 to 4095, encoded in the “i:imm3:imm8” field.

Operation

The description of ADR gives the operational pseudocode for this instruction.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.5 ADD, ADDS (immediate)

Add (immediate) adds an immediate value to a register value, and writes the result to the destination register.

If the destination register is not the PC, the ADDS variant of the instruction updates the condition flags based on the result.

The field descriptions for `<Rd>` identify the encodings where the PC is permitted as the destination register. If the destination register is the PC:

- The ADD variant of the instruction is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- The ADDS variant of the instruction performs an exception return without the use of the stack. Arm deprecates use of this instruction. However, in this case:
  - The PE branches to the address written to the PC, and restores PSTATE from SPSR_<current_mode>.
  - The PE checks SPSR_<current_mode> for an illegal return event. See Illegal return events from AArch32 state.
  - The instruction is UNDEFINED in Hyp mode.
  - The instruction is CONSTRAINED UNPREDICTABLE in User mode and System mode.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1, T2, T3, and T4).

A1

Encoding for the ADD variant

Applies when `(S == 0 && Rn != 11x1)`

```
ADD{<c>}{<q>}  {<Rd>, }<Rn>, #<const>
```

Encoding for the ADDS variant

Applies when `(S == 1 && Rn != 1101)`

```
ADDS{<c>}{<q>}  {<Rd>, }<Rn>, #<const>
```

Decode for all variants of this encoding

```
if Rn == '1111' && S == '0' then SEE "ADR";
if Rn == '1101' then SEE "ADD (SP plus immediate)";
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant boolean setflags = (S == '1');
constant bits(32) imm32 = A32ExpandImm(imm12);
```

T1

Encoding

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
ADD<c>{<q>}  <Rd>, <Rn>, #<imm3> // (InITBlock())
```

```
ADDS{<q>}  <Rd>, <Rn>, #<imm3> // (Outside IT block)
```

Decode for this encoding

```
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant boolean setflags = !InITBlock();
constant bits(32) imm32 = ZeroExtend(imm3, 32);
```

T2

15 13	12 11	10 8	7 0
0 0 1	1 0	Rdn	imm8
	op		

Encoding

```
ADD<c>{<q>}  <Rdn>, #<imm8> // (InITBlock() && UInt(imm8) < 8)
```

```
ADD<c>{<q>}  {<Rdn>, }<Rdn>, #<imm8> // (Inside IT block, and <Rdn>, <imm8> cannot be represented in T1)
```

```
ADDS{<q>}  <Rdn>, #<imm8> // (Outside IT block, and <Rdn>, <imm8> can be represented in T1)
```

```
ADDS{<q>}  {<Rdn>, }<Rdn>, #<imm8> // (Outside IT block, and <Rdn>, <imm8> cannot be represented in T1)
```

Decode for this encoding

```
constant integer d = UInt(Rdn);
constant integer n = UInt(Rdn);
constant boolean setflags = !InITBlock();
constant bits(32) imm32 = ZeroExtend(imm8, 32);
```

T3

15 11	10	9	8 5	4	3 0	15	14 12	11 8	7 0
1 1 1 1 0	i	0	1 0 0 0	S	!=1101	0	imm3	Rd	imm8
			op1		Rn				

Encoding for the ADD variant

Applies when `(S == 0)`

```
ADD{<c>}{<q>}  {<Rd>, }<Rn>, #<const>
```

```
ADD<c>.W  {<Rd>, }<Rn>, #<const> // (Inside IT block, and <Rd>, <Rn>, <const> can be represented in T1
    ↪or T2)
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Encoding for the ADDS variant

Applies when `(S == 1 && Rd != 1111)`

```
ADDS{<c>}{<q>}  {<Rd>, }<Rn>, #<const>
```

```
ADDS.W  {<Rd>, }<Rn>, #<const> // (Outside IT block, and <Rd>, <Rn>, <const> can be represented in T1 or
    ↪T2)
```

Decode for all variants of this encoding

```
if Rd == '1111' && S == '1' then SEE "CMN (immediate)";
if Rn == '1101' then SEE "ADD (SP plus immediate)";
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant boolean setflags = (S == '1');
constant bits(32) imm32 = T32ExpandImm(i:imm3:imm8);
// Armv8-A removes UNPREDICTABLE for R13
if (d == 15 && !setflags) || n == 15 then UNPREDICTABLE;
```

T4

15 … 11	10	9 … 8	7	6	5	4	3 … 0	15	14 … 12	11 … 8	7 … 0
1 1 1 1 0	i	1 0	0	0	0	0	!=11x1	0	imm3	Rd	imm8
			o1		o2		Rn				

Encoding

```
ADD{<c>}{<q>}  {<Rd>, }<Rn>, #<imm12> // (<imm12> cannot be represented in T1, T2, or T3)
```

```
ADDW{<c>}{<q>}  {<Rd>, }<Rn>, #<imm12> // (<imm12> can be represented in T1, T2, or T3)
```

Decode for this encoding

```
if Rn == '1111' then SEE "ADR";
if Rn == '1101' then SEE "ADD (SP plus immediate)";
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant boolean setflags = FALSE;
constant bits(32) imm32 = ZeroExtend(i:imm3:imm8, 32);
// Armv8-A removes UNPREDICTABLE for R13
if d == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

For the “A1 ADD” and “A1 ADDS” variants: is the general-purpose destination register, encoded in the “Rd” field. If omitted, this register is the same as `<Rn>`. If the PC is used:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- For the ADD variant, the instruction is a branch to the address calculated by the operation. This is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- For the ADDS variant, the instruction performs an exception return, that restores PSTATE from SPSR_<current_mode>. Arm deprecates use of this instruction.

For the “T1”, “T3 ADD”, “T3 ADDS”, and “T4” variants: is the general-purpose destination register, encoded in the “Rd” field. If omitted, this register is the same as `<Rn>`.

<Rn>

For the “A1 ADD”, “A1 ADDS”, and “T4” variants: is the general-purpose source register, encoded in the “Rn” field. If the SP is used, see ADD (SP plus immediate). If the PC is used, see ADR.

For the “T1” variant: is the general-purpose source register, encoded in the “Rn” field.

For the “T3 ADD” and “T3 ADDS” variants: is the general-purpose source register, encoded in the “Rn” field. If the SP is used, see ADD (SP plus immediate).

<const>

For the “A1 ADD” and “A1 ADDS” variants: an immediate value. See Modified immediate constants in A32 instructions for the range of values.

For the “T3 ADD” and “T3 ADDS” variants: an immediate value. See Modified immediate constants in T32 instructions for the range of values.

<imm3>

Is a 3-bit unsigned immediate, in the range 0 to 7, encoded in the “imm3” field.

<Rdn>

Is the general-purpose source and destination register, encoded in the “Rdn” field.

<imm8>

Is a 8-bit unsigned immediate, in the range 0 to 255, encoded in the “imm8” field.

<imm12>

Is a 12-bit unsigned immediate, in the range 0 to 4095, encoded in the “i:imm3:imm8” field.

When multiple encodings of the same length are available for an instruction, encoding T3 is preferred to encoding T4 (if encoding T4 is required, use the `ADDW` syntax). Encoding T1 is preferred to encoding T2 if `<Rd>` is specified and encoding T2 is preferred to encoding T1 if `<Rd>` is omitted.

Operation

```
if CurrentInstrSet() == InstrSet_A32 then
    if ConditionPassed() then
        EncodingSpecificOperations();
        bits(32) result;
        bits(4) nzcv;
        (result, nzcv) = AddWithCarry(R[n], imm32, '0');
        if d == 15 then          // Can only occur for A32 encoding
            if setflags then
                ALUExceptionReturn(result);
            else
                ALUWritePC(result);
        else
            R[d] = result;
            if setflags then
                PSTATE.<N,Z,C,V> = nzcv;
else
    if ConditionPassed() then
        EncodingSpecificOperations();
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
bits(32) result;
bits(4) nzcv;
(result, nzcv) = AddWithCarry(R[n], imm32, '0');
R[d] = result;
if setflags then
    PSTATE.<N,Z,C,V> = nzcv;
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.6 ADD, ADDS (register)

Add (register) adds a register value and an optionally-shifted register value, and writes the result to the destination register.

If the destination register is not the PC, the ADDS variant of the instruction updates the condition flags based on the result.

The field descriptions for `<Rd>` identify the encodings where the PC is permitted as the destination register. If the destination register is the PC:

- The ADD variant of the instruction is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- The ADDS variant of the instruction performs an exception return without the use of the stack. Arm deprecates use of this instruction. However, in this case:
  - The PE branches to the address written to the PC, and restores PSTATE from SPSR_<current_mode>.
  - The PE checks SPSR_<current_mode> for an illegal return event. See Illegal return events from AArch32 state.
  - The instruction is UNDEFINED in Hyp mode.
  - The instruction is CONSTRAINED UNPREDICTABLE in User mode and System mode.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1, T2, and T3).

A1

31–28	27–24	23–21	20	19–16	15–12	11–7	6–5	4	3–0
!=1111	0 0 0 0	1 0 0	S	!=1101	Rd	imm5	stype	0	Rm
cond		opc		Rn					

Encoding for the ADD, rotate right with extend variant

Applies when `(S == 0 && imm5 == 00000 && stype == 11)`

```
ADD{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, RRX
```

Encoding for the ADD, shift or rotate by value variant

Applies when `(S == 0 && !(imm5 == 00000 && stype == 11))`

```
ADD{<c>}{<q>}  {<Rd>, }<Rn>, <Rm> {, <shift> #<amount>}
```

Encoding for the ADDS, rotate right with extend variant

Applies when `(S == 1 && imm5 == 00000 && stype == 11)`

```
ADDS{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, RRX
```

Encoding for the ADDS, shift or rotate by value variant

Applies when `(S == 1 && !(imm5 == 00000 && stype == 11))`

```
ADDS{<c>}{<q>}  {<Rd>, }<Rn>, <Rm> {, <shift> #<amount>}
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for all variants of this encoding

```
if Rn == '1101' then SEE "ADD (SP plus register)";
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean setflags = (S == '1');
SRType shift_t;
integer shift_n;
(shift_t, shift_n) = DecodeImmShift(stype, imm5);
```

T1

Encoding

```
ADD<c>{<q>}  <Rd>, <Rn>, <Rm> // (InITBlock())
```

```
ADDS{<q>}  {<Rd>, }<Rn>, <Rm> // (Outside IT block)
```

Decode for this encoding

```
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean setflags = !InITBlock();
constant SRType shift_t = SRType_LSL;
constant integer shift_n = 0;
```

T2

Encoding

Applies when `(!(DN == '1' && Rdn == '101'))`

```
ADD{<c>}{<q>}  {<Rdn>, }<Rdn>, <Rm>
```

```
ADD<c>{<q>}  <Rdn>, <Rm> // (Preferred syntax, Inside IT block)
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for this encoding

```
if (DN:Rdn) == '1101' || Rm == '1101' then SEE "ADD (SP plus register)";
constant integer d = UInt(DN:Rdn);
constant integer n = d;
constant integer m = UInt(Rm);
constant boolean setflags = FALSE;
constant SRType shift_t = SRType_LSL;
constant integer shift_n = 0;
if n == 15 && m == 15 then UNPREDICTABLE;
if d == 15 && InITBlock() && !LastInITBlock() then UNPREDICTABLE;
```

T3

15 … 9	8 … 5	4	3 … 0	15	14 … 12	11 … 8	7 … 6	5 … 4	3 … 0
1 1 1 0 1 0 1	1 0 0 0	S	!=1101	(0)	imm3	Rd	imm2	stype	Rm
	op1		Rn						

Encoding for the ADD, rotate right with extend variant

Applies when `(S == 0 && imm3 == 000 && imm2 == 00 && stype == 11)`

```
ADD{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, RRX
```

Encoding for the ADD, shift or rotate by value variant

Applies when `(S == 0 && !(imm3 == 000 && imm2 == 00 && stype == 11))`

```
ADD{<c>}{<q>}  {<Rd>, }<Rn>, <Rm> {, <shift> #<amount>}
```

```
ADD<c>.W  {<Rd>, }<Rn>, <Rm> // (Inside IT block, and <Rd>, <Rn>, <Rm> can be represented in T1)
```

```
ADD{<c>}.W  {<Rd>, }<Rn>, <Rm> // (<Rd> == <Rn>, and <Rd>, <Rn>, <Rm> can be represented in T2)
```

Encoding for the ADDS, rotate right with extend variant

Applies when `(S == 1 && imm3 == 000 && Rd != 1111 && imm2 == 00 && stype == 11)`

```
ADDS{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, RRX
```

Encoding for the ADDS, shift or rotate by value variant

Applies when `(S == 1 && Rd != 1111 && !(imm3 == 000 && imm2 == 00 && stype == 11))`

```
ADDS{<c>}{<q>}  {<Rd>, }<Rn>, <Rm> {, <shift> #<amount>}
```

```
ADDS.W  {<Rd>, }<Rn>, <Rm> // (Outside IT block, and <Rd>, <Rn>, <Rm> can be represented in T1 or T2)
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for all variants of this encoding

```
if Rd == '1111' && S == '1' then SEE "CMN (register)";
if Rn == '1101' then SEE "ADD (SP plus register)";
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean setflags = (S == '1');
SRType shift_t;
integer shift_n;
(shift_t, shift_n) = DecodeImmShift(stype, imm3:imm2);
// Armv8-A removes UNPREDICTABLE for R13
if (d == 15 && !setflags) || n == 15 || m == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

For the “A1 ADD, rotate right with extend”, “A1 ADD, shift or rotate by value”, “A1 ADDS, rotate right with extend”, and “A1 ADDS, shift or rotate by value” variants: is the general-purpose destination register, encoded in the “Rd” field. If omitted, this register is the same as `<Rn>`. If the PC is used:

- For the ADD variant, the instruction is a branch to the address calculated by the operation. This is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- For the ADDS variant, the instruction performs an exception return, that restores PSTATE from SPSR_<current_mode>. Arm deprecates use of this instruction.

For the “T1” variant: is the general-purpose destination register, encoded in the “Rd” field.

When used inside an IT block, `<Rd>` must be specified. When used outside an IT block, `<Rd>` is optional, and:

- If omitted, this register is the same as `<Rn>`.
- If present, encoding T1 is preferred to encoding T2.

For the “T3 ADD, rotate right with extend”, “T3 ADD, shift or rotate by value”, “T3 ADDS, rotate right with extend”, and “T3 ADDS, shift or rotate by value” variants: is the general-purpose destination register, encoded in the “Rd” field. If omitted, this register is the same as `<Rn>`.

<Rn>

For the “A1 ADD, rotate right with extend”, “A1 ADD, shift or rotate by value”, “A1 ADDS, rotate right with extend”, and “A1 ADDS, shift or rotate by value” variants: is the first general-purpose source register, encoded in the “Rn” field. The PC can be used. If the SP is used, see ADD (SP plus register).

For the “T1” variant: is the first general-purpose source register, encoded in the “Rn” field.

For the “T3 ADD, rotate right with extend”, “T3 ADD, shift or rotate by value”, “T3 ADDS, rotate right with extend”, and “T3 ADDS, shift or rotate by value” variants: is the first general-purpose source register, encoded in the “Rn” field. If the SP is used, see ADD (SP plus register).

<Rm>

For the “A1 ADD, rotate right with extend”, “A1 ADD, shift or rotate by value”, “A1 ADDS, rotate right with extend”, and “A1 ADDS, shift or rotate by value” variants: is the second general-purpose source register, encoded in the “Rm” field. The PC can be used, but this is deprecated.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

For the “T1”, “T3 ADD, rotate right with extend”, “T3 ADD, shift or rotate by value”, “T3 ADDS, rotate right with extend”, and “T3 ADDS, shift or rotate by value” variants: is the second general-purpose source register, encoded in the “Rm” field.

For the “T2” variant: is the second general-purpose source register, encoded in the “Rm” field. The PC can be used.

<shift>

Is the type of shift to be applied to the second source register, encoded in “stype”:

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

<amount>

For the “A1 ADD, shift or rotate by value” and “A1 ADDS, shift or rotate by value” variants: is the shift amount, in the range 1 to 31 (when <shift> = LSL or ROR) or 1 to 32 (when <shift> = LSR or ASR) encoded in the “imm5” field as <amount> modulo 32.

For the “T3 ADD, shift or rotate by value” and “T3 ADDS, shift or rotate by value” variants: is the shift amount, in the range 1 to 31 (when <shift> = LSL or ROR) or 1 to 32 (when <shift> = LSR or ASR), encoded in the “imm3:imm2” field as <amount> modulo 32.

<Rdn>

Is the general-purpose source and destination register, encoded in the “DN:Rdn” field. If the PC is used, the instruction is a branch to the address calculated by the operation. This is a simple branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.

The assembler language allows `<Rdn>` to be specified once or twice in the assembler syntax. When used inside an IT block, and `<Rdn>` and `<Rm>` are in the range R0 to R7, `<Rdn>` must be specified once so that encoding T2 is preferred to encoding T1. In all other cases there is no difference in behavior when `<Rdn>` is specified once or twice.

Inside an IT block, if `ADD<c> <Rd>, <Rn>, <Rd>` cannot be assembled using encoding T1, it is assembled using encoding T2 as though `ADD<c> <Rd>, <Rn>` had been written. To prevent this happening, use the `.W` qualifier.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) shifted = Shift(R[m], shift_t, shift_n, PSTATE.C);
    bits(32) result;
    bits(4) nzcv;
    (result, nzcv) = AddWithCarry(R[n], shifted, '0');
    if d == 15 then
        if setflags then
            ALUExceptionReturn(result);
        else
            ALUWritePC(result);
    else
        R[d] = result;
        if setflags then
            PSTATE.<N,Z,C,V> = nzcv;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.7 ADD, ADDS (register-shifted register)

Add (register-shifted register) adds a register value and a register-shifted register value. It writes the result to the destination register, and can optionally update the condition flags based on the result.

31–28	27–24	23–21	20	19–16	15–12	11–8	7	6–5	4	3–0
!=1111 (cond)	0 0 0 0	1 0 0 (opc)	S	Rn	Rd	Rs	0	stype	1	Rm

Encoding for the Flag setting variant

Applies when `(S == 1)`

```
ADDS{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, <shift> <Rs>
```

Encoding for the Not flag setting variant

Applies when `(S == 0)`

```
ADD{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, <shift> <Rs>
```

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant integer s = UInt(Rs);
constant boolean setflags = (S == '1');
constant SRType shift_t = DecodeRegShift(stype);
if d == 15 || n == 15 || m == 15 || s == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

Is the general-purpose destination register, encoded in the “Rd” field.

<Rn>

Is the first general-purpose source register, encoded in the “Rn” field.

<Rm>

Is the second general-purpose source register, encoded in the “Rm” field.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

<shift>

Is the type of shift to be applied to the second source register, encoded in “stype”:

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

<Rs>

Is the third general-purpose source register holding a shift amount in its bottom 8 bits, encoded in the “Rs” field.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant integer shift_n = UInt(R[s]<7:0>);
    constant bits(32) shifted = Shift(R[m], shift_t, shift_n, PSTATE.C);
    bits(32) result;
    bits(4) nzcv;
    (result, nzcv) = AddWithCarry(R[n], shifted, '0');
    R[d] = result;
    if setflags then
        PSTATE.<N,Z,C,V> = nzcv;
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.8 ADD, ADDS (SP plus immediate)

Add to SP (immediate) adds an immediate value to the SP value, and writes the result to the destination register.

If the destination register is not the PC, the ADDS variant of the instruction updates the condition flags based on the result.

The field descriptions for `<Rd>` identify the encodings where the PC is permitted as the destination register. However, when the destination register is the PC:

- The ADD variant of the instruction is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- The ADDS variant of the instruction performs an exception return without the use of the stack. Arm deprecates use of this instruction. However, in this case:
  - The PE branches to the address written to the PC, and restores PSTATE from SPSR_<current_mode>.
  - The PE checks SPSR_<current_mode> for an illegal return event. See Illegal return events from AArch32 state.
  - The instruction is UNDEFINED in Hyp mode.
  - The instruction is CONSTRAINED UNPREDICTABLE in User mode and System mode.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1, T2, T3, and T4).

A1

Encoding for the ADD variant

Applies when `(S == 0)`

```
ADD{<c>}{<q>}  {<Rd>, }SP, #<const>
```

Encoding for the ADDS variant

Applies when `(S == 1)`

```
ADDS{<c>}{<q>}  {<Rd>, }SP, #<const>
```

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant boolean setflags = (S == '1');
constant bits(32) imm32 = A32ExpandImm(imm12);
```

T1

Encoding

```
ADD{<c>}{<q>}  <Rd>, SP, #<imm8>
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for this encoding

```
constant integer d = UInt(Rd);
constant boolean setflags = FALSE;
constant bits(32) imm32 = ZeroExtend(imm8:'00', 32);
```

T2

Encoding

```
ADD{<c>}{<q>}  {SP, }SP, #<imm7>
```

Decode for this encoding

```
constant integer d = 13;
constant boolean setflags = FALSE;
constant bits(32) imm32 = ZeroExtend(imm7:'00', 32);
```

T3

Encoding for the ADD variant

Applies when `(S == 0)`

```
ADD{<c>}{<q>}  {<Rd>, }SP, #<const>
```

```
ADD{<c>}.W  {<Rd>, }SP, #<const> // (<Rd>, <const> can be represented in T1 or T2)
```

Encoding for the ADDS variant

Applies when `(S == 1 && Rd != 1111)`

```
ADDS{<c>}{<q>}  {<Rd>, }SP, #<const>
```

Decode for all variants of this encoding

```
if Rd == '1111' && S == '1' then SEE "CMN (immediate)";
constant integer d = UInt(Rd);
constant boolean setflags = (S == '1');
constant bits(32) imm32 = T32ExpandImm(i:imm3:imm8);
if d == 15 && !setflags then UNPREDICTABLE;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T4

Encoding

```
ADD{<c>}{<q>}  {<Rd>, }SP, #<imm12> // (<imm12> cannot be represented in T1, T2, or T3)
```

```
ADDW{<c>}{<q>}  {<Rd>, }SP, #<imm12> // (<imm12> can be represented in T1, T2, or T3)
```

Decode for this encoding

```
constant integer d = UInt(Rd);
constant boolean setflags = FALSE;
constant bits(32) imm32 = ZeroExtend(i:imm3:imm8, 32);
if d == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

For the “A1 ADD” and “A1 ADDS” variants: is the general-purpose destination register, encoded in the “Rd” field. If omitted, this register is the SP. Arm deprecates using the PC as the destination register, but if the PC is used:

- For the ADD variant, the instruction is a branch to the address calculated by the operation. This is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- For the ADDS variant, the instruction performs an exception return, that restores PSTATE from SPSR_<current_mode>.

For the “T1” variant: is the general-purpose destination register, encoded in the “Rd” field.

For the “T3 ADD”, “T3 ADDS”, and “T4” variants: is the general-purpose destination register, encoded in the “Rd” field. If omitted, this register is the SP.

<const>

For the “A1 ADD” and “A1 ADDS” variants: an immediate value. See Modified immediate constants in A32 instructions for the range of values.

For the “T3 ADD” and “T3 ADDS” variants: an immediate value. See Modified immediate constants in T32 instructions for the range of values.

<imm8>

Is an unsigned immediate, a multiple of 4, in the range 0 to 1020, encoded in the “imm8” field as <imm8>/4.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

SP

Is the stack pointer.

<imm7>

Is the unsigned immediate, a multiple of 4, in the range 0 to 508, encoded in the “imm7” field as <imm7>/4.

<imm12>

Is a 12-bit unsigned immediate, in the range 0 to 4095, encoded in the “i:imm3:imm8” field.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    bits(32) result;
    bits(4) nzcv;
    (result, nzcv) = AddWithCarry(R[13], imm32, '0');
    if d == 15 then          // Can only occur for A32 encoding
        if setflags then
            ALUExceptionReturn(result);
        else
            ALUWritePC(result);
    else
        R[d] = result;
        if setflags then
            PSTATE.<N,Z,C,V> = nzcv;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.9 ADD, ADDS (SP plus register)

Add to SP (register) adds an optionally-shifted register value to the SP value, and writes the result to the destination register.

If the destination register is not the PC, the ADDS variant of the instruction updates the condition flags based on the result.

The field descriptions for `<Rd>` identify the encodings where the PC is permitted as the destination register. Arm deprecates any use of these encodings. However, when the destination register is the PC:

- The ADD variant of the instruction is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- The ADDS variant of the instruction performs an exception return without the use of the stack. In this case:
  - The PE branches to the address written to the PC, and restores PSTATE from SPSR_<current_mode>.
  - The PE checks SPSR_<current_mode> for an illegal return event. See Illegal return events from AArch32 state.
  - The instruction is UNDEFINED in Hyp mode.
  - The instruction is CONSTRAINED UNPREDICTABLE in User mode and System mode.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1, T2, and T3).

A1

31–28	27–24	23–21	20	19–16	15–12	11–7	6–5	4	3–0
!=1111	0 0 0 0	1 0 0	S	1 1 0 1	Rd	imm5	stype	0	Rm
cond		opc		Rn					

Encoding for the ADD, rotate right with extend variant

Applies when `(S == 0 && imm5 == 00000 && stype == 11)`

```
ADD{<c>}{<q>}  {<Rd>, }SP, <Rm> , RRX
```

Encoding for the ADD, shift or rotate by value variant

Applies when `(S == 0 && !(imm5 == 00000 && stype == 11))`

```
ADD{<c>}{<q>}  {<Rd>, }SP, <Rm> {, <shift> #<amount>}
```

Encoding for the ADDS, rotate right with extend variant

Applies when `(S == 1 && imm5 == 00000 && stype == 11)`

```
ADDS{<c>}{<q>}  {<Rd>, }SP, <Rm> , RRX
```

Encoding for the ADDS, shift or rotate by value variant

Applies when `(S == 1 && !(imm5 == 00000 && stype == 11))`

```
ADDS{<c>}{<q>}  {<Rd>, }SP, <Rm> {, <shift> #<amount>}
```

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant integer m = UInt(Rm);
constant boolean setflags = (S == '1');
SRType shift_t;
integer shift_n;
(shift_t, shift_n) = DecodeImmShift(stype, imm5);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T1

15–10	9–8	7	6–3	2–0
0 1 0 0 0 1	0 0	DM	1 1 0 1	Rdm
	op		Rs	

Encoding

```
ADD{<c>}{<q>}  {<Rdm>, }SP, <Rdm>
```

Decode for this encoding

```
constant integer d = UInt(DM:Rdm);
constant integer m = UInt(DM:Rdm);
constant boolean setflags = FALSE;
constant SRType shift_t = SRType_LSL;
constant integer shift_n = 0;
if d == 15 && InITBlock() && !LastInITBlock() then UNPREDICTABLE;
```

T2

15–10	9–8	7	6–3	2–0
0 1 0 0 0 1	0 0	1	!=1101	1 0 1
	op	D	Rm	Rd

Encoding

```
ADD{<c>}{<q>}  {SP, }SP, <Rm>
```

Decode for this encoding

```
if Rm == '1101' then SEE "encoding T1";
constant integer d = 13;
constant integer m = UInt(Rm);
constant boolean setflags = FALSE;
constant SRType shift_t = SRType_LSL;
constant integer shift_n = 0;
```

T3

15–9	8–5	4	3–0	15	14–12	11–8	7–6	5–4	3–0
1 1 1 0 1 0 1	1 0 0 0	S	1 1 0 1	(0)	imm3	Rd	imm2	stype	Rm
	op1		Rn						

Encoding for the ADD, rotate right with extend variant

Applies when `(S == 0 && imm3 == 000 && imm2 == 00 && stype == 11)`

```
ADD{<c>}{<q>}  {<Rd>, }SP, <Rm>, RRX
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Encoding for the ADD, shift or rotate by value variant

Applies when `(S == 0 && !(imm3 == 000 && imm2 == 00 && stype == 11))`

```
ADD{<c>}{<q>}  {<Rd>, }SP, <Rm> {, <shift> #<amount>}
```

```
ADD{<c>}.W  {<Rd>, }SP, <Rm> // (<Rd>, <Rm> can be represented in T1 or T2)
```

Encoding for the ADDS, rotate right with extend variant

Applies when `(S == 1 && imm3 == 000 && Rd != 1111 && imm2 == 00 && stype == 11)`

```
ADDS{<c>}{<q>}  {<Rd>, }SP, <Rm>, RRX
```

Encoding for the ADDS, shift or rotate by value variant

Applies when `(S == 1 && Rd != 1111 && !(imm3 == 000 && imm2 == 00 && stype == 11))`

```
ADDS{<c>}{<q>}  {<Rd>, }SP, <Rm> {, <shift> #<amount>}
```

Decode for all variants of this encoding

```
if Rd == '1111' && S == '1' then SEE "CMN (register)";
constant integer d = UInt(Rd);
constant integer m = UInt(Rm);
constant boolean setflags = (S == '1');
SRType shift_t;
integer shift_n;
(shift_t, shift_n) = DecodeImmShift(stype, imm3:imm2);
// Armv8-A removes UNPREDICTABLE for R13
if (d == 15 && !setflags) || m == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

For the “A1 ADD, rotate right with extend”, “A1 ADD, shift or rotate by value”, “A1 ADDS, rotate right with extend”, and “A1 ADDS, shift or rotate by value” variants: is the general-purpose destination register, encoded in the “Rd” field. If omitted, this register is the SP. Arm deprecates using the PC as the destination register, but if the PC is used:

- For the ADD variant, the instruction is a branch to the address calculated by the operation. This is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- For the ADDS variant, the instruction performs an exception return, that restores PSTATE from SPSR_<current_mode>.

For the “T3 ADD, rotate right with extend”, “T3 ADD, shift or rotate by value”, “T3 ADDS, rotate right with extend”, and “T3 ADDS, shift or rotate by value” variants: is the general-purpose destination register, encoded in the “Rd” field. If omitted, this register is the SP.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

<Rm>

For the “A1 ADD, rotate right with extend”, “A1 ADD, shift or rotate by value”, “A1 ADDS, rotate right with extend”, “A1 ADDS, shift or rotate by value”, and “T2” variants: is the second general-purpose source register, encoded in the “Rm” field. The PC can be used, but this is deprecated.

For the “T3 ADD, rotate right with extend”, “T3 ADD, shift or rotate by value”, “T3 ADDS, rotate right with extend”, and “T3 ADDS, shift or rotate by value” variants: is the second general-purpose source register, encoded in the “Rm” field.

<shift>

Is the type of shift to be applied to the second source register, encoded in “stype”:

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

<amount>

For the “A1 ADD, shift or rotate by value” and “A1 ADDS, shift or rotate by value” variants: is the shift amount, in the range 1 to 31 (when <shift> = LSL or ROR) or 1 to 32 (when <shift> = LSR or ASR) encoded in the “imm5” field as <amount> modulo 32.

For the “T3 ADD, shift or rotate by value” and “T3 ADDS, shift or rotate by value” variants: is the shift amount, in the range 1 to 31 (when <shift> = LSL or ROR) or 1 to 32 (when <shift> = LSR or ASR), encoded in the “imm3:imm2” field as <amount> modulo 32.

<Rdm>

Is the general-purpose destination and second source register, encoded in the “Rdm” field. If omitted, this register is the SP. Arm deprecates using the PC as the destination register, but if the PC is used, the instruction is a branch to the address calculated by the operation. This is a simple branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.

SP

Is the stack pointer.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) shifted = Shift(R[m], shift_t, shift_n, PSTATE.C);
    bits(32) result;
    bits(4) nzcv;
    (result, nzcv) = AddWithCarry(R[13], shifted, '0');
    if d == 15 then
        if setflags then
            ALUExceptionReturn(result);
        else
            ALUWritePC(result);
    else
        R[d] = result;
        if setflags then
            PSTATE.<N,Z,C,V> = nzcv;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.10 ADR

Form PC-relative address adds an immediate value to the PC value to form a PC-relative address, and writes the result to the destination register.

This instruction is used by the alias SUB (immediate, from PC).

This instruction is used by the pseudo-instruction ADD (immediate, to PC).

It has encodings from the following instruction sets: A32 (A1 and A2) and T32 (T1, T2, and T3).

A1

Encoding

```
ADR{<c>}{<q>}  <Rd>, <label>
```

Decode for this encoding

Listing F5-1

```
constant integer d = UInt(Rd);
constant bits(32) imm32 = A32ExpandImm(imm12);
constant boolean add = TRUE;
```

A2

Encoding

```
ADR{<c>}{<q>}  <Rd>, <label>
```

Decode for this encoding

Listing F5-2

```
constant integer d = UInt(Rd);
constant bits(32) imm32 = A32ExpandImm(imm12);
constant boolean add = FALSE;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T1

15			12	11	10	8	7	0
1	0	1	0	0	Rd		imm8	

SP

Encoding

```
ADR{<c>}{<q>}  <Rd>, <label>
```

Decode for this encoding

Listing F5-3

```
constant integer d = UInt(Rd);
constant bits(32) imm32 = ZeroExtend(imm8:'00', 32);
constant boolean add = TRUE;
```

T2

15			13	12	11	10	9	8	7	6	5	4	3				0	15	14	12	11	8	7	0
1	1	1		1	0	i	1	0	1	0	1	0	1	1	1	1		0	imm3		Rd		imm8	

o1 o2 Rn

Encoding

```
ADR{<c>}{<q>}  <Rd>, <label>
```

Decode for this encoding

Listing F5-4

```
constant integer d = UInt(Rd);
constant bits(32) imm32 = ZeroExtend(i:imm3:imm8, 32);
constant boolean add = FALSE;
// Armv8-A removes UNPREDICTABLE for R13
if d == 15 then UNPREDICTABLE; end
```

T3

15				11	10	9	8	7	6	5	4	3				0	15	14	12	11	8	7	0
1	1	1	1	0	i	1	0	0	0	0	0	1	1	1	1		0	imm3		Rd		imm8	

o1 o2 Rn

Encoding

```
ADR{<c>}{<q>}  <Rd>, <label>
```

```
ADR{<c>}.W  <Rd>, <label> // (<Rd>, <label> can be presented in T1)
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for this encoding

Listing F5-5

```
constant integer d = UInt(Rd);
constant bits(32) imm32 = ZeroExtend(i:imm3:imm8, 32);
constant boolean add = TRUE;
// Armv8-A removes UNPREDICTABLE for R13
if d == 15 then UNPREDICTABLE; end
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

For the "A1" and "A2" variants: is the general-purpose destination register, encoded in the "Rd" field. If the PC is used, the instruction is a branch to the address calculated by the operation. This is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.

For the "T1", "T2", and "T3" variants: is the general-purpose destination register, encoded in the "Rd" field.

<label>

For the "A1" and "A2" variants: the label of an instruction or literal data item whose address is to be loaded into `<Rd>`. The assembler calculates the required value of the offset from the `Align(PC, 4)` value of the `ADR` instruction to this label.

If the offset is zero or positive, encoding A1 is used, with `imm32` equal to the offset.

If the offset is negative, encoding A2 is used, with `imm32` equal to the size of the offset. That is, the use of encoding A2 indicates that the required offset is minus the value of `imm32`.

Permitted values of the size of the offset are any of the constants described in Modified immediate constants in A32 instructions.

For the "T1" variant: the label of an instruction or literal data item whose address is to be loaded into `<Rd>`. The assembler calculates the required value of the offset from the `Align(PC, 4)` value of the `ADR` instruction to this label. Permitted values of the size of the offset are multiples of 4 in the range 0 to 1020.

For the "T2" and "T3" variants: the label of an instruction or literal data item whose address is to be loaded into `<Rd>`. The assembler calculates the required value of the offset from the `Align(PC, 4)` value of the `ADR` instruction to this label.

If the offset is zero or positive, encoding T3 is used, with `imm32` equal to the offset.

If the offset is negative, encoding T2 is used, with `imm32` equal to the size of the offset. That is, the use of encoding T2 indicates that the required offset is minus the value of `imm32`.

Permitted values of the size of the offset are 0-4095.

The instruction aliases permit the addition or subtraction of the offset and the immediate offset to be specified separately, including permitting a subtraction of 0 that cannot be specified using the normal syntax. For more information, see Use of labels in UAL instruction syntax.

Alias Conditions

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Alias	Of variant	Is preferred when
ADD (immediate, to PC)		Never
SUB (immediate, from PC)	A2	imm12 == '000000000000'
SUB (immediate, from PC)	T2	i : imm3 : imm8 == '000000000000'

Operation

Listing F5-6

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) result = (if add then (Align(PC32,4) + imm32)
                                else (Align(PC32,4) - imm32));
    if d == 15 then          // Can only occur for A32 encodings
        ALUWritePC(result);
    else
        R[d] = result;
    end
end
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.11 AND, ANDS (immediate)

Bitwise AND (immediate) performs a bitwise AND of a register value and an immediate value, and writes the result to the destination register.

If the destination register is not the PC, the ANDS variant of the instruction updates the condition flags based on the result.

The field descriptions for `<Rd>` identify the encodings where the PC is permitted as the destination register. Arm deprecates any use of these encodings. However, when the destination register is the PC:

- The AND variant of the instruction is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- The ANDS variant of the instruction performs an exception return without the use of the stack. In this case:
  - The PE branches to the address written to the PC, and restores PSTATE from SPSR_<current_mode>.
  - The PE checks SPSR_<current_mode> for an illegal return event. See Illegal return events from AArch32 state.
  - The instruction is UNDEFINED in Hyp mode.
  - The instruction is CONSTRAINED UNPREDICTABLE in User mode and System mode.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding for the AND variant

Applies when `(S == 0)`

```
AND{<c>}{<q>}  {<Rd>, }<Rn>, #<const>
```

Encoding for the ANDS variant

Applies when `(S == 1)`

```
ANDS{<c>}{<q>}  {<Rd>, }<Rn>, #<const>
```

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant boolean setflags = (S == '1');
constant boolean a32 = TRUE;
constant bits(12) imm = imm12;
```

T1

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Encoding for the AND variant

Applies when `(S == 0)`

```
AND{<c>}{<q>}  {<Rd>, }<Rn>, #<const>
```

Encoding for the ANDS variant

Applies when `(S == 1 && Rd != 1111)`

```
ANDS{<c>}{<q>}  {<Rd>, }<Rn>, #<const>
```

Decode for all variants of this encoding

```
if Rd == '1111' && S == '1' then SEE "TST (immediate)";
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant boolean setflags = (S == '1');
constant boolean a32 = FALSE;
constant bits(12) imm = i:imm3:imm8;
// Armv8-A removes UNPREDICTABLE for R13
if (d == 15 && !setflags) || n == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

For the “A1 AND” and “A1 ANDS” variants: is the general-purpose destination register, encoded in the “Rd” field. If omitted, this register is the same as `<Rn>`. Arm deprecates using the PC as the destination register, but if the PC is used:

- For the AND variant, the instruction is a branch to the address calculated by the operation. This is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- For the ANDS variant, the instruction performs an exception return, that restores PSTATE from SPSR_<current_mode>.

For the “T1 AND” and “T1 ANDS” variants: is the general-purpose destination register, encoded in the “Rd” field. If omitted, this register is the same as `<Rn>`.

<Rn>

For the “A1 AND” and “A1 ANDS” variants: is the general-purpose source register, encoded in the “Rn” field. The PC can be used, but this is deprecated.

For the “T1 AND” and “T1 ANDS” variants: is the general-purpose source register, encoded in the “Rn” field.

<const>

For the “A1 AND” and “A1 ANDS” variants: an immediate value. See Modified immediate constants in A32 instructions for the range of values.

For the “T1 AND” and “T1 ANDS” variants: an immediate value. See Modified immediate constants in T32 instructions for the range of values.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    bits(32) imm32;
    bit carry;
    (imm32, carry) = (if a32 then A32ExpandImm_C(imm, PSTATE.C) else T32ExpandImm_C(imm, PSTATE.C));
    constant bits(32) result = R[n] AND imm32;
    if d == 15 then          // Can only occur for A32 encoding
        if setflags then
            ALUExceptionReturn(result);
        else
            ALUWritePC(result);
    else
        R[d] = result;
        if setflags then
            PSTATE.N = result<31>;
            PSTATE.Z = IsZeroBit(result);
            PSTATE.C = carry;
            // PSTATE.V unchanged
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.12 AND, ANDS (register)

Bitwise AND (register) performs a bitwise AND of a register value and an optionally-shifted register value, and writes the result to the destination register.

If the destination register is not the PC, the ANDS variant of the instruction updates the condition flags based on the result.

The field descriptions for `<Rd>` identify the encodings where the PC is permitted as the destination register. Arm deprecates any use of these encodings. However, when the destination register is the PC:

- The AND variant of the instruction is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- The ANDS variant of the instruction performs an exception return without the use of the stack. In this case:
  - The PE branches to the address written to the PC, and restores PSTATE from SPSR_<current_mode>.
  - The PE checks SPSR_<current_mode> for an illegal return event. See Illegal return events from AArch32 state.
  - The instruction is UNDEFINED in Hyp mode.
  - The instruction is CONSTRAINED UNPREDICTABLE in User mode and System mode.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1 and T2).

A1

31–28	27–24	23–21	20	19–16	15–12	11–7	6–5	4	3–0
!=1111	0 0 0 0	0 0 0	S	Rn	Rd	imm5	stype	0	Rm
cond		opc							

Encoding for the AND, rotate right with extend variant

Applies when `(S == 0 && imm5 == 00000 && stype == 11)`

```
AND{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, RRX
```

Encoding for the AND, shift or rotate by value variant

Applies when `(S == 0 && !(imm5 == 00000 && stype == 11))`

```
AND{<c>}{<q>}  {<Rd>, }<Rn>, <Rm> {, <shift> #<amount>}
```

Encoding for the ANDS, rotate right with extend variant

Applies when `(S == 1 && imm5 == 00000 && stype == 11)`

```
ANDS{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, RRX
```

Encoding for the ANDS, shift or rotate by value variant

Applies when `(S == 1 && !(imm5 == 00000 && stype == 11))`

```
ANDS{<c>}{<q>}  {<Rd>, }<Rn>, <Rm> {, <shift> #<amount>}
```

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean setflags = (S == '1');
SRType shift_t;
integer shift_n;
(shift_t, shift_n) = DecodeImmShift(stype, imm5);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T1

15–10	9–6	5–3	2–0
0 1 0 0 0 0	0 0 0 0	Rm	Rdn
	op		

Encoding

```
AND<c>{<q>}  {<Rdn>, }<Rdn>, <Rm> // (InITBlock())
```

```
ANDS{<q>}  {<Rdn>, }<Rdn>, <Rm> // (Outside IT block)
```

Decode for this encoding

```
constant integer d = UInt(Rdn);
constant integer n = UInt(Rdn);
constant integer m = UInt(Rm);
constant boolean setflags = !InITBlock();
constant SRType shift_t = SRType_LSL;
constant integer shift_n = 0;
```

T2

15–9	8–5	4	3–0	15	14–12	11–8	7–6	5–4	3–0
1 1 1 0 1 0 1	0 0 0 0	S	Rn	(0)	imm3	Rd	imm2	stype	Rm
	op1								

Encoding for the AND, rotate right with extend variant

Applies when `(S == 0 && imm3 == 000 && imm2 == 00 && stype == 11)`

```
AND{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, RRX
```

Encoding for the AND, shift or rotate by value variant

Applies when `(S == 0 && !(imm3 == 000 && imm2 == 00 && stype == 11))`

```
AND{<c>}{<q>}  {<Rd>, }<Rn>, <Rm> {, <shift> #<amount>}
```

```
AND<c>.W  {<Rd>, }<Rn>, <Rm> // (Inside IT block, and <Rd>, <Rn>, <Rm> can be represented in T1)
```

Encoding for the ANDS, rotate right with extend variant

Applies when `(S == 1 && imm3 == 000 && Rd != 1111 && imm2 == 00 && stype == 11)`

```
ANDS{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, RRX
```

Encoding for the ANDS, shift or rotate by value variant

Applies when `(S == 1 && Rd != 1111 && !(imm3 == 000 && imm2 == 00 && stype == 11))`

```
ANDS{<c>}{<q>}  {<Rd>, }<Rn>, <Rm> {, <shift> #<amount>}
```

```
ANDS.W  {<Rd>, }<Rn>, <Rm> // (Outside IT block, and <Rd>, <Rn>, <Rm> can be represented in T1)
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for all variants of this encoding

```
if Rd == '1111' && S == '1' then SEE "TST (register)";
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean setflags = (S == '1');
SRType shift_t;
integer shift_n;
(shift_t, shift_n) = DecodeImmShift(stype, imm3:imm2);
// Armv8-A removes UNPREDICTABLE for R13
if (d == 15 && !setflags) || n == 15 || m == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

For the “A1 AND, rotate right with extend”, “A1 AND, shift or rotate by value”, “A1 ANDS, rotate right with extend”, and “A1 ANDS, shift or rotate by value” variants: is the general-purpose destination register, encoded in the “Rd” field. If omitted, this register is the same as `<Rn>`. Arm deprecates using the PC as the destination register, but if the PC is used:

- For the AND variant, the instruction is a branch to the address calculated by the operation. This is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- For the ANDS variant, the instruction performs an exception return, that restores PSTATE from SPSR_<current_mode>.

For the “T2 AND, rotate right with extend”, “T2 AND, shift or rotate by value”, “T2 ANDS, rotate right with extend”, and “T2 ANDS, shift or rotate by value” variants: is the general-purpose destination register, encoded in the “Rd” field. If omitted, this register is the same as `<Rn>`.

<Rn>

For the “A1 AND, rotate right with extend”, “A1 AND, shift or rotate by value”, “A1 ANDS, rotate right with extend”, and “A1 ANDS, shift or rotate by value” variants: is the first general-purpose source register, encoded in the “Rn” field. The PC can be used, but this is deprecated.

For the “T2 AND, rotate right with extend”, “T2 AND, shift or rotate by value”, “T2 ANDS, rotate right with extend”, and “T2 ANDS, shift or rotate by value” variants: is the first general-purpose source register, encoded in the “Rn” field.

<Rm>

For the “A1 AND, rotate right with extend”, “A1 AND, shift or rotate by value”, “A1 ANDS, rotate right with extend”, and “A1 ANDS, shift or rotate by value” variants: is the second general-purpose source register, encoded in the “Rm” field. The PC can be used, but this is deprecated.

For the “T1”, “T2 AND, rotate right with extend”, “T2 AND, shift or rotate by value”, “T2 ANDS, rotate right with extend”, and “T2 ANDS, shift or rotate by value” variants: is the second general-purpose source register, encoded in the “Rm” field.

<shift>

Is the type of shift to be applied to the second source register, encoded in “stype”:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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

<amount>

For the “A1 AND, shift or rotate by value” and “A1 ANDS, shift or rotate by value” variants: is the shift amount, in the range 1 to 31 (when <shift> = LSL or ROR) or 1 to 32 (when <shift> = LSR or ASR) encoded in the “imm5” field as <amount> modulo 32.

For the “T2 AND, shift or rotate by value” and “T2 ANDS, shift or rotate by value” variants: is the shift amount, in the range 1 to 31 (when <shift> = LSL or ROR) or 1 to 32 (when <shift> = LSR or ASR), encoded in the “imm3:imm2” field as <amount> modulo 32.

<Rdn>

Is the first general-purpose source register and the destination register, encoded in the “Rdn” field.

In T32 assembly:

- Outside an IT block, if `ANDS <Rd>, <Rn>, <Rd>` has `<Rd>` and `<Rn>` both in the range R0-R7, it is assembled using encoding T1 as though `ANDS <Rd>, <Rn>` had been written.
- Inside an IT block, if `AND<c> <Rd>, <Rn>, <Rd>` has `<Rd>` and `<Rn>` both in the range R0-R7, it is assembled using encoding T1 as though `AND<c> <Rd>, <Rn>` had been written.

To prevent either of these happening, use the `.W` qualifier.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    bits(32) shifted;
    bit carry;
    (shifted, carry) = Shift_C(R[m], shift_t, shift_n, PSTATE.C);
    constant bits(32) result = R[n] AND shifted;
    if d == 15 then          // Can only occur for A32 encoding
        if setflags then
            ALUExceptionReturn(result);
        else
            ALUWritePC(result);
    else
        R[d] = result;
        if setflags then
            PSTATE.N = result<31>;
            PSTATE.Z = IsZeroBit(result);
            PSTATE.C = carry;
            // PSTATE.V unchanged
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.13 AND, ANDS (register-shifted register)

Bitwise AND (register-shifted register) performs a bitwise AND of a register value and a register-shifted register value. It writes the result to the destination register, and can optionally update the condition flags based on the result.

31–28	27–24	23–21	20	19–16	15–12	11–8	7	6–5	4	3–0
!=1111 (cond)	0 0 0 0	0 0 0 (opc)	S	Rn	Rd	Rs	0	stype	1	Rm

Encoding for the Flag setting variant

Applies when `(S == 1)`

```
ANDS{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, <shift> <Rs>
```

Encoding for the Not flag setting variant

Applies when `(S == 0)`

```
AND{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, <shift> <Rs>
```

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant integer s = UInt(Rs);
constant boolean setflags = (S == '1');
constant SRType shift_t = DecodeRegShift(stype);
if d == 15 || n == 15 || m == 15 || s == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

Is the general-purpose destination register, encoded in the “Rd” field.

<Rn>

Is the first general-purpose source register, encoded in the “Rn” field.

<Rm>

Is the second general-purpose source register, encoded in the “Rm” field.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

<shift>

Is the type of shift to be applied to the second source register, encoded in “stype”:

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

<Rs>

Is the third general-purpose source register holding a shift amount in its bottom 8 bits, encoded in the “Rs” field.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant integer shift_n = UInt(R[s]<7:0>);
    bits(32) shifted;
    bit carry;
    (shifted, carry) = Shift_C(R[m], shift_t, shift_n, PSTATE.C);
    constant bits(32) result = R[n] AND shifted;
    R[d] = result;
    if setflags then
        PSTATE.N = result<31>;
        PSTATE.Z = IsZeroBit(result);
        PSTATE.C = carry;
        // PSTATE.V unchanged
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.14 ASR (immediate)

Arithmetic Shift Right (immediate) shifts a register value right by an immediate number of bits, shifting in copies of its sign bit, and writes the result to the destination register.

This is an alias of MOV, MOVS (register). This means:

- The encodings in this description are named to match the encodings of MOV, MOVS (register).
- The description of MOV, MOVS (register) gives the operational pseudocode, any CONSTRAINED UNPREDICTABLE behavior, and any operational information for this instruction.

It has encodings from the following instruction sets: A32 (A1) and T32 (T2 and T3).

A1

Encoding for the MOV, shift or rotate by value variant

```
ASR{<c>}{<q>}  {<Rd>, }<Rm>, #<imm>
```

is equivalent to

```
MOV{<c>}{<q>}  <Rd>, <Rm>, ASR #<imm>
```

and is always the preferred disassembly.

T2

Encoding for the T2 variant

```
ASR<c>{<q>}  {<Rd>, }<Rm>, #<imm>
```

is equivalent to

```
MOV<c>{<q>}  <Rd>, <Rm>, ASR #<imm>
```

and is the preferred disassembly when `InITBlock()`.

T3

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Encoding for the MOV, shift or rotate by value variant

```
ASR{<c>}{<q>}  {<Rd>, }<Rm>, #<imm>
```

```
ASR<c>.W  {<Rd>, }<Rm>, #<imm> // (!InITBlock() && UInt(Rd) < 8 && UInt(Rm) < 8)
```

is equivalent to

```
MOV{<c>}{<q>}  <Rd>, <Rm>, ASR #<imm>
```

and is always the preferred disassembly.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

For the “A1 MOV, shift or rotate by value” variant: is the general-purpose destination register, encoded in the “Rd” field. Arm deprecates using the PC as the destination register, but if the PC is used, the instruction is a branch to the address calculated by the operation. This is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.

For the “T2” and “T3 MOV, shift or rotate by value” variants: is the general-purpose destination register, encoded in the “Rd” field.

<Rm>

For the “A1 MOV, shift or rotate by value” variant: is the general-purpose source register, encoded in the “Rm” field. The PC can be used, but this is deprecated.

For the “T2” and “T3 MOV, shift or rotate by value” variants: is the general-purpose source register, encoded in the “Rm” field.

<imm>

For the “A1 MOV, shift or rotate by value” and “T2” variants: is the shift amount, in the range 1 to 32, encoded in the “imm5” field as <imm> modulo 32.

For the “T3 MOV, shift or rotate by value” variant: is the shift amount, in the range 1 to 32, encoded in the “imm3:imm2” field as <imm> modulo 32.

Operation

The description of MOV, MOVS (register) gives the operational pseudocode for this instruction.

Operational Information

The description of MOV, MOVS (register) gives the operational information for this instruction.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.15 ASR (register)

Arithmetic Shift Right (register) shifts a register value right by a variable number of bits, shifting in copies of its sign bit, and writes the result to the destination register. The variable number of bits is read from the bottom byte of a register.

This is an alias of MOV, MOVS (register-shifted register). This means:

- The encodings in this description are named to match the encodings of MOV, MOVS (register-shifted register).
- The description of MOV, MOVS (register-shifted register) gives the operational pseudocode, any CONSTRAINED UNPREDICTABLE behavior, and any operational information for this instruction.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1 and T2).

A1

Encoding for the Not flag setting variant

```
ASR{<c>}{<q>}  {<Rd>, }<Rm>, <Rs>
```

is equivalent to

```
MOV{<c>}{<q>}  <Rd>, <Rm>, ASR <Rs>
```

and is always the preferred disassembly.

T1

Encoding for the Arithmetic shift right variant

```
ASR<c>{<q>}  {<Rdm>, }<Rdm>, <Rs>
```

is equivalent to

```
MOV<c>{<q>}  <Rdm>, <Rdm>, ASR <Rs>
```

and is the preferred disassembly when `InITBlock()`.

T2

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Encoding for the Not flag setting variant

```
ASR{<c>}{<q>}  {<Rd>, }<Rm>, <Rs>
```

```
ASR<c>.W  {<Rd>, }<Rm>, <Rs> // (InITBlock() && (UInt(Rd) < 8 && Rd == Rm && UInt(Rs) < 8))
```

is equivalent to

```
MOV{<c>}{<q>}  <Rd>, <Rm>, ASR <Rs>
```

and is always the preferred disassembly.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

Is the general-purpose destination register, encoded in the “Rd” field.

<Rm>

Is the first general-purpose source register, encoded in the “Rm” field.

<Rs>

Is the second general-purpose source register holding a shift amount in its bottom 8 bits, encoded in the “Rs” field.

<Rdm>

Is the first general-purpose source register and the destination register, encoded in the “Rdm” field.

Operation

The description of MOV, MOVS (register-shifted register) gives the operational pseudocode for this instruction.

Operational Information

The description of MOV, MOVS (register-shifted register) gives the operational information for this instruction.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.16 ASRS (immediate)

Arithmetic Shift Right, setting flags (immediate) shifts a register value right by an immediate number of bits, shifting in copies of its sign bit, and writes the result to the destination register.

If the destination register is not the PC, this instruction updates the condition flags based on the result.

The field descriptions for `<Rd>` identify the encodings where the PC is permitted as the destination register. Arm deprecates any use of these encodings. However, when the destination register is the PC:

- The PE branches to the address written to the PC, and restores PSTATE from SPSR_<current_mode>.
- The PE checks SPSR_<current_mode> for an illegal return event. See Illegal return events from AArch32 state.
- The instruction is UNDEFINED in Hyp mode.
- The instruction is CONSTRAINED UNPREDICTABLE in User mode and System mode.

This is an alias of MOV, MOVS (register). This means:

- The encodings in this description are named to match the encodings of MOV, MOVS (register).
- The description of MOV, MOVS (register) gives the operational pseudocode, any CONSTRAINED UNPREDICTABLE behavior, and any operational information for this instruction.

It has encodings from the following instruction sets: A32 (A1) and T32 (T2 and T3).

A1

Encoding for the MOVS, shift or rotate by value variant

```
ASRS{<c>}{<q>}  {<Rd>, }<Rm>, #<imm>
```

is equivalent to

```
MOVS{<c>}{<q>}  <Rd>, <Rm>, ASR #<imm>
```

and is always the preferred disassembly.

T2

Encoding for the T2 variant

```
ASRS{<q>}  {<Rd>, }<Rm>, #<imm>
```

is equivalent to

```
MOVS<c>{<q>}  <Rd>, <Rm>, ASR #<imm>
```

and is the preferred disassembly when `!InITBlock()`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T3

15 13	12 9	8 5	4	3 0	15	14 12	11 8	7 6	5 4	3 0
1 1 1	0 1 0 1	0 0 1 0	1	1 1 1 1	(0)	imm3	Rd	imm2	1 0	Rm
		op1	S	Rn					stype	

Encoding for the MOVS, shift or rotate by value variant

```
ASRS{<c>}{<q>}  {<Rd>, }<Rm>, #<imm>
```

```
ASRS.W  {<Rd>, }<Rm>, #<imm> // (!InITBlock() && UInt(Rd) < 8 && UInt(Rm) < 8)
```

is equivalent to

```
MOVS{<c>}{<q>}  <Rd>, <Rm>, ASR #<imm>
```

and is always the preferred disassembly.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

For the “A1 MOVS, shift or rotate by value” variant: is the general-purpose destination register, encoded in the “Rd” field. Arm deprecates using the PC as the destination register, but if the PC is used, the instruction performs an exception return, that restores PSTATE from SPSR_<current_mode>.

For the “T2” and “T3 MOVS, shift or rotate by value” variants: is the general-purpose destination register, encoded in the “Rd” field.

<Rm>

For the “A1 MOVS, shift or rotate by value” variant: is the general-purpose source register, encoded in the “Rm” field. The PC can be used, but this is deprecated.

For the “T2” and “T3 MOVS, shift or rotate by value” variants: is the general-purpose source register, encoded in the “Rm” field.

<imm>

For the “A1 MOVS, shift or rotate by value” and “T2” variants: is the shift amount, in the range 1 to 32, encoded in the “imm5” field as <imm> modulo 32.

For the “T3 MOVS, shift or rotate by value” variant: is the shift amount, in the range 1 to 32, encoded in the “imm3:imm2” field as <imm> modulo 32.

Operation

The description of MOV, MOVS (register) gives the operational pseudocode for this instruction.

Operational Information

The description of MOV, MOVS (register) gives the operational information for this instruction.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.17 ASRS (register)

Arithmetic Shift Right, setting flags (register) shifts a register value right by a variable number of bits, shifting in copies of its sign bit, writes the result to the destination register, and updates the condition flags based on the result. The variable number of bits is read from the bottom byte of a register.

This is an alias of MOV, MOVS (register-shifted register). This means:

- The encodings in this description are named to match the encodings of MOV, MOVS (register-shifted register).
- The description of MOV, MOVS (register-shifted register) gives the operational pseudocode, any CONSTRAINED UNPREDICTABLE behavior, and any operational information for this instruction.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1 and T2).

A1

Encoding for the Flag setting variant

```
ASRS{<c>}{<q>}  {<Rd>, }<Rm>, <Rs>
```

is equivalent to

```
MOVS{<c>}{<q>}  <Rd>, <Rm>, ASR <Rs>
```

and is always the preferred disassembly.

T1

Encoding for the Arithmetic shift right variant

```
ASRS{<q>}  {<Rdm>, }<Rdm>, <Rs>
```

is equivalent to

```
MOVS{<q>}  <Rdm>, <Rdm>, ASR <Rs>
```

and is the preferred disassembly when `!InITBlock()`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T2

15–13	12–8	7	6–5	4	3–0	15–12	11–8	7–4	3–0
1 1 1	1 1 0 1 0	0	1 0	1	Rm	1 1 1 1	Rd	0 0 0 0	Rs
			stype	S					

Encoding for the Flag setting variant

```
ASRS{<c>}{<q>}  {<Rd>, }<Rm>, <Rs>
```

```
ASRS.W  {<Rd>, }<Rm>, <Rs> // (!InITBlock() && (UInt(Rd) < 8 && Rd == Rm && UInt(Rs) < 8))
```

is equivalent to

```
MOVS{<c>}{<q>}  <Rd>, <Rm>, ASR <Rs>
```

and is always the preferred disassembly.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

Is the general-purpose destination register, encoded in the “Rd” field.

<Rm>

Is the first general-purpose source register, encoded in the “Rm” field.

<Rs>

Is the second general-purpose source register holding a shift amount in its bottom 8 bits, encoded in the “Rs” field.

<Rdm>

Is the first general-purpose source register and the destination register, encoded in the “Rdm” field.

Operation

The description of MOV, MOVS (register-shifted register) gives the operational pseudocode for this instruction.

Operational Information

The description of MOV, MOVS (register-shifted register) gives the operational information for this instruction.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.18 B

Branch causes a branch to a target address.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1, T2, T3, and T4).

A1

Encoding

```
B{<c>}{<q>}  <label>
```

Decode for this encoding

```
constant bits(32) imm32 = SignExtend(imm24:'00', 32);
```

T1

Encoding

```
B<c>{<q>}  <label> // (Not permitted in IT block)
```

Decode for this encoding

```
if cond == '1110' then SEE "UDF";
if cond == '1111' then SEE "SVC";
constant bits(32) imm32 = SignExtend(imm8:'0', 32);
if InITBlock() then UNPREDICTABLE;
```

T2

Encoding

```
B{<c>}{<q>}  <label> // (Outside or last in IT block)
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for this encoding

```
constant bits(32) imm32 = SignExtend(imm11:'0', 32);
if InITBlock() && !LastInITBlock() then UNPREDICTABLE;
```

T3

15 … 11	10	9 … 6	5 … 0	15	14	13	12	11	10 … 0
1 1 1 1 0	S	!=111x (cond)	imm6	1	0	J1	0	J2	imm11

Encoding

```
B<c>.W  <label> // (Not permitted in IT block, and <label> can be represented in T1)
```

```
B<c>{<q>}  <label> // (Not permitted in IT block)
```

Decode for this encoding

```
if cond<3:1> == '111' then SEE "Related encodings";
constant bits(32) imm32 = SignExtend(S:J2:J1:imm6:imm11:'0', 32);
if InITBlock() then UNPREDICTABLE;
```

T4

15 … 11	10	9 … 0	15	14	13	12	11	10 … 0
1 1 1 1 0	S	imm10	1	0	J1	1	J2	imm11

Encoding

```
B{<c>}{<q>}  <label>
```

```
B{<c>}.W  <label> // (<label> can be represented in T2)
```

Decode for this encoding

```
constant bit I1 = NOT(J1 EOR S);
constant bit I2 = NOT(J2 EOR S);
constant bits(32) imm32 = SignExtend(S:I1:I2:imm10:imm11:'0', 32);
if InITBlock() && !LastInITBlock() then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Related encodings: Branches and miscellaneous control.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

For the “A1”, “T2”, and “T4” variants: see Standard assembler syntax fields.

For the “T1” and “T3” variants: see Standard assembler syntax fields. Must not be `AL` or omitted.

<q>

See Standard assembler syntax fields.

<label>

For the “A1” variant: the label of the instruction that is to be branched to. The assembler calculates the required value of the offset from the PC value of the `B` instruction to this label, then selects an encoding that sets `imm32` to that offset.

Permitted offsets are multiples of 4 in the range –33554432 to 33554428.

For the “T1” variant: the label of the instruction that is to be branched to. The assembler calculates the required value of the offset from the PC value of the `B` instruction to this label, then selects an encoding that sets `imm32` to that offset. Permitted offsets are even numbers in the range –256 to 254.

For the “T2” variant: the label of the instruction that is to be branched to. The assembler calculates the required value of the offset from the PC value of the `B` instruction to this label, then selects an encoding that sets `imm32` to that offset. Permitted offsets are even numbers in the range –2048 to 2046.

For the “T3” variant: the label of the instruction that is to be branched to. The assembler calculates the required value of the offset from the PC value of the `B` instruction to this label, then selects an encoding that sets `imm32` to that offset.

Permitted offsets are even numbers in the range –1048576 to 1048574.

For the “T4” variant: the label of the instruction that is to be branched to. The assembler calculates the required value of the offset from the PC value of the `B` instruction to this label, then selects an encoding that sets `imm32` to that offset.

Permitted offsets are even numbers in the range –16777216 to 16777214.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    BranchWritePC(PC32 + imm32, BranchType_DIR);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.19 BFC

Bit Field Clear clears any number of adjacent bits at any position in a register, without affecting the other bits in the register.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

31–28	27–25	24–21	20–16	15–12	11–7	6–5	4	3–0
!=1111 (cond)	0 1 1	1 1 1 0	msb	Rd	lsb	0 0	1	1 1 1 1 (Rn)

Encoding

```
BFC{<c>}{<q>}  <Rd>, #<lsb>, #<width>
```

Decode for this encoding

```
constant integer d = UInt(Rd);
constant integer msbit = UInt(msb);
constant integer lsbit = UInt(lsb);
if d == 15 then UNPREDICTABLE;
if msbit < lsbit then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `msbit < lsbit`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The value in the destination register is UNKNOWN.

T1

15–13	12–11	10	9	8	7–5	4	3–0	15	14–12	11–8	7–6	5	4–0
1 1 1	1 0	(0)	1	1	0 1 1 (op1)	0	1 1 1 1 (Rn)	0	imm3	Rd	imm2	(0)	msb

Encoding

```
BFC{<c>}{<q>}  <Rd>, #<lsb>, #<width>
```

Decode for this encoding

```
constant integer d = UInt(Rd);
constant integer msbit = UInt(msb);
constant integer lsbit = UInt(imm3:imm2);
// Armv8-A removes UNPREDICTABLE for R13
if d == 15 then UNPREDICTABLE;
if msbit < lsbit then UNPREDICTABLE;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CONSTRAINED UNPREDICTABLE behavior

If `msbit < lsbit`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The value in the destination register is UNKNOWN.

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

Is the general-purpose destination register, encoded in the “Rd” field.

<lsb>

For the “A1” variant: is the least significant bit to be cleared, in the range 0 to 31, encoded in the “lsb” field.

For the “T1” variant: is the least significant bit that is to be cleared, in the range 0 to 31, encoded in the “imm3:imm2” field.

<width>

Is the number of bits to be cleared, in the range 1 to 32-<lsb>, encoded in the “msb” field as <lsb>+<width>-1.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    R[d]<msbit:lsbit> = Replicate('0', (msbit-lsbit)+1);
    // Other bits of R[d] are unchanged
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.20 BFI

Bit Field Insert copies any number of low order bits from a register into the same number of adjacent bits at any position in the destination register.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

31–28	27–25	24–21	20–16	15–12	11–7	6–5	4	3–0
!=1111 (cond)	0 1 1	1 1 1 0	msb	Rd	lsb	0 0	1	!=1111 (Rn)

Encoding

```
BFI{<c>}{<q>}  <Rd>, <Rn>, #<lsb>, #<width>
```

Decode for this encoding

```
if Rn == '1111' then SEE "BFC";
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer msbit = UInt(msb);
constant integer lsbit = UInt(lsb);
if d == 15 then UNPREDICTABLE;
if msbit < lsbit then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `msbit < lsbit`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The value in the destination register is UNKNOWN.

T1

15–13	12–11	10	9	8	7–5	4	3–0	15	14–12	11–8	7–6	5	4–0
1 1 1	1 0	(0)	1	1	0 1 1 (op1)	0	!=1111 (Rn)	0	imm3	Rd	imm2	(0)	msb

Encoding

```
BFI{<c>}{<q>}  <Rd>, <Rn>, #<lsb>, #<width>
```

Decode for this encoding

```
if Rn == '1111' then SEE "BFC";
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer msbit = UInt(msb);
constant integer lsbit = UInt(imm3:imm2);
// Armv8-A removes UNPREDICTABLE for R13
if d == 15 then UNPREDICTABLE;
if msbit < lsbit then UNPREDICTABLE;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CONSTRAINED UNPREDICTABLE behavior

If `msbit < lsbit`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The value in the destination register is UNKNOWN.

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

Is the general-purpose destination register, encoded in the “Rd” field.

<Rn>

Is the general-purpose source register, encoded in the “Rn” field.

<lsb>

For the “A1” variant: is the least significant destination bit, in the range 0 to 31, encoded in the “lsb” field.

For the “T1” variant: is the least significant destination bit, in the range 0 to 31, encoded in the “imm3:imm2” field.

<width>

Is the number of bits to be copied, in the range 1 to 32-<lsb>, encoded in the “msb” field as <lsb>+<width>-1.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    R[d]<msbit:lsbit> = R[n]<(msbit-lsbit):0>;
    // Other bits of R[d] are unchanged
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.21 BIC, BICS (immediate)

Bitwise Bit Clear (immediate) performs a bitwise AND of a register value and the complement of an immediate value, and writes the result to the destination register.

If the destination register is not the PC, the BICS variant of the instruction updates the condition flags based on the result.

The field descriptions for `<Rd>` identify the encodings where the PC is permitted as the destination register. Arm deprecates any use of these encodings. However, when the destination register is the PC:

- The BIC variant of the instruction is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- The BICS variant of the instruction performs an exception return without the use of the stack. In this case:
  - The PE branches to the address written to the PC, and restores PSTATE from SPSR_<current_mode>.
  - The PE checks SPSR_<current_mode> for an illegal return event. See Illegal return events from AArch32 state.
  - The instruction is UNDEFINED in Hyp mode.
  - The instruction is CONSTRAINED UNPREDICTABLE in User mode and System mode.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding for the BIC variant

Applies when `(S == 0)`

```
BIC{<c>}{<q>}  {<Rd>, }<Rn>, #<const>
```

Encoding for the BICS variant

Applies when `(S == 1)`

```
BICS{<c>}{<q>}  {<Rd>, }<Rn>, #<const>
```

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant boolean setflags = (S == '1');
constant boolean a32 = TRUE;
constant bits(12) imm = imm12;
```

T1

Encoding for the BIC variant

Applies when `(S == 0)`

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
BIC{<c>}{<q>}  {<Rd>, }<Rn>, #<const>
```

Encoding for the BICS variant

Applies when `(S == 1)`

```
BICS{<c>}{<q>}  {<Rd>, }<Rn>, #<const>
```

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant boolean setflags = (S == '1');
constant boolean a32 = FALSE;
constant bits(12) imm = i:imm3:imm8;
// Armv8-A removes UNPREDICTABLE for R13
if d == 15 || n == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

For the “A1 BIC” and “A1 BICS” variants: is the general-purpose destination register, encoded in the “Rd” field. If omitted, this register is the same as `<Rn>`. Arm deprecates using the PC as the destination register, but if the PC is used:

- For the BIC variant, the instruction is a branch to the address calculated by the operation. This is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- For the BICS variant, the instruction performs an exception return, that restores PSTATE from SPSR_<current_mode>.

For the “T1 BIC” and “T1 BICS” variants: is the general-purpose destination register, encoded in the “Rd” field. If omitted, this register is the same as `<Rn>`.

<Rn>

For the “A1 BIC” and “A1 BICS” variants: is the general-purpose source register, encoded in the “Rn” field. The PC can be used, but this is deprecated.

For the “T1 BIC” and “T1 BICS” variants: is the general-purpose source register, encoded in the “Rn” field.

<const>

For the “A1 BIC” and “A1 BICS” variants: an immediate value. See Modified immediate constants in A32 instructions for the range of values.

For the “T1 BIC” and “T1 BICS” variants: an immediate value. See Modified immediate constants in T32 instructions for the range of values.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    bits(32) imm32;
    bit carry;
    (imm32, carry) = if a32 then A32ExpandImm_C(imm, PSTATE.C) else T32ExpandImm_C(imm, PSTATE.C);
    constant bits(32) result = R[n] AND NOT(imm32);
    if d == 15 then          // Can only occur for A32 encoding
        if setflags then
            ALUExceptionReturn(result);
        else
            ALUWritePC(result);
    else
        R[d] = result;
        if setflags then
            PSTATE.N = result<31>;
            PSTATE.Z = IsZeroBit(result);
            PSTATE.C = carry;
            // PSTATE.V unchanged
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.22 BIC, BICS (register)

Bitwise Bit Clear (register) performs a bitwise AND of a register value and the complement of an optionally-shifted register value, and writes the result to the destination register.

If the destination register is not the PC, the BICS variant of the instruction updates the condition flags based on the result.

The field descriptions for `<Rd>` identify the encodings where the PC is permitted as the destination register. Arm deprecates any use of these encodings. However, when the destination register is the PC:

- The BIC variant of the instruction is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- The BICS variant of the instruction performs an exception return without the use of the stack. In this case:
  - The PE branches to the address written to the PC, and restores PSTATE from SPSR_<current_mode>.
  - The PE checks SPSR_<current_mode> for an illegal return event. See Illegal return events from AArch32 state.
  - The instruction is UNDEFINED in Hyp mode.
  - The instruction is CONSTRAINED UNPREDICTABLE in User mode and System mode.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1 and T2).

A1

31–28	27–23	22–21	20	19–16	15–12	11–7	6–5	4	3–0
!=1111	0 0 0 1 1	1 0	S	Rn	Rd	imm5	stype	0	Rm
cond		opc							

Encoding for the BIC, rotate right with extend variant

Applies when `(S == 0 && imm5 == 00000 && stype == 11)`

```
BIC{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, RRX
```

Encoding for the BIC, shift or rotate by value variant

Applies when `(S == 0 && !(imm5 == 00000 && stype == 11))`

```
BIC{<c>}{<q>}  {<Rd>, }<Rn>, <Rm> {, <shift> #<amount>}
```

Encoding for the BICS, rotate right with extend variant

Applies when `(S == 1 && imm5 == 00000 && stype == 11)`

```
BICS{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, RRX
```

Encoding for the BICS, shift or rotate by value variant

Applies when `(S == 1 && !(imm5 == 00000 && stype == 11))`

```
BICS{<c>}{<q>}  {<Rd>, }<Rn>, <Rm> {, <shift> #<amount>}
```

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean setflags = (S == '1');
SRType shift_t;
integer shift_n;
(shift_t, shift_n) = DecodeImmShift(stype, imm5);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T1

15–10	9–6	5–3	2–0
0 1 0 0 0 0	1 1 1 0	Rm	Rdn
	op		

Encoding

```
BIC<c>{<q>}  {<Rdn>, }<Rdn>, <Rm> // (InITBlock())
```

```
BICS{<q>}  {<Rdn>, }<Rdn>, <Rm> // (Outside IT block)
```

Decode for this encoding

```
constant integer d = UInt(Rdn);
constant integer n = UInt(Rdn);
constant integer m = UInt(Rm);
constant boolean setflags = !InITBlock();
constant SRType shift_t = SRType_LSL;
constant integer shift_n = 0;
```

T2

15–9	8–5	4	3–0	15	14–12	11–8	7–6	5–4	3–0
1 1 1 0 1 0 1	0 0 0 1	S	Rn	(0)	imm3	Rd	imm2	stype	Rm
	op1								

Encoding for the BIC, rotate right with extend variant

Applies when `(S == 0 && imm3 == 000 && imm2 == 00 && stype == 11)`

```
BIC{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, RRX
```

Encoding for the BIC, shift or rotate by value variant

Applies when `(S == 0 && !(imm3 == 000 && imm2 == 00 && stype == 11))`

```
BIC{<c>}{<q>}  {<Rd>, }<Rn>, <Rm> {, <shift> #<amount>}
```

```
BIC<c>.W  {<Rd>, }<Rn>, <Rm> // (Inside IT block, and <Rd>, <Rn>, <Rm> can be represented in T1)
```

Encoding for the BICS, rotate right with extend variant

Applies when `(S == 1 && imm3 == 000 && imm2 == 00 && stype == 11)`

```
BICS{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, RRX
```

Encoding for the BICS, shift or rotate by value variant

Applies when `(S == 1 && !(imm3 == 000 && imm2 == 00 && stype == 11))`

```
BICS{<c>}{<q>}  {<Rd>, }<Rn>, <Rm> {, <shift> #<amount>}
```

```
BICS.W  {<Rd>, }<Rn>, <Rm> // (Outside IT block, and <Rd>, <Rn>, <Rm> can be represented in T1)
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean setflags = (S == '1');
SRType shift_t;
integer shift_n;
(shift_t, shift_n) = DecodeImmShift(stype, imm3:imm2);
// Armv8-A removes UNPREDICTABLE for R13
if d == 15 || n == 15 || m == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

For the “A1 BIC, rotate right with extend”, “A1 BIC, shift or rotate by value”, “A1 BICS, rotate right with extend”, and “A1 BICS, shift or rotate by value” variants: is the general-purpose destination register, encoded in the “Rd” field. If omitted, this register is the same as `<Rn>`. Arm deprecates using the PC as the destination register, but if the PC is used:

- For the BIC variant, the instruction is a branch to the address calculated by the operation. This is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- For the BICS variant, the instruction performs an exception return, that restores PSTATE from SPSR_<current_mode>.

For the “T2 BIC, rotate right with extend”, “T2 BIC, shift or rotate by value”, “T2 BICS, rotate right with extend”, and “T2 BICS, shift or rotate by value” variants: is the general-purpose destination register, encoded in the “Rd” field. If omitted, this register is the same as `<Rn>`.

<Rn>

For the “A1 BIC, rotate right with extend”, “A1 BIC, shift or rotate by value”, “A1 BICS, rotate right with extend”, and “A1 BICS, shift or rotate by value” variants: is the first general-purpose source register, encoded in the “Rn” field. The PC can be used, but this is deprecated.

For the “T2 BIC, rotate right with extend”, “T2 BIC, shift or rotate by value”, “T2 BICS, rotate right with extend”, and “T2 BICS, shift or rotate by value” variants: is the first general-purpose source register, encoded in the “Rn” field.

<Rm>

For the “A1 BIC, rotate right with extend”, “A1 BIC, shift or rotate by value”, “A1 BICS, rotate right with extend”, and “A1 BICS, shift or rotate by value” variants: is the second general-purpose source register, encoded in the “Rm” field. The PC can be used, but this is deprecated.

For the “T1”, “T2 BIC, rotate right with extend”, “T2 BIC, shift or rotate by value”, “T2 BICS, rotate right with extend”, and “T2 BICS, shift or rotate by value” variants: is the second general-purpose source register, encoded in the “Rm” field.

<shift>

Is the type of shift to be applied to the second source register, encoded in “stype”:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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

<amount>

For the “A1 BIC, shift or rotate by value” and “A1 BICS, shift or rotate by value” variants: is the shift amount, in the range 1 to 31 (when <shift> = LSL or ROR) or 1 to 32 (when <shift> = LSR or ASR), encoded in the “imm5” field as <amount> modulo 32.

For the “T2 BIC, shift or rotate by value” and “T2 BICS, shift or rotate by value” variants: is the shift amount, in the range 1 to 31 (when <shift> = LSL or ROR) or 1 to 32 (when <shift> = LSR or ASR), encoded in the “imm3:imm2” field as <amount> modulo 32.

<Rdn>

Is the first general-purpose source register and the destination register, encoded in the “Rdn” field.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    bits(32) shifted;
    bit carry;
    (shifted, carry) = Shift_C(R[m], shift_t, shift_n, PSTATE.C);
    constant bits(32) result = R[n] AND NOT(shifted);
    if d == 15 then          // Can only occur for A32 encoding
        if setflags then
            ALUExceptionReturn(result);
        else
            ALUWritePC(result);
    else
        R[d] = result;
        if setflags then
            PSTATE.N = result<31>;
            PSTATE.Z = IsZeroBit(result);
            PSTATE.C = carry;
            // PSTATE.V unchanged
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.23 BIC, BICS (register-shifted register)

Bitwise Bit Clear (register-shifted register) performs a bitwise AND of a register value and the complement of a register-shifted register value. It writes the result to the destination register, and can optionally update the condition flags based on the result.

31–28	27–23	22–21	20	19–16	15–12	11–8	7	6–5	4	3–0
!=1111 (cond)	0 0 0 1 1	1 0 (opc)	S	Rn	Rd	Rs	0	stype	1	Rm

Encoding for the Flag setting variant

Applies when `(S == 1)`

```
BICS{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, <shift> <Rs>
```

Encoding for the Not flag setting variant

Applies when `(S == 0)`

```
BIC{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, <shift> <Rs>
```

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant integer s = UInt(Rs);
constant boolean setflags = (S == '1');
constant SRType shift_t = DecodeRegShift(stype);
if d == 15 || n == 15 || m == 15 || s == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

Is the general-purpose destination register, encoded in the “Rd” field.

<Rn>

Is the first general-purpose source register, encoded in the “Rn” field.

<Rm>

Is the second general-purpose source register, encoded in the “Rm” field.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

<shift>

Is the type of shift to be applied to the second source register, encoded in “stype”:

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

<Rs>

Is the general-purpose source register holding a shift amount in its bottom 8 bits, encoded in the “Rs” field.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant integer shift_n = UInt(R[s]<7:0>);
    bits(32) shifted;
    bit carry;
    (shifted, carry) = Shift_C(R[m], shift_t, shift_n, PSTATE.C);
    constant bits(32) result = R[n] AND NOT(shifted);
    R[d] = result;
    if setflags then
        PSTATE.N = result<31>;
        PSTATE.Z = IsZeroBit(result);
        PSTATE.C = carry;
        // PSTATE.V unchanged
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.24 BKPT

Breakpoint causes a Breakpoint Instruction exception.

Breakpoint is always unconditional, even when inside an IT block.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

31–28	27–26	25–23	22–21	20	19–8	7	6–4	3–0
!=1111 (cond)	0 0	0 1 0	0 1 (opc)	0	imm12	0	1 1 1	imm4

Encoding

```
BKPT{<q>}  {#}<imm>
```

Decode for this encoding

Listing F5-7

```
constant bits(16) imm16 = imm12:imm4;
if cond != '1110' then UNPREDICTABLE; end  // BKPT must be encoded with AL condition
```

CONSTRAINED UNPREDICTABLE behavior

If `cond != '1110'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction executes unconditionally.
- The instruction executes conditionally.

T1

15–12	11–8	7–0
1 0 1 1	1 1 1 0	imm8

Encoding

```
BKPT{<q>}  {#}<imm>
```

Decode for this encoding

```
constant bits(16) imm16 = ZeroExtend(imm8, 16);
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<q>

See Standard assembler syntax fields. A `BKPT` instruction must be unconditional.

<imm>

For the "A1" variant: is a 16-bit unsigned immediate, in the range 0 to 65535, encoded in the "imm12:imm4" field. This value:

- Is recorded in the Comment field of ESR_ELx.ISS if the Software Breakpoint Instruction exception is taken to an exception level that is using AArch64.
- Is ignored otherwise.

For the "T1" variant: is a 8-bit unsigned immediate, in the range 0 to 255, encoded in the "imm8" field. This value:

- Is recorded in the Comment field of ESR_ELx.ISS if the Software Breakpoint Instruction exception is taken to an exception level that is using AArch64.
- Is ignored otherwise.

Operation

```
EncodingSpecificOperations();
AArch32.SoftwareBreakpoint(imm16);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.25 BL, BLX (immediate)

Branch with Link and optional Exchange (immediate)

Branch with Link calls a subroutine at a PC-relative address, and setting LR to the return address.

Branch with Link and Exchange Instruction Sets (immediate) calls a subroutine at a PC-relative address, setting LR to the return address, and changes the instruction set from A32 to T32, or from T32 to A32.

It has encodings from the following instruction sets: A32 (A1 and A2) and T32 (T1 and T2).

A1

31–28	27	26	25	24	23–0
!=1111 (cond)	1	0	1	1 (H)	imm24

Encoding

```
BL{<c>}{<q>}  <label>
```

Decode for this encoding

```
constant bits(32) imm32 = SignExtend(imm24:'00', 32);
constant InstrSet targetInstrSet = InstrSet_A32;
```

A2

31–28	27	26	25	24	23–0
1 1 1 1	1	0	1	H	imm24

Encoding

```
BLX{<c>}{<q>}  <label>
```

Decode for this encoding

```
constant bits(32) imm32 = SignExtend(imm24:H:'0', 32);
constant InstrSet targetInstrSet = InstrSet_T32;
```

T1

15–13	12	11	10	9–0	15	14	13	12	11	10–0
1 1 1	1	0	S	imm10	1	1	J1	1	J2	imm11

Encoding

```
BL{<c>}{<q>}  <label>
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for this encoding

```
constant bit I1 = NOT(J1 EOR S);
constant bit I2 = NOT(J2 EOR S);
constant bits(32) imm32 = SignExtend(S:I1:I2:imm10:imm11:'0', 32);
constant InstrSet targetInstrSet = InstrSet_T32;
if InITBlock() && !LastInITBlock() then UNPREDICTABLE;
```

T2

15		13	12	11	10	9 … 0	15	14	13	12	11	10 … 1	0
1	1	1	1	0	S	imm10H	1	1	J1	0	J2	imm10L	H

Encoding

```
BLX{<c>}{<q>}  <label>
```

Decode for this encoding

```
if H == '1' then UNDEFINED;
constant bit I1 = NOT(J1 EOR S);
constant bit I2 = NOT(J2 EOR S);
constant bits(32) imm32 = SignExtend(S:I1:I2:imm10H:imm10L:'00', 32);
constant InstrSet targetInstrSet = InstrSet_A32;
if InITBlock() && !LastInITBlock() then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

For the “A1”, “T1”, and “T2” variants: see Standard assembler syntax fields.

For the “A2” variant: see Standard assembler syntax fields. `<c>` must be `AL` or omitted.

<q>

See Standard assembler syntax fields.

<label>

For the “A1” variant: the label of the instruction that is to be branched to. The assembler calculates the required value of the offset from the PC value of the `BL` instruction to this label, then selects an encoding that sets `imm32` to that offset.

Permitted offsets are multiples of 4 in the range –33554432 to 33554428.

For the “A2” variant: the label of the instruction that is to be branched to. The assembler calculates the required value of the offset from the PC value of the `BLX` instruction to this label, then selects an encoding with `imm32` set to that offset.

Permitted offsets are even numbers in the range –33554432 to 33554430.

For the “T1” variant: the label of the instruction that is to be branched to.

The assembler calculates the required value of the offset from the PC value of the `BL` instruction to this label, then selects an encoding with `imm32` set to that offset.

Permitted offsets are even numbers in the range –16777216 to 16777214.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

For the “T2” variant: the label of the instruction that is to be branched to.

The assembler calculates the required value of the offset from the `Align(PC, 4)` value of the `BLX` instruction to this label, then selects an encoding with `imm32` set to that offset.

Permitted offsets are multiples of 4 in the range –16777216 to 16777212.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    if CurrentInstrSet() == InstrSet_A32 then
        LR = PC32 - 4;
    else
        LR = PC32<31:1> : '1';
    bits(32) targetAddress;
    if targetInstrSet == InstrSet_A32 then
        targetAddress = Align(PC32,4) + imm32;
    else
        targetAddress = PC32 + imm32;
    SelectInstrSet(targetInstrSet);
    BranchWritePC(targetAddress, BranchType_DIRCALL);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.26 BLX (register)

Branch with Link and Exchange (register) calls a subroutine at an address specified in the register, and if necessary changes to the instruction set indicated by bit[0] of the register value. If the value in bit[0] is 0, the instruction set after the branch will be A32. If the value in bit[0] is 1, the instruction set after the branch will be T32.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

31–28	27 26	25–23	22 21	20	19–8	7	6–4	3–0
!=1111 (cond)	0 0	0 1 0	0 1 (op0)	0	(1) (1) (1) (1) (1) (1) (1) (1) (1) (1) (1) (1)	0	0 1 1	Rm

Encoding

```
BLX{<c>}{<q>}  <Rm>
```

Decode for this encoding

```
constant integer m = UInt(Rm);
if m == 15 then UNPREDICTABLE;
```

T1

15–10	9 8	7	6–3	2–0
0 1 0 0 0 1	1 1	1 (L)	Rm	(0) (0) (0)

Encoding

```
BLX{<c>}{<q>}  <Rm>
```

Decode for this encoding

```
constant integer m = UInt(Rm);
if m == 15 then UNPREDICTABLE;
if InITBlock() && !LastInITBlock() then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rm>

Is the general-purpose register holding the address to be branched to, encoded in the “Rm” field.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) target = R[m];
    bits(32) next_instr_addr;
    if CurrentInstrSet() == InstrSet_A32 then
        next_instr_addr = PC32 - 4;
        LR = next_instr_addr;
    else
        next_instr_addr = PC32 - 2;
        LR = next_instr_addr<31:1> : '1';
    BXWritePC(target, BranchType_INDCALL);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.27 BX

Branch and Exchange causes a branch to an address and instruction set specified by a register.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

31–28	27	26	25	23–22	21	20	19–8	7	6–4	3–0
!=1111 (cond)	0	0	0	1	0	0 (op0)	(1)(1)(1)(1)(1)(1)(1)(1)(1)(1)(1)(1)	0	0 0 1	Rm

Encoding

```
BX{<c>}{<q>}  <Rm>
```

Decode for this encoding

```
constant integer m = UInt(Rm);
```

T1

15–10	9	8	7	6–3	2–0
0 1 0 0 0 1	1	1	0 (L)	Rm	(0)(0)(0)

Encoding

```
BX{<c>}{<q>}  <Rm>
```

Decode for this encoding

```
constant integer m = UInt(Rm);
if InITBlock() && !LastInITBlock() then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rm>

For the “A1” variant: is the general-purpose register holding the address to be branched to, encoded in the “Rm” field. The PC can be used.

For the “T1” variant: is the general-purpose register holding the address to be branched to, encoded in the “Rm” field. The PC can be used.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Note

If `<Rm>` is the PC at a non word-aligned address, it results in UNPREDICTABLE behavior because the address passed to the `BXWritePC()` pseudocode function has bits<1:0> = ‘10’.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    BXWritePC(R[m], BranchType_INDIR);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.28 BXJ

Branch and Exchange, previously Branch and Exchange Jazelle.

`BXJ` behaves as a `BX` instruction, see BX. This means it causes a branch to an address and instruction set specified by a register.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

31–28	27	26	25	23	22	21	20	19–8	7	6	4	3–0
!=1111 (cond)	0	0	0	0	0	1	0	(1)(1)(1)(1)(1)(1)(1)(1)(1)(1)(1)(1)	0	0	0	Rm

op0

Encoding

```
BXJ{<c>}{<q>}  <Rm>
```

Decode for this encoding

```
constant integer m = UInt(Rm);
if m == 15 then UNPREDICTABLE;
```

T1

15–13	12	11	10	9–6	5	4	3–0	15	14–12	11	10–8	7	6	5	4–0
1 1 1	1	0	0	1 1 1 1	0	0	Rm	1	0(0)0	(1)	(1)(1)(1)	(0)	(0)	(0)	(0)(0)(0)(0)(0)

Encoding

```
BXJ{<c>}{<q>}  <Rm>
```

Decode for this encoding

```
constant integer m = UInt(Rm);
// Armv8-A removes UNPREDICTABLE for R13
if m == 15 then UNPREDICTABLE;
if InITBlock() && !LastInITBlock() then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rm>

Is the general-purpose register holding the address to be branched to, encoded in the “Rm” field.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    BXWritePC(R[m], BranchType_INDIR);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.29 CBNZ, CBZ

Compare and Branch on Nonzero or Zero

Compare and Branch on Nonzero and Compare and Branch on Zero compare the value in a register with zero, and conditionally branch forward a constant value. They do not affect the condition flags.

15			12	11	10	9	8	7	3	2	0
1	0	1	1	op	0	i	1	imm5		Rn	

Encoding for the CBNZ variant

Applies when `(op == 1)`

```
CBNZ{<q>}  <Rn>, <label>
```

Encoding for the CBZ variant

Applies when `(op == 0)`

```
CBZ{<q>}  <Rn>, <label>
```

Decode for all variants of this encoding

Listing F5-8

```
constant integer n = UInt(Rn);
constant bits(32) imm32 = ZeroExtend(i:imm5:'0', 32);
constant boolean nonzero = (op == '1');
if InITBlock() then UNPREDICTABLE; end
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<q>

See Standard assembler syntax fields.

<Rn>

Is the general-purpose register to be tested, encoded in the “Rn” field.

<label>

Is the program label to be conditionally branched to. Its offset from the PC, a multiple of 2 and in the range 0 to 126, is encoded as “i:imm5” times 2.

Operation

Listing F5-9

```
EncodingSpecificOperations();
if nonzero != IsZero(R[n]) then
    CBWritePC(PC32 + imm32);
end
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.30 CLRBHB

Clear Branch History clears the branch history for the current context to the extent that branch history information created before the `CLRBHB` instruction cannot be used by code before the `CLRBHB` instruction to exploitatively control the execution of any indirect branches in code in the current context that appear in program order after the instruction.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

(FEAT_CLRBHB)

Encoding

```
CLRBHB{<c>}{<q>}
```

Decode for this encoding

Listing F5-10

```
if !IsFeatureImplemented(FEAT_CLRBHB) then ExecuteAsNOP(); end
```

T1

(FEAT_CLRBHB)

Encoding

```
CLRBHB{<c>}{<q>}
```

Decode for this encoding

Listing F5-11

```
if !IsFeatureImplemented(FEAT_CLRBHB) then ExecuteAsNOP(); end
```

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Operation

Listing F5-12

```
if ConditionPassed() then
    EncodingSpecificOperations();
    Hint_CLRBHB();
end
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.31 CLREX

Clear-Exclusive clears the local monitor of the executing PE.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

31–28	27	26	25–21	20	19–8	7–4	3–0
1 1 1 1	0	1	0 1 0 1 1	1	(1) (1) (1) (1) (1) (1) (1) (1) (0) (0) (0) (0)	0 0 0 1	(1) (1) (1) (1)
						opcode	option

Encoding

```
CLREX{<c>}{<q>}
```

Decode for this encoding

Listing F5-13

```
// No additional decoding required
```

T1

15–13	12 11	10	9–6	5 4	3–0	15	14–12	11	10–8	7–4	3–0
1 1 1	1 0	0	1 1 1 0	1 1	(1) (1) (1) (1)	1	0 (0) 0	(1)	(1) (1) (1)	0 0 1 0	(1) (1) (1) (1)
										opc	option

Encoding

```
CLREX{<c>}{<q>}
```

Decode for this encoding

Listing F5-14

```
// No additional decoding required
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

For the "A1" variant: see Standard assembler syntax fields. Must be `AL` or omitted.

For the "T1" variant: see Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Operation

Listing F5-15

```
if ConditionPassed() then
    EncodingSpecificOperations();
    ClearExclusiveLocal(ProcessorID());
end
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.32 CLZ

Count Leading Zeros returns the number of binary zero bits before the first binary one bit in a value.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

31–28	27–26	25–23	22–21	20	19–16	15–12	11–8	7	6–4	3–0
!=1111 (cond)	0 0	0 1 0	1 1 (op0)	0	(1) (1) (1) (1)	Rd	(1) (1) (1) (1)	0	0 0 1	Rm

Encoding

```
CLZ{<c>}{<q>}  <Rd>, <Rm>
```

Decode for this encoding

```
constant integer d = UInt(Rd);
constant integer m = UInt(Rm);
if d == 15 || m == 15 then UNPREDICTABLE;
```

T1

15–13	12–8	7	6–4	3–0	15–12	11–8	7 6	5 4	3–0
1 1 1	1 1 0 1 0	1	0 1 1 (op1)	Rn	1 1 1 1	Rd	1 0	0 0 (op2)	Rm

Encoding

```
CLZ{<c>}{<q>}  <Rd>, <Rm>
```

Decode for this encoding

```
constant integer d = UInt(Rd);
constant integer m = UInt(Rm);
constant integer n = UInt(Rn);
// Armv8-A removes UNPREDICTABLE for R13
if m != n || d == 15 || m == 15 then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `m != n`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction executes as described, with no change to its behavior and no additional side effects.
- The instruction executes with the additional decode: m = UInt(Rn);.
- The value in the destination register is UNKNOWN.

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

Is the general-purpose destination register, encoded in the “Rd” field.

<Rm>

For the “A1” variant: is the general-purpose source register, encoded in the “Rm” field.

For the “T1” variant: is the general-purpose source register, encoded in the “Rm” field. It must be encoded with an identical value in the “Rn” field.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant integer result = CountLeadingZeroBits(R[m]);
    R[d] = result<31:0>;
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.33 CMN (immediate)

Compare Negative (immediate) adds a register value and an immediate value. It updates the condition flags based on the result, and discards the result.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding

```
CMN{<c>}{<q>}  <Rn>, #<const>
```

Decode for this encoding

```
constant integer n = UInt(Rn);
constant bits(32) imm32 = A32ExpandImm(imm12);
```

T1

Encoding

```
CMN{<c>}{<q>}  <Rn>, #<const>
```

Decode for this encoding

```
constant integer n = UInt(Rn);
constant bits(32) imm32 = T32ExpandImm(i:imm3:imm8);
if n == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

<Rn>

For the “A1” variant: is the general-purpose source register, encoded in the “Rn” field. The PC can be used, but this is deprecated.

For the “T1” variant: is the general-purpose source register, encoded in the “Rn” field.

<const>

For the “A1” variant: an immediate value. See Modified immediate constants in A32 instructions for the range of values.

For the “T1” variant: an immediate value. See Modified immediate constants in T32 instructions for the range of values.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    bits(32) result;
    bits(4) nzcv;
    (result, nzcv) = AddWithCarry(R[n], imm32, '0');
    PSTATE.<N,Z,C,V> = nzcv;
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.34 CMN (register)

Compare Negative (register) adds a register value and an optionally-shifted register value. It updates the condition flags based on the result, and discards the result.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1 and T2).

A1

Encoding for the Rotate right with extend variant

Applies when `(imm5 == 00000 && stype == 11)`

```
CMN{<c>}{<q>}  <Rn>, <Rm>, RRX
```

Encoding for the Shift or rotate by value variant

Applies when `(!(imm5 == 00000 && stype == 11))`

```
CMN{<c>}{<q>}  <Rn>, <Rm> {, <shift> #<amount>}
```

Decode for all variants of this encoding

```
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
SRType shift_t;
integer shift_n;
(shift_t, shift_n) = DecodeImmShift(stype, imm5);
```

T1

Encoding

```
CMN{<c>}{<q>}  <Rn>, <Rm>
```

Decode for this encoding

```
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant SRType shift_t = SRType_LSL;
constant integer shift_n = 0;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T2

15–9	8–5	4	3–0	15	14–12	11–8	7–6	5–4	3–0
1 1 1 0 1 0 1	1 0 0 0 (op1)	1 (S)	Rn	(0)	imm3	1 1 1 1 (Rd)	imm2	stype	Rm

Encoding for the Rotate right with extend variant

Applies when `(imm3 == 000 && imm2 == 00 && stype == 11)`

```
CMN{<c>}{<q>}  <Rn>, <Rm>, RRX
```

Encoding for the Shift or rotate by value variant

Applies when `(!(imm3 == 000 && imm2 == 00 && stype == 11))`

```
CMN{<c>}{<q>}  <Rn>, <Rm> {, <shift> #<amount>}
```

```
CMN{<c>}.W  <Rn>, <Rm> // (<Rn>, <Rm> can be represented in T1)
```

Decode for all variants of this encoding

```
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
SRType shift_t;
integer shift_n;
(shift_t, shift_n) = DecodeImmShift(stype, imm3:imm2);
// Armv8-A removes UNPREDICTABLE for R13
if n == 15 || m == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rn>

For the “A1 Rotate right with extend” and “A1 Shift or rotate by value” variants: is the first general-purpose source register, encoded in the “Rn” field. The PC can be used, but this is deprecated.

For the “T1”, “T2 Rotate right with extend”, and “T2 Shift or rotate by value” variants: is the first general-purpose source register, encoded in the “Rn” field.

<Rm>

For the “A1 Rotate right with extend” and “A1 Shift or rotate by value” variants: is the second general-purpose source register, encoded in the “Rm” field. The PC can be used, but this is deprecated.

For the “T1”, “T2 Rotate right with extend”, and “T2 Shift or rotate by value” variants: is the second general-purpose source register, encoded in the “Rm” field.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

<shift>

Is the type of shift to be applied to the second source register, encoded in “stype”:

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

<amount>

For the “A1 Shift or rotate by value” variant: is the shift amount, in the range 1 to 31 (when <shift> = LSL or ROR) or 1 to 32 (when <shift> = LSR or ASR) encoded in the “imm5” field as <amount> modulo 32.

For the “T2 Shift or rotate by value” variant: is the shift amount, in the range 1 to 31 (when <shift> = LSL or ROR) or 1 to 32 (when <shift> = LSR or ASR), encoded in the “imm3:imm2” field as <amount> modulo 32.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) shifted = Shift(R[m], shift_t, shift_n, PSTATE.C);
    bits(32) result;
    bits(4) nzcv;
    (result, nzcv) = AddWithCarry(R[n], shifted, '0');
    PSTATE.<N,Z,C,V> = nzcv;
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.35 CMN (register-shifted register)

Compare Negative (register-shifted register) adds a register value and a register-shifted register value. It updates the condition flags based on the result, and discards the result.

31–28	27–26	25	24–23	22–21	20	19–16	15–12	11–8	7	6–5	4	3–0
!=1111 (cond)	0 0	0	1 0	1 1 (opc)	1	Rn	(0)(0)(0)(0)	Rs	0	stype	1	Rm

Encoding

```
CMN{<c>}{<q>}  <Rn>, <Rm>, <type> <Rs>
```

Decode for this encoding

```
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant integer s = UInt(Rs);
constant SRType shift_t = DecodeRegShift(stype);
if n == 15 || m == 15 || s == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rn>

Is the first general-purpose source register, encoded in the “Rn” field.

<Rm>

Is the second general-purpose source register, encoded in the “Rm” field.

<type>

Is the type of shift to be applied to the second source register, encoded in “stype”:

stype	<type>
00	LSL
01	LSR
10	ASR
11	ROR

<Rs>

Is the third general-purpose source register holding a shift amount in its bottom 8 bits, encoded in the “Rs” field.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant integer shift_n = UInt(R[s]<7:0>);
    constant bits(32) shifted = Shift(R[m], shift_t, shift_n, PSTATE.C);
    bits(32) result;
    bits(4) nzcv;
    (result, nzcv) = AddWithCarry(R[n], shifted, '0');
    PSTATE.<N,Z,C,V> = nzcv;
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.36 CMP (immediate)

Compare (immediate) subtracts an immediate value from a register value. It updates the condition flags based on the result, and discards the result.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1 and T2).

A1

Encoding

```
CMP{<c>}{<q>}  <Rn>, #<const>
```

Decode for this encoding

```
constant integer n = UInt(Rn);
constant bits(32) imm32 = A32ExpandImm(imm12);
```

T1

Encoding

```
CMP{<c>}{<q>}  <Rn>, #<imm8>
```

Decode for this encoding

```
constant integer n = UInt(Rn);
constant bits(32) imm32 = ZeroExtend(imm8, 32);
```

T2

Encoding

```
CMP{<c>}{<q>}  <Rn>, #<const>
```

```
CMP{<c>}.W  <Rn>, #<const> // (<Rd>, <const> can be represented in T1)
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for this encoding

```
constant integer n = UInt(Rn);
constant bits(32) imm32 = T32ExpandImm(i:imm3:imm8);
if n == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rn>

For the "A1" variant: is the general-purpose source register, encoded in the "Rn" field. The PC can be used, but this is deprecated.

For the "T1" variant: is a general-purpose source register, encoded in the "Rn" field.

For the "T2" variant: is the general-purpose source register, encoded in the "Rn" field.

<const>

For the "A1" variant: an immediate value. See Modified immediate constants in A32 instructions for the range of values.

For the "T2" variant: an immediate value. See Modified immediate constants in T32 instructions for the range of values.

<imm8>

Is a 8-bit unsigned immediate, in the range 0 to 255, encoded in the "imm8" field.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    bits(32) result;
    bits(4) nzcv;
    (result, nzcv) = AddWithCarry(R[n], NOT(imm32), '1');
    PSTATE.<N,Z,C,V> = nzcv;
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.37 CMP (register)

Compare (register) subtracts an optionally-shifted register value from a register value. It updates the condition flags based on the result, and discards the result.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1, T2, and T3).

A1

Encoding for the Rotate right with extend variant

Applies when `(imm5 == 00000 && stype == 11)`

```
CMP{<c>}{<q>}  <Rn>, <Rm>, RRX
```

Encoding for the Shift or rotate by value variant

Applies when `(!(imm5 == 00000 && stype == 11))`

```
CMP{<c>}{<q>}  <Rn>, <Rm> {, <shift> #<amount>}
```

Decode for all variants of this encoding

```
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
SRType shift_t;
integer shift_n;
(shift_t, shift_n) = DecodeImmShift(stype, imm5);
```

T1

Encoding

```
CMP{<c>}{<q>}  <Rn>, <Rm> // (<Rn> and <Rm> both from R0-R7)
```

Decode for this encoding

```
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant SRType shift_t = SRType_LSL;
constant integer shift_n = 0;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T2

Encoding

```
CMP{<c>}{<q>}  <Rn>, <Rm> // (<Rn> and <Rm> not both from R0-R7)
```

Decode for this encoding

```
constant integer n = UInt(N:Rn);
constant integer m = UInt(Rm);
constant SRType shift_t = SRType_LSL;
constant integer shift_n = 0;
if n < 8 && m < 8 then UNPREDICTABLE;
if n == 15 || m == 15 then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `n < 8 && m < 8`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction executes as described, with no change to its behavior and no additional side effects.
- The condition flags become UNKNOWN.

T3

Encoding for the Rotate right with extend variant

Applies when `(imm3 == 000 && imm2 == 00 && stype == 11)`

```
CMP{<c>}{<q>}  <Rn>, <Rm>, RRX
```

Encoding for the Shift or rotate by value variant

Applies when `(!(imm3 == 000 && imm2 == 00 && stype == 11))`

```
CMP{<c>}{<q>}  <Rn>, <Rm>, <shift> #<amount>
```

```
CMP{<c>}.W  <Rn>, <Rm> // (<Rn>, <Rm> can be represented in T1 or T2)
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for all variants of this encoding

```
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
SRType shift_t;
integer shift_n;
(shift_t, shift_n) = DecodeImmShift(stype, imm3:imm2);
// Armv8-A removes UNPREDICTABLE for R13
if n == 15 || m == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rn>

For the “A1 Rotate right with extend” and “A1 Shift or rotate by value” variants: is the first general-purpose source register, encoded in the “Rn” field. The PC can be used, but this is deprecated.

For the “T1”, “T3 Rotate right with extend”, and “T3 Shift or rotate by value” variants: is the first general-purpose source register, encoded in the “Rn” field.

For the “T2” variant: is the first general-purpose source register, encoded in the “N:Rn” field.

<Rm>

For the “A1 Rotate right with extend” and “A1 Shift or rotate by value” variants: is the second general-purpose source register, encoded in the “Rm” field. The PC can be used, but this is deprecated.

For the “T1”, “T2”, “T3 Rotate right with extend”, and “T3 Shift or rotate by value” variants: is the second general-purpose source register, encoded in the “Rm” field.

<shift>

Is the type of shift to be applied to the second source register, encoded in “stype”:

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

<amount>

For the “A1 Shift or rotate by value” variant: is the shift amount, in the range 1 to 31 (when <shift> = LSL or ROR) or 1 to 32 (when <shift> = LSR or ASR) encoded in the “imm5” field as <amount> modulo 32.

For the “T3 Shift or rotate by value” variant: is the shift amount, in the range 1 to 31 (when <shift> = LSL or ROR) or 1 to 32 (when <shift> = LSR or ASR), encoded in the “imm3:imm2” field as <amount> modulo 32.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) shifted = Shift(R[m], shift_t, shift_n, PSTATE.C);
    bits(32) result;
    bits(4) nzcv;
    (result, nzcv) = AddWithCarry(R[n], NOT(shifted), '1');
    PSTATE.<N,Z,C,V> = nzcv;
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.38 CMP (register-shifted register)

Compare (register-shifted register) subtracts a register-shifted register value from a register value. It updates the condition flags based on the result, and discards the result.

31–28	27–26	25	24–23	22–21	20	19–16	15–12	11–8	7	6–5	4	3–0
!=1111 (cond)	0 0	0	1 0	1 0 (opc)	1	Rn	(0)(0)(0)(0)	Rs	0	stype	1	Rm

Encoding

```
CMP{<c>}{<q>}  <Rn>, <Rm>, <type> <Rs>
```

Decode for this encoding

```
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant integer s = UInt(Rs);
constant SRType shift_t = DecodeRegShift(stype);
if n == 15 || m == 15 || s == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rn>

Is the first general-purpose source register, encoded in the “Rn” field.

<Rm>

Is the second general-purpose source register, encoded in the “Rm” field.

<type>

Is the type of shift to be applied to the second source register, encoded in “stype”:

stype	<type>
00	LSL
01	LSR
10	ASR
11	ROR

<Rs>

Is the third general-purpose source register holding a shift amount in its bottom 8 bits, encoded in the “Rs” field.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant integer shift_n = UInt(R[s]<7:0>);
    constant bits(32) shifted = Shift(R[m], shift_t, shift_n, PSTATE.C);
    bits(32) result;
    bits(4) nzcv;
    (result, nzcv) = AddWithCarry(R[n], NOT(shifted), '1');
    PSTATE.<N,Z,C,V> = nzcv;
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.39 CPS, CPSID, CPSIE

Change PE State changes one or more of the PSTATE.{A, I, F} interrupt mask bits and, optionally, the PSTATE.M mode field, without changing any other PSTATE bits.

`CPS` is treated as `NOP` if executed in User mode unless it is defined as being CONSTRAINED UNPREDICTABLE elsewhere in this section.

The PE checks whether the value being written to PSTATE.M is legal. See Illegal changes to PSTATE.M.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1 and T2).

A1

31											20	19 18	17	16	15						10	9	8	7	6	5	4	0
1	1	1	1	0	0	0	1	0	0	0	0	imod	M	0	(0)	(0)	(0)	(0)	(0)	(0)		(0)	A	I	F	0	mode	
														op								E						

Encoding for the Change mode variant

Applies when `(imod == 00 && M == 1)`

```
CPS{<q>}  #<mode> // (Cannot be conditional)
```

Encoding for the Interrupt disable variant

Applies when `(imod == 11 && M == 0)`

```
CPSID{<q>}  <iflags> // (Cannot be conditional)
```

Encoding for the Interrupt disable and change mode variant

Applies when `(imod == 11 && M == 1)`

```
CPSID{<q>}  <iflags> , #<mode> // (Cannot be conditional)
```

Encoding for the Interrupt enable variant

Applies when `(imod == 10 && M == 0)`

```
CPSIE{<q>}  <iflags> // (Cannot be conditional)
```

Encoding for the Interrupt enable and change mode variant

Applies when `(imod == 10 && M == 1)`

```
CPSIE{<q>}  <iflags> , #<mode> // (Cannot be conditional)
```

Decode for all variants of this encoding

```
if mode != '00000' && M == '0' then UNPREDICTABLE;
if (imod<1> == '1' && A:I:F == '000') || (imod<1> == '0' && A:I:F != '000') then UNPREDICTABLE;
constant boolean enable = (imod == '10');
constant boolean disable = (imod == '11');
constant boolean changemode = (M == '1');
constant bits(5) pemode = mode;
constant boolean affectA = (A == '1');
constant boolean affectI = (I == '1');
constant boolean affectF = (F == '1');
if (imod == '00' && M == '0') || imod == '01' then UNPREDICTABLE;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CONSTRAINED UNPREDICTABLE behavior

If `imod == '01'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.

If `imod == '00' && M == '0'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.

If `mode != '00000' && M == '0'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction executes with the additional decode: changemode = TRUE.
- The instruction executes as described, and the value specified by mode is ignored. There are no additional side-effects.

If `imod<1> == '1' && A:I:F == '000'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction behaves as if imod<1> == ‘0’.
- The instruction behaves as if A:I:F has an UNKNOWN nonzero value.

If `imod<1> == '0' && A:I:F != '000'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction behaves as if imod<1> == ‘1’.
- The instruction behaves as if A:I:F == ‘000’.

T1

15									6	5	4	3	2	1	0
1	0	1	1	0	1	1	0	0	1	1	im	(0)	A	I	F
										op					

Encoding for the Interrupt disable variant

Applies when `(im == 1)`

```
CPSID{<q>}  <iflags> // (Not permitted in IT block)
```

Encoding for the Interrupt enable variant

Applies when `(im == 0)`

```
CPSIE{<q>}  <iflags> // (Not permitted in IT block)
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for all variants of this encoding

```
if A:I:F == '000' then UNPREDICTABLE;
constant boolean enable = (im == '0');
constant boolean disable = (im == '1');
constant boolean changemode = FALSE;
constant boolean affectA = (A == '1');
constant boolean affectI = (I == '1');
constant boolean affectF = (F == '1');
constant bits(5) pemode = bits(5) UNKNOWN;
if InITBlock() then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `A:I:F == '000'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.

T2

15 … 11	10 … 9	8	7	6	5	4 … 0
111100111010(1)(1)(1)(1)10(0)0(0)	imod	M	A	I	F	mode

Encoding for the Change mode variant

Applies when `(imod == 00 && M == 1)`

```
CPS{<q>}  #<mode> // (Not permitted in IT block)
```

Encoding for the Interrupt disable variant

Applies when `(imod == 11 && M == 0)`

```
CPSID.W  <iflags> // (Not permitted in IT block)
```

Encoding for the Interrupt disable and change mode variant

Applies when `(imod == 11 && M == 1)`

```
CPSID{<q>}  <iflags>, #<mode> // (Not permitted in IT block)
```

Encoding for the Interrupt enable variant

Applies when `(imod == 10 && M == 0)`

```
CPSIE.W  <iflags> // (Not permitted in IT block)
```

Encoding for the Interrupt enable and change mode variant

Applies when `(imod == 10 && M == 1)`

```
CPSIE{<q>}  <iflags>, #<mode> // (Not permitted in IT block)
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for all variants of this encoding

```
if imod == '00' && M == '0' then SEE "Hint instructions";
if mode != '00000' && M == '0' then UNPREDICTABLE;
if (imod<1> == '1' && A:I:F == '000') || (imod<1> == '0' && A:I:F != '000') then UNPREDICTABLE;
constant boolean enable = (imod == '10');
constant boolean disable = (imod == '11');
constant boolean changemode = (M == '1');
constant bits(5) pemode = mode;
constant boolean affectA = (A == '1');
constant boolean affectI = (I == '1');
constant boolean affectF = (F == '1');
if imod == '01' || InITBlock() then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `imod == '01'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.

If `mode != '00000' && M == '0'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction executes with the additional decode: changemode = TRUE.
- The instruction executes as described, and the value specified by mode is ignored. There are no additional side-effects.

If `imod<1> == '1' && A:I:F == '000'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction behaves as if imod<1> == ‘0’.
- The instruction behaves as if A:I:F has an UNKNOWN nonzero value.

If `imod<1> == '0' && A:I:F != '000'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction behaves as if imod<1> == ‘1’.
- The instruction behaves as if A:I:F == ‘000’.

Hint instructions: In encoding T2, if the `imod` field is `0b00` and the `M` bit is `0b0`, a hint instruction is encoded. To determine which hint instruction, see Branches and miscellaneous control.

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<q>

See Standard assembler syntax fields.

<mode>

Is the number of the mode to change to, in the range 0 to 31, encoded in the “mode” field.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

<iflags>

Is a sequence of one or more of the following, specifying which interrupt mask bits are affected:

a Sets the A bit in the instruction, causing the specified effect on PSTATE.A, the SError interrupt mask bit.
i Sets the I bit in the instruction, causing the specified effect on PSTATE.I, the IRQ interrupt mask bit.
f Sets the F bit in the instruction, causing the specified effect on PSTATE.F, the FIQ interrupt mask bit.

Operation

```
if CurrentInstrSet() == InstrSet_A32 then
    EncodingSpecificOperations();
    if PSTATE.EL != EL0 then
        if enable then
            if affectA then PSTATE.A = '0';
            if affectI then PSTATE.I = '0';
            if affectF then PSTATE.F = '0';
        if disable then
            if affectA then PSTATE.A = '1';
            if affectI then PSTATE.I = '1';
            if affectF then PSTATE.F = '1';
        if changemode then
            // AArch32.WriteModeByInstr() sets PSTATE.IL to 1 if this is an illegal mode change.
            AArch32.WriteModeByInstr(pemode);
else
    EncodingSpecificOperations();
    if PSTATE.EL != EL0 then
        if enable then
            if affectA then PSTATE.A = '0';
            if affectI then PSTATE.I = '0';
            if affectF then PSTATE.F = '0';
        if disable then
            if affectA then PSTATE.A = '1';
            if affectI then PSTATE.I = '1';
            if affectF then PSTATE.F = '1';
        if changemode then
            // AArch32.WriteModeByInstr() sets PSTATE.IL to 1 if this is an illegal mode change.
            AArch32.WriteModeByInstr(pemode);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.40 CRC32

CRC32

`CRC32` performs a cyclic redundancy check (CRC) calculation on a value held in a general-purpose register. It takes an input CRC value in the first source operand, performs a CRC on the input value in the second source operand, and returns the output CRC value. The second source operand can be 8, 16, or 32 bits. To align with common usage, the bit order of the values is reversed as part of the operation, and the polynomial `0x04C11DB7` is used for the CRC calculation.

In an Armv8.0 implementation, this is an OPTIONAL instruction. From Armv8.1, it is mandatory for all implementations to implement this instruction.

Note

ID_ISAR5.CRC32 indicates whether this instruction is supported in the T32 and A32 instruction sets.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

(FEAT_CRC32)

31–28	27–23	22 21	20	19–16	15–12	11 10	9	8–4	3–0
!=1111	0 0 0 1 0	sz	0	Rn	Rd	(0)(0)	0	(0)0100	Rm
cond							C		

Encoding for the CRC32B variant

Applies when `(sz == 00)`

```
CRC32B{<q>}  <Rd>, <Rn>, <Rm>
```

Encoding for the CRC32H variant

Applies when `(sz == 01)`

```
CRC32H{<q>}  <Rd>, <Rn>, <Rm>
```

Encoding for the CRC32W variant

Applies when `(sz == 10)`

```
CRC32W{<q>}  <Rd>, <Rn>, <Rm>
```

Decode for all variants of this encoding

Listing F5-16

```
if !IsFeatureImplemented(FEAT_CRC32) then UNDEFINED; end
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant integer{} size = 8 << UInt(sz);
constant boolean crc32c = (C == '1');
if d == 15 || n == 15 || m == 15 then UNPREDICTABLE; end
if size == 64 then UNPREDICTABLE; end
if cond != '1110' then UNPREDICTABLE; end
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CONSTRAINED UNPREDICTABLE behavior

If `size == 64`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction executes with the additional decode: size = 32;.

If `cond != '1110'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction executes unconditionally.
- The instruction executes conditionally.

T1

(FEAT_CRC32)

15–5	4	3–0	15–12	11–8	7–6	5–4	3–0
1 1 1 1 1 0 1 0 1 1 0	0	Rn	1 1 1 1	Rd	1 0	sz	Rm
	C						

Encoding for the CRC32B variant

Applies when `(sz == 00)`

```
CRC32B{<q>}  <Rd>, <Rn>, <Rm>
```

Encoding for the CRC32H variant

Applies when `(sz == 01)`

```
CRC32H{<q>}  <Rd>, <Rn>, <Rm>
```

Encoding for the CRC32W variant

Applies when `(sz == 10)`

```
CRC32W{<q>}  <Rd>, <Rn>, <Rm>
```

Decode for all variants of this encoding

Listing F5-17

```
if InITBlock() then UNPREDICTABLE; end
if !IsFeatureImplemented(FEAT_CRC32) then UNDEFINED; end
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant integer{} size = 8 << UInt(sz);
constant boolean crc32c = (C == '1');
if d == 15 || n == 15 || m == 15 then UNPREDICTABLE; end
if size == 64 then UNPREDICTABLE; end
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CONSTRAINED UNPREDICTABLE behavior

If `size == 64`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction executes with the additional decode: size = 32;.

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<q>

See Standard assembler syntax fields. A `CRC32` instruction must be unconditional.

<Rd>

Is the general-purpose accumulator output register, encoded in the “Rd” field.

<Rn>

Is the general-purpose accumulator input register, encoded in the “Rn” field.

<Rm>

Is the general-purpose data source register, encoded in the “Rm” field.

Operation

Listing F5-18

```
if ConditionPassed() then
    EncodingSpecificOperations();

    constant bits(32) acc = R[n];             // accumulator
    constant bits(size) val = R[m]<size-1:0>; // input value
    constant bits(32) poly = (if crc32c then 0x1EDC6F41 else 0x04C11DB7)<31:0>;
    constant bits(32+size) tempacc = BitReverse(acc):Zeros(size);
    constant bits(32+size) tempval = BitReverse(val):Zeros(32);
    // Poly32Mod2 on a bitstring does a polynomial Modulus over {0,1} operation
    R[d] = BitReverse(Poly32Mod2(tempacc EOR tempval, poly));
end
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.41 CRC32C

CRC32C

`CRC32C` performs a cyclic redundancy check (CRC) calculation on a value held in a general-purpose register. It takes an input CRC value in the first source operand, performs a CRC on the input value in the second source operand, and returns the output CRC value. The second source operand can be 8, 16, or 32 bits. To align with common usage, the bit order of the values is reversed as part of the operation, and the polynomial `0x1EDC6F41` is used for the CRC calculation.

In an Armv8.0 implementation, this is an OPTIONAL instruction. From Armv8.1, it is mandatory for all implementations to implement this instruction.

Note

ID_ISAR5.CRC32 indicates whether this instruction is supported in the T32 and A32 instruction sets.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

(FEAT_CRC32)

Encoding for the CRC32CB variant

Applies when `(sz == 00)`

```
CRC32CB{<q>}  <Rd>, <Rn>, <Rm>
```

Encoding for the CRC32CH variant

Applies when `(sz == 01)`

```
CRC32CH{<q>}  <Rd>, <Rn>, <Rm>
```

Encoding for the CRC32CW variant

Applies when `(sz == 10)`

```
CRC32CW{<q>}  <Rd>, <Rn>, <Rm>
```

Decode for all variants of this encoding

Listing F5-19

```
if !IsFeatureImplemented(FEAT_CRC32) then UNDEFINED; end
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant integer{} size = 8 << UInt(sz);
constant boolean crc32c = (C == '1');
if d == 15 || n == 15 || m == 15 then UNPREDICTABLE; end
if size == 64 then UNPREDICTABLE; end
if cond != '1110' then UNPREDICTABLE; end
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CONSTRAINED UNPREDICTABLE behavior

If `size == 64`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction executes with the additional decode: size = 32;.

If `cond != '1110'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction executes unconditionally.
- The instruction executes conditionally.

T1

(FEAT_CRC32)

15 … 5	4	3 … 0	15 … 12	11 … 8	7 6	5 4	3 … 0
1 1 1 1 1 0 1 0 1 1 0	1	Rn	1 1 1 1	Rd	1 0	sz	Rm
	C						

Encoding for the CRC32CB variant

Applies when `(sz == 00)`

```
CRC32CB{<q>}  <Rd>, <Rn>, <Rm>
```

Encoding for the CRC32CH variant

Applies when `(sz == 01)`

```
CRC32CH{<q>}  <Rd>, <Rn>, <Rm>
```

Encoding for the CRC32CW variant

Applies when `(sz == 10)`

```
CRC32CW{<q>}  <Rd>, <Rn>, <Rm>
```

Decode for all variants of this encoding

Listing F5-20

```
if InITBlock() then UNPREDICTABLE; end
if !IsFeatureImplemented(FEAT_CRC32) then UNDEFINED; end
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant integer{} size = 8 << UInt(sz);
constant boolean crc32c = (C == '1');
if d == 15 || n == 15 || m == 15 then UNPREDICTABLE; end
if size == 64 then UNPREDICTABLE; end
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CONSTRAINED UNPREDICTABLE behavior

If `size == 64`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction executes with the additional decode: size = 32;.

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<q>

See Standard assembler syntax fields. A `CRC32C` instruction must be unconditional.

<Rd>

Is the general-purpose accumulator output register, encoded in the “Rd” field.

<Rn>

Is the general-purpose accumulator input register, encoded in the “Rn” field.

<Rm>

Is the general-purpose data source register, encoded in the “Rm” field.

Operation

Listing F5-21

```
if ConditionPassed() then
    EncodingSpecificOperations();

    constant bits(32) acc = R[n];             // accumulator
    constant bits(size) val = R[m]<size-1:0>; // input value
    constant bits(32) poly = (if crc32c then 0x1EDC6F41 else 0x04C11DB7)<31:0>;
    constant bits(32+size) tempacc = BitReverse(acc):Zeros(size);
    constant bits(32+size) tempval = BitReverse(val):Zeros(32);
    // Poly32Mod2 on a bitstring does a polynomial Modulus over {0,1} operation
    R[d] = BitReverse(Poly32Mod2(tempacc EOR tempval, poly));
end
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.42 CSDB

Consumption of Speculative Data Barrier is a memory barrier that controls speculative execution arising from data value prediction. For more information and details of the semantics, see Consumption of Speculative Data Barrier (CSDB).

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding

```
CSDB{<c>}{<q>}
```

Decode for this encoding

Listing F5-22

```
if cond != '1110' then UNPREDICTABLE; end      // CSDB must be encoded with AL condition
```

CONSTRAINED UNPREDICTABLE behavior

If `cond != '1110'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction executes unconditionally.
- The instruction executes conditionally.

T1

Encoding

```
CSDB{<c>}{<q>}
```

Decode for this encoding

Listing F5-23

```
if InITBlock() then UNPREDICTABLE; end
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CONSTRAINED UNPREDICTABLE behavior

If `InITBlock()`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction executes unconditionally.
- The instruction executes conditionally.

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

Operation

Listing F5-24

```
if ConditionPassed() then
    EncodingSpecificOperations();

    ConsumptionOfSpeculativeDataBarrier();
end
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.43 DBG

Debug hint

`DBG` executes as a `NOP`. Arm deprecates any use of the `DBG` instruction.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

31–28	27 26	25	24 23	22	21 20	19–16	15–12	11–4	3–0
!=1111 (cond)	0 0	1	1 0	0 (R)	1 0	0 0 0 0 (imm4)	(1)(1)(1)(1)	(0)(0)(0)(0)1111	option

Encoding

```
DBG{<c>}{<q>}  #<option>
```

Decode for this encoding

```
// DBG executes as a NOP. The 'option' field is ignored
```

T1

15–13	12 11	10	9–6	5 4	3–0	15	14–12	11	10–8	7–4	3–0
1 1 1	1 0	0	1 1 1 0	1 0	(1)(1)(1)(1)	1	0(0)0	(0)	0 0 0	1 1 1 1 (hint)	option

Encoding

```
DBG{<c>}{<q>}  #<option>
```

Decode for this encoding

```
// DBG executes as a NOP. The 'option' field is ignored
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<option>

Is a 4-bit unsigned immediate, in the range 0 to 15, encoded in the “option” field.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.44 DCPS1

Debug Change PE State to EL1 allows the debugger to move the PE into EL1 from EL0 or to a specific mode at the current Exception level.

`DCPS1` is UNDEFINED if any of:

- The PE is in Non-debug state.
- EL2 is implemented, EL2 is implemented and enabled in the current Security state, and any of:
  - EL2 is using AArch32 and HCR.TGE is set to 1.
  - EL2 is using AArch64 and HCR_EL2.TGE is set to 1.

When the PE executes `DCPS1` at EL0, EL1 or EL3:

- If EL3 or EL1 is using AArch32, the PE enters SVC mode and LR_svc, SPSR_svc, DLR, and DSPSR become UNKNOWN. If `DCPS1` is executed in Monitor mode, SCR.NS is cleared to 0.
- If EL1 is using AArch64, the PE enters EL1 using AArch64, selects SP_EL1, and ELR_EL1, ESR_EL1, SPSR_EL1, DLR_EL0 and DSPSR_EL0 become UNKNOWN.

When the PE executes `DCPS1` at EL2 the PE does not change mode, and ELR_hyp, HSR, SPSR_hyp, DLR and DSPSR become UNKNOWN.

For more information on the operation of the DCPS<n> instructions, see DCPS.

15 13	12 11	10	9 6	5 4	3 0	15	14 12	11 2	1 0
1 1 1	1 0	1	1 1 1 0	0 0	1 1 1 1	1	0 0 0	0 0 0 0 0 0 0 0 0 0	0 1
					imm4			imm10	opt

Encoding

```
DCPS1
```

Decode for this encoding

Listing F5-25

```
// No additional decoding required.
```

Operation

Listing F5-26

```
if !Halted() then UNDEFINED; end

if EL2Enabled() && PSTATE.EL == EL0 then
    constant bit tge = if ELUsingAArch32(EL2) then HCR.TGE else HCR_EL2.TGE;
    if tge == '1' then UNDEFINED; end
end

if PSTATE.EL != EL0 || ELUsingAArch32(EL1) then
    if PSTATE.M == M32_Monitor then SCR.NS = '0'; end
    if PSTATE.EL != EL2 then
        AArch32.WriteMode(M32_Svc);
        PSTATE.E = SCTLR.EE;
        if IsFeatureImplemented(FEAT_PAN) && SCTLR.SPAN == '0' then PSTATE.PAN = '1'; end
        Rmode[14, M32_Svc] = bits(32) UNKNOWN;    // LR_svc
        SPSR_svc = bits(32) UNKNOWN;
    else
        PSTATE.E = HSCTLR.EE;
        ELR_hyp = bits(32) UNKNOWN;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        HSR = bits(32) UNKNOWN;
        SPSR_hyp = bits(32) UNKNOWN;
    end

    DLR = bits(32) UNKNOWN;
    DSPSR = bits(32) UNKNOWN;
else                                        // Targeting EL1 using AArch64
    AArch64.MaybeZeroRegisterUppers();
    MaybeZeroSVEUppers(EL1);
    PSTATE.nRW = '0';
    PSTATE.SP = '1';
    PSTATE.EL = EL1;
    if IsFeatureImplemented(FEAT_PAN) && SCTLR_EL1.SPAN == '0' then PSTATE.PAN = '1'; end
    if IsFeatureImplemented(FEAT_UAO) then PSTATE.UAO = '0'; end

    ELR_EL1 = bits(64) UNKNOWN;
    ESR_EL1 = bits(64) UNKNOWN;
    SPSR_EL1 = bits(64) UNKNOWN;

    DLR_EL0 = bits(64) UNKNOWN;
    DSPSR_EL0 = bits(64) UNKNOWN;

    // SCTLR_EL1.IESB might be ignored in Debug state.
    if (IsFeatureImplemented(FEAT_IESB) && SCTLR_EL1.IESB == '1' &&
          !ConstrainUnpredictableBool(Unpredictable_IESBinDebug)) then
        SynchronizeErrors();
    end
end

UpdateEDSCRFields();                        // Update EDSCR PE state flags
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.45 DCPS2

Debug Change PE State to EL2 allows the debugger to move the PE into EL2 from a lower Exception level.

`DCPS2` is UNDEFINED if any of:

- The PE is in Non-debug state.
- EL2 is not implemented.
- The PE is in Secure state and any of:
  - Secure EL2 is not implemented.
  - Secure EL2 is implemented and Secure EL2 is disabled.

When the PE executes `DCPS2`:

- If EL2 is using AArch32, the PE enters Hyp mode and ELR_hyp, HSR, SPSR_hyp, DLR and DSPSR become UNKNOWN.
- If EL2 is using AArch64, the PE enters EL2 using AArch64, selects SP_EL2, and ELR_EL2, ESR_EL2, SPSR_EL2, DLR_EL0 and DSPSR_EL0 become UNKNOWN.

For more information on the operation of the DCPS<n> instructions, see DCPS.

15 13	12 11	10	9 6	5 4	3 0	15	14 12	11 2	1 0
1 1 1	1 0	1	1 1 1 0	0 0	1 1 1 1	1	0 0 0	0 0 0 0 0 0 0 0 0 0	1 0
					imm4			imm10	opt

Encoding

```
DCPS2
```

Decode for this encoding

Listing F5-27

```
if !HaveEL(EL2) then UNDEFINED; end
```

Operation

Listing F5-28

```
if !Halted() || !EL2Enabled() then UNDEFINED; end

if ELUsingAArch32(EL2) then
    AArch32.WriteMode(M32_Hyp);
    PSTATE.E = HSCTLR.EE;

    ELR_hyp = bits(32) UNKNOWN;
    HSR = bits(32) UNKNOWN;
    SPSR_hyp = bits(32) UNKNOWN;

    DLR = bits(32) UNKNOWN;
    DSPSR = bits(32) UNKNOWN;
else                                        // Targeting EL2 using AArch64
    AArch64.MaybeZeroRegisterUppers();
    MaybeZeroSVEUppers(EL2);
    PSTATE.nRW = '0';
    PSTATE.SP = '1';
    PSTATE.EL = EL2;
    if IsFeatureImplemented(FEAT_PAN) && SCTLR_EL2.SPAN == '0' && ELIsInHost(EL0) then
        PSTATE.PAN = '1';
    end
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    if IsFeatureImplemented(FEAT_UAO) then PSTATE.UAO = '0'; end

    ELR_EL2 = bits(64) UNKNOWN;
    ESR_EL2 = bits(64) UNKNOWN;
    SPSR_EL2 = bits(64) UNKNOWN;

    DLR_EL0 = bits(64) UNKNOWN;
    DSPSR_EL0 = bits(64) UNKNOWN;

    // SCTLR_EL2.IESB might be ignored in Debug state.
    if (IsFeatureImplemented(FEAT_IESB) && SCTLR_EL2.IESB == '1' &&
          !ConstrainUnpredictableBool(Unpredictable_IESBinDebug)) then
        SynchronizeErrors();
    end
end

UpdateEDSCRFields();                        // Update EDSCR PE state flags
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.46 DCPS3

Debug Change PE State to EL3 allows the debugger to move the PE into EL3 from a lower Exception level or to a specific mode at the current Exception level.

`DCPS3` is UNDEFINED if any of:

- The PE is in Non-debug state.
- EL3 is not implemented.
- EDSCR.SDD is set to 1.

When the PE executes `DCPS3`:

- If EL3 is using AArch32, the PE enters Monitor mode and LR_mon, SPSR_mon, DLR and DSPSR become UNKNOWN. If `DCPS3` is executed in Monitor mode, SCR.NS is cleared to 0.
- If EL3 is using AArch64, the PE enters EL3 using AArch64, selects SP_EL3, and ELR_EL3, ESR_EL3, SPSR_EL3, DLR_EL0 and DSPSR_EL0 become UNKNOWN.

For more information on the operation of the DCPS<n> instructions, see DCPS.

15 13	12 11	10	9 6	5 4	3 0	15	14 12	11 2	1 0
1 1 1	1 0	1	1 1 1 0	0 0	1 1 1 1	1	0 0 0	0 0 0 0 0 0 0 0 0 0	1 1
					imm4			imm10	opt

Encoding

```
DCPS3
```

Decode for this encoding

Listing F5-29

```
if !HaveEL(EL3) then UNDEFINED; end
```

Operation

Listing F5-30

```
if !Halted() || EDSCR.SDD == '1' then UNDEFINED; end

if ELUsingAArch32(EL3) then
    constant boolean from_secure = (CurrentSecurityState() == SS_Secure);
    if PSTATE.M == M32_Monitor then SCR.NS = '0'; end
    AArch32.WriteMode(M32_Monitor);
    if IsFeatureImplemented(FEAT_PAN) then
        if !from_secure then
            PSTATE.PAN = '0';
        elsif SCTLR.SPAN == '0' then
            PSTATE.PAN = '1';
        end
    end
    PSTATE.E = SCTLR.EE;

    LR_mon = bits(32) UNKNOWN;
    SPSR_mon = bits(32) UNKNOWN;

    DLR = bits(32) UNKNOWN;
    DSPSR = bits(32) UNKNOWN;
else                                        // Targeting EL3 using AArch64
    AArch64.MaybeZeroRegisterUppers();
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    MaybeZeroSVEUppers(EL3);
    PSTATE.nRW = '0';
    PSTATE.SP = '1';
    PSTATE.EL = EL3;
    if IsFeatureImplemented(FEAT_UAO) then PSTATE.UAO = '0'; end

    ELR_EL3 = bits(64) UNKNOWN;
    ESR_EL3 = bits(64) UNKNOWN;
    SPSR_EL3 = bits(64) UNKNOWN;

    DLR_EL0 = bits(64) UNKNOWN;
    DSPSR_EL0 = bits(64) UNKNOWN;

    boolean sync_errors = IsFeatureImplemented(FEAT_IESB) && SCTLR_EL3.IESB == '1';
    if IsFeatureImplemented(FEAT_DoubleFault) && EffectiveEA() == '1' && SCR_EL3.NMEA == '1' then
        sync_errors = TRUE;
    end
    // SCTLR_EL3.IESB might be ignored in Debug state.
    if !ConstrainUnpredictableBool(Unpredictable_IESBinDebug) then
        sync_errors = FALSE;
    end
    if sync_errors then SynchronizeErrors(); end
end

UpdateEDSCRFields();                        // Update EDSCR PE state flags
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.47 DMB

Data Memory Barrier is a memory barrier that ensures the ordering of observations of memory accesses, see Data Memory Barrier (DMB).

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

31–28	27	26	25–21	20	19–8	7–4	3–0
1 1 1 1	0	1	0 1 0 1 1	1	(1) (1) (1) (1) (1) (1) (1) (1) (0) (0) (0) (0)	0 1 0 1 (opcode)	option

Encoding

```
DMB{<c>}{<q>}  {<option>}
```

Decode for this encoding

Listing F5-31

```
// No additional decoding required
```

T1

15–13	12 11	10	9–6	5 4	3–0	15	14–12	11	10–8	7–4	3–0
1 1 1	1 0	0	1 1 1 0	1 1	(1) (1) (1) (1)	1	0 (0) 0	(1)	(1) (1) (1)	0 1 0 1 (opc)	option

Encoding

```
DMB{<c>}{<q>}  {<option>}
```

Decode for this encoding

Listing F5-32

```
// No additional decoding required
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

For the “A1” variant: see Standard assembler syntax fields. Must be `AL` or omitted.

For the “T1” variant: see Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

<option>

Specifies an optional limitation on the barrier operation. Values are:

SY Full system is the required shareability domain, reads and writes are the required access types, both before and after the barrier instruction. Can be omitted. This option is referred to as the full system barrier. Encoded as option = `0b1111`.

ST Full system is the required shareability domain, writes are the required access type, both before and after the barrier instruction. `SYST` is a synonym for `ST`. Encoded as option = `0b1110`.

LD Full system is the required shareability domain, reads are the required access type before the barrier instruction, and reads and writes are the required access types after the barrier instruction. Encoded as option = `0b1101`.

ISH Inner Shareable is the required shareability domain, reads and writes are the required access types, both before and after the barrier instruction. Encoded as option = `0b1011`.

ISHST Inner Shareable is the required shareability domain, writes are the required access type, both before and after the barrier instruction. Encoded as option = `0b1010`.

ISHLD Inner Shareable is the required shareability domain, reads are the required access type before the barrier instruction, and reads and writes are the required access types after the barrier instruction. Encoded as option = `0b1001`.

NSH Non-shareable is the required shareability domain, reads and writes are the required access, both before and after the barrier instruction. Encoded as option = `0b0111`.

NSHST Non-shareable is the required shareability domain, writes are the required access type both before and after the barrier instruction. Encoded as option = `0b0110`.

NSHLD Non-shareable is the required shareability domain, reads are the required access type before the barrier instruction, and reads and writes are the required access types after the barrier instruction. Encoded as option = `0b0101`.

OSH Outer Shareable is the required shareability domain, reads and writes are the required access types, both before and after the barrier instruction. Encoded as option = `0b0011`.

OSHST Outer Shareable is the required shareability domain, writes are the required access type, both before and after the barrier instruction. Encoded as option = `0b0010`.

OSHLD Outer Shareable is the required shareability domain, reads are the required access type before the barrier instruction, and reads and writes are the required access types after the barrier instruction. Encoded as option = `0b0001`.

For more information on whether an access is before or after a barrier instruction, see Data Memory Barrier (DMB). All other encodings of option are reserved. All unsupported and reserved options must execute as a full system DMB operation, but software must not rely on this behavior.

Note

The instruction supports the following alternative `<option>` values, but Arm recommends that software does not use these alternative values:

- `SH` as an alias for `ISH`.
- `SHST` as an alias for `ISHST`.
- `UN` as an alias for `NSH`.
- `UNST` as an alias for `NSHST`.

Operation

Listing F5-33

```
if ConditionPassed() then
    EncodingSpecificOperations();
    MBReqDomain domain;
    MBReqTypes types;
    case option of
        when '0001'  domain = MBReqDomain_OuterShareable;  types = MBReqTypes_Reads;
        when '0010'  domain = MBReqDomain_OuterShareable;  types = MBReqTypes_Writes;
        when '0011'  domain = MBReqDomain_OuterShareable;  types = MBReqTypes_All;
        when '0101'  domain = MBReqDomain_Nonshareable;    types = MBReqTypes_Reads;
        when '0110'  domain = MBReqDomain_Nonshareable;    types = MBReqTypes_Writes;
        when '0111'  domain = MBReqDomain_Nonshareable;    types = MBReqTypes_All;
        when '1001'  domain = MBReqDomain_InnerShareable;  types = MBReqTypes_Reads;
        when '1010'  domain = MBReqDomain_InnerShareable;  types = MBReqTypes_Writes;
        when '1011'  domain = MBReqDomain_InnerShareable;  types = MBReqTypes_All;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        when '1101'  domain = MBReqDomain_FullSystem;      types = MBReqTypes_Reads;
        when '1110'  domain = MBReqDomain_FullSystem;      types = MBReqTypes_Writes;
        otherwise    domain = MBReqDomain_FullSystem;      types = MBReqTypes_All;
    end

    if PSTATE.EL IN {EL0, EL1} && EL2Enabled() then
        if HCR.BSU == '11' then
            domain = MBReqDomain_FullSystem;
        end
        if HCR.BSU == '10' && domain != MBReqDomain_FullSystem then
            domain = MBReqDomain_OuterShareable;
        end
        if HCR.BSU == '01' && domain == MBReqDomain_Nonshareable then
            domain = MBReqDomain_InnerShareable;
        end
    end

    DataMemoryBarrier(domain, types);
end
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.48 DSB

Data Synchronization Barrier is a memory barrier that ensures the completion of memory accesses, see Data Synchronization Barrier (DSB).

An AArch32 DSB instruction does not require the completion of any AArch64 TLB maintenance instructions, regardless of the nXS qualifier, appearing in program order before the AArch32 DSB.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

31–28	27	26	25–21	20	19–8	7–4	3–0
1 1 1 1	0	1	0 1 0 1 1	1	(1) (1) (1) (1) (1) (1) (1) (1) (0) (0) (0) (0)	0 1 0 0 (opcode)	!=0x00 (option)

Encoding

```
DSB{<c>}{<q>}  {<option>}
```

Decode for this encoding

Listing F5-34

```
// No additional decoding required
```

T1

15–13	12 11	10	9–6	5 4	3–0	15	14–12	11	10–8	7–4	3–0
1 1 1	1 0	0	1 1 1 0	1 1	(1) (1) (1) (1)	1	0 (0) 0	(1)	(1) (1) (1)	0 1 0 0 (opc)	!=0x00 (option)

Encoding

```
DSB{<c>}{<q>}  {<option>}
```

Decode for this encoding

Listing F5-35

```
// No additional decoding required
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

For the "A1" variant: see Standard assembler syntax fields. Must be `AL` or omitted.

For the "T1" variant: see Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<option>

Specifies an optional limitation on the barrier operation. Values are:

SY Full system is the required shareability domain, reads and writes are the required access types, both before and after the barrier instruction. Can be omitted. This option is referred to as the full system barrier. Encoded as option = `0b1111`.

ST Full system is the required shareability domain, writes are the required access type, both before and after the barrier instruction. `SYST` is a synonym for `ST`. Encoded as option = `0b1110`.

LD Full system is the required shareability domain, reads are the required access type before the barrier instruction, and reads and writes are the required access types after the barrier instruction. Encoded as option = `0b1101`.

ISH Inner Shareable is the required shareability domain, reads and writes are the required access types, both before and after the barrier instruction. Encoded as option = `0b1011`.

ISHST Inner Shareable is the required shareability domain, writes are the required access type, both before and after the barrier instruction. Encoded as option = `0b1010`.

ISHLD Inner Shareable is the required shareability domain, reads are the required access type before the barrier instruction, and reads and writes are the required access types after the barrier instruction. Encoded as option = `0b1001`.

NSH Non-shareable is the required shareability domain, reads and writes are the required access, both before and after the barrier instruction. Encoded as option = `0b0111`.

NSHST Non-shareable is the required shareability domain, writes are the required access type both before and after the barrier instruction. Encoded as option = `0b0110`.

NSHLD Non-shareable is the required shareability domain, reads are the required access type before the barrier instruction, and reads and writes are the required access types after the barrier instruction. Encoded as option = `0b0101`.

OSH Outer Shareable is the required shareability domain, reads and writes are the required access types, both before and after the barrier instruction. Encoded as option = `0b0011`.

OSHST Outer Shareable is the required shareability domain, writes are the required access type, both before and after the barrier instruction. Encoded as option = `0b0010`.

OSHLD Outer Shareable is the required shareability domain, reads are the required access type before the barrier instruction, and reads and writes are the required access types after the barrier instruction. Encoded as option = `0b0001`.

For more information on whether an access is before or after a barrier instruction, see Data Synchronization Barrier (DSB). All other encodings of option are reserved, other than the values `0b0000` and `0b0100`. All unsupported and reserved options must execute as a full system DSB operation, but software must not rely on this behavior.

Note

The value `0b0000` is used to encode SSBB and the value `0b0100` is used to encode PSSBB.
The instruction supports the following alternative `<option>` values, but Arm recommends that software does not use these alternative values:

- `SH` as an alias for `ISH`.
- `SHST` as an alias for `ISHST`.
- `UN` as an alias for `NSH`.
- `UNST` as an alias for `NSHST`.

Operation

Listing F5-36

```
if ConditionPassed() then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
    EncodingSpecificOperations();
    boolean nXS;
    if IsFeatureImplemented(FEAT_XS) then
        nXS = (PSTATE.EL IN {EL0, EL1} && !ELUsingAArch32(EL2) &&
               IsHCRXEL2Enabled() && HCRX_EL2.FnXS == '1');
    else
        nXS = FALSE;
    end
    MBReqDomain domain;
    MBReqTypes types;
    case option of
        when '0001'  domain = MBReqDomain_OuterShareable;  types = MBReqTypes_Reads;
        when '0010'  domain = MBReqDomain_OuterShareable;  types = MBReqTypes_Writes;
        when '0011'  domain = MBReqDomain_OuterShareable;  types = MBReqTypes_All;
        when '0101'  domain = MBReqDomain_Nonshareable;    types = MBReqTypes_Reads;
        when '0110'  domain = MBReqDomain_Nonshareable;    types = MBReqTypes_Writes;
        when '0111'  domain = MBReqDomain_Nonshareable;    types = MBReqTypes_All;
        when '1001'  domain = MBReqDomain_InnerShareable;  types = MBReqTypes_Reads;
        when '1010'  domain = MBReqDomain_InnerShareable;  types = MBReqTypes_Writes;
        when '1011'  domain = MBReqDomain_InnerShareable;  types = MBReqTypes_All;
        when '1101'  domain = MBReqDomain_FullSystem;      types = MBReqTypes_Reads;
        when '1110'  domain = MBReqDomain_FullSystem;      types = MBReqTypes_Writes;
        otherwise
            assert option != '0x00';
            domain = MBReqDomain_FullSystem;  types = MBReqTypes_All;
    end

    if PSTATE.EL IN {EL0, EL1} && EL2Enabled() then
        if HCR.BSU == '11' then
            domain = MBReqDomain_FullSystem;
        end
        if HCR.BSU == '10' && domain != MBReqDomain_FullSystem then
            domain = MBReqDomain_OuterShareable;
        end
        if HCR.BSU == '01' && domain == MBReqDomain_Nonshareable then
            domain = MBReqDomain_InnerShareable;
        end
    end

    DataSynchronizationBarrier(domain, types, nXS);
end
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.49 EOR, EORS (immediate)

Bitwise Exclusive-OR (immediate) performs a bitwise exclusive-OR of a register value and an immediate value, and writes the result to the destination register.

If the destination register is not the PC, the EORS variant of the instruction updates the condition flags based on the result.

The field descriptions for `<Rd>` identify the encodings where the PC is permitted as the destination register. Arm deprecates any use of these encodings. However, when the destination register is the PC:

- The EOR variant of the instruction is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- The EORS variant of the instruction performs an exception return without the use of the stack. In this case:
  - The PE branches to the address written to the PC, and restores PSTATE from SPSR_<current_mode>.
  - The PE checks SPSR_<current_mode> for an illegal return event. See Illegal return events from AArch32 state.
  - The instruction is UNDEFINED in Hyp mode.
  - The instruction is CONSTRAINED UNPREDICTABLE in User mode and System mode.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

31–28	27–24	23–21	20	19–16	15–12	11–0
!=1111	0 0 1 0	0 0 1	S	Rn	Rd	imm12
cond		opc				

Encoding for the EOR variant

Applies when `(S == 0)`

```
EOR{<c>}{<q>}  {<Rd>, }<Rn>, #<const>
```

Encoding for the EORS variant

Applies when `(S == 1)`

```
EORS{<c>}{<q>}  {<Rd>, }<Rn>, #<const>
```

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant boolean setflags = (S == '1');
constant boolean a32 = TRUE;
constant bits(12) imm = imm12;
```

T1

15–11	10	9	8–5	4	3–0	15	14–12	11–8	7–0
1 1 1 1 0	i	0	0 1 0 0	S	Rn	0	imm3	Rd	imm8
			op1						

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Encoding for the EOR variant

Applies when `(S == 0)`

```
EOR{<c>}{<q>}  {<Rd>, }<Rn>, #<const>
```

Encoding for the EORS variant

Applies when `(S == 1 && Rd != 1111)`

```
EORS{<c>}{<q>}  {<Rd>, }<Rn>, #<const>
```

Decode for all variants of this encoding

```
if Rd == '1111' && S == '1' then SEE "TEQ (immediate)";
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant boolean setflags = (S == '1');
constant boolean a32 = FALSE;
constant bits(12) imm = i:imm3:imm8;
// Armv8-A removes UNPREDICTABLE for R13
if (d == 15 && !setflags) || n == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

For the “A1 EOR” and “A1 EORS” variants: is the general-purpose destination register, encoded in the “Rd” field. If omitted, this register is the same as `<Rn>`. Arm deprecates using the PC as the destination register, but if the PC is used:

- For the EOR variant, the instruction is a branch to the address calculated by the operation. This is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- For the EORS variant, the instruction performs an exception return, that restores PSTATE from SPSR_<current_mode>.

For the “T1 EOR” and “T1 EORS” variants: is the general-purpose destination register, encoded in the “Rd” field. If omitted, this register is the same as `<Rn>`.

<Rn>

For the “A1 EOR” and “A1 EORS” variants: is the general-purpose source register, encoded in the “Rn” field. The PC can be used, but this is deprecated.

For the “T1 EOR” and “T1 EORS” variants: is the general-purpose source register, encoded in the “Rn” field.

<const>

For the “A1 EOR” and “A1 EORS” variants: an immediate value. See Modified immediate constants in A32 instructions for the range of values.

For the “T1 EOR” and “T1 EORS” variants: an immediate value. See Modified immediate constants in T32 instructions for the range of values.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    bits(32) imm32;
    bit carry;
    (imm32, carry) = if a32 then A32ExpandImm_C(imm, PSTATE.C) else T32ExpandImm_C(imm, PSTATE.C);
    constant bits(32) result = R[n] EOR imm32;
    if d == 15 then          // Can only occur for A32 encoding
        if setflags then
            ALUExceptionReturn(result);
        else
            ALUWritePC(result);
    else
        R[d] = result;
        if setflags then
            PSTATE.N = result<31>;
            PSTATE.Z = IsZeroBit(result);
            PSTATE.C = carry;
            // PSTATE.V unchanged
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.50 EOR, EORS (register)

Bitwise Exclusive-OR (register) performs a bitwise exclusive-OR of a register value and an optionally-shifted register value, and writes the result to the destination register.

If the destination register is not the PC, the EORS variant of the instruction updates the condition flags based on the result.

The field descriptions for `<Rd>` identify the encodings where the PC is permitted as the destination register. Arm deprecates any use of these encodings. However, when the destination register is the PC:

- The EOR variant of the instruction is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- The EORS variant of the instruction performs an exception return without the use of the stack. In this case:
  - The PE branches to the address written to the PC, and restores PSTATE from SPSR_<current_mode>.
  - The PE checks SPSR_<current_mode> for an illegal return event. See Illegal return events from AArch32 state.
  - The instruction is UNDEFINED in Hyp mode.
  - The instruction is CONSTRAINED UNPREDICTABLE in User mode and System mode.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1 and T2).

A1

Encoding for the EOR, rotate right with extend variant

Applies when `(S == 0 && imm5 == 00000 && stype == 11)`

```
EOR{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, RRX
```

Encoding for the EOR, shift or rotate by value variant

Applies when `(S == 0 && !(imm5 == 00000 && stype == 11))`

```
EOR{<c>}{<q>}  {<Rd>, }<Rn>, <Rm> {, <shift> #<amount>}
```

Encoding for the EORS, rotate right with extend variant

Applies when `(S == 1 && imm5 == 00000 && stype == 11)`

```
EORS{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, RRX
```

Encoding for the EORS, shift or rotate by value variant

Applies when `(S == 1 && !(imm5 == 00000 && stype == 11))`

```
EORS{<c>}{<q>}  {<Rd>, }<Rn>, <Rm> {, <shift> #<amount>}
```

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean setflags = (S == '1');
SRType shift_t;
integer shift_n;
(shift_t, shift_n) = DecodeImmShift(stype, imm5);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T1

15–10	9–6	5–3	2–0
0 1 0 0 0 0	0 0 0 1	Rm	Rdn

op

Encoding

```
EOR<c>{<q>}  {<Rdn>, }<Rdn>, <Rm> // (InITBlock())
```

```
EORS{<q>}  {<Rdn>, }<Rdn>, <Rm> // (Outside IT block)
```

Decode for this encoding

```
constant integer d = UInt(Rdn);
constant integer n = UInt(Rdn);
constant integer m = UInt(Rm);
constant boolean setflags = !InITBlock();
constant SRType shift_t = SRType_LSL;
constant integer shift_n = 0;
```

T2

15–9	8–5	4	3–0	15	14–12	11–8	7 6	5 4	3–0
1 1 1 0 1 0 1	0 1 0 0	S	Rn	(0)	imm3	Rd	imm2	stype	Rm

op1

Encoding for the EOR, rotate right with extend variant

Applies when `(S == 0 && imm3 == 000 && imm2 == 00 && stype == 11)`

```
EOR{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, RRX
```

Encoding for the EOR, shift or rotate by value variant

Applies when `(S == 0 && !(imm3 == 000 && imm2 == 00 && stype == 11))`

```
EOR{<c>}{<q>}  {<Rd>, }<Rn>, <Rm> {, <shift> #<amount>}
```

```
EOR<c>.W  {<Rd>, }<Rn>, <Rm> // (Inside IT block, and <Rd>, <Rn>, <Rm> can be represented in T1)
```

Encoding for the EORS, rotate right with extend variant

Applies when `(S == 1 && imm3 == 000 && Rd != 1111 && imm2 == 00 && stype == 11)`

```
EORS{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, RRX
```

Encoding for the EORS, shift or rotate by value variant

Applies when `(S == 1 && Rd != 1111 && !(imm3 == 000 && imm2 == 00 && stype == 11))`

```
EORS{<c>}{<q>}  {<Rd>, }<Rn>, <Rm> {, <shift> #<amount>}
```

```
EORS.W  {<Rd>, }<Rn>, <Rm> // (Outside IT block, and <Rd>, <Rn>, <Rm> can be represented in T1)
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for all variants of this encoding

```
if Rd == '1111' && S == '1' then SEE "TEQ (register)";
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean setflags = (S == '1');
SRType shift_t;
integer shift_n;
(shift_t, shift_n) = DecodeImmShift(stype, imm3:imm2);
// Armv8-A removes UNPREDICTABLE for R13
if (d == 15 && !setflags) || n == 15 || m == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

For the “A1 EOR, rotate right with extend”, “A1 EOR, shift or rotate by value”, “A1 EORS, rotate right with extend”, and “A1 EORS, shift or rotate by value” variants: is the general-purpose destination register, encoded in the “Rd” field. If omitted, this register is the same as `<Rn>`. Arm deprecates using the PC as the destination register, but if the PC is used:

- For the EOR variant, the instruction is a branch to the address calculated by the operation. This is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- For the EORS variant, the instruction performs an exception return, that restores PSTATE from SPSR_<current_mode>.

For the “T2 EOR, rotate right with extend”, “T2 EOR, shift or rotate by value”, “T2 EORS, rotate right with extend”, and “T2 EORS, shift or rotate by value” variants: is the general-purpose destination register, encoded in the “Rd” field. If omitted, this register is the same as `<Rn>`.

<Rn>

For the “A1 EOR, rotate right with extend”, “A1 EOR, shift or rotate by value”, “A1 EORS, rotate right with extend”, and “A1 EORS, shift or rotate by value” variants: is the first general-purpose source register, encoded in the “Rn” field. The PC can be used, but this is deprecated.

For the “T2 EOR, rotate right with extend”, “T2 EOR, shift or rotate by value”, “T2 EORS, rotate right with extend”, and “T2 EORS, shift or rotate by value” variants: is the first general-purpose source register, encoded in the “Rn” field.

<Rm>

For the “A1 EOR, rotate right with extend”, “A1 EOR, shift or rotate by value”, “A1 EORS, rotate right with extend”, and “A1 EORS, shift or rotate by value” variants: is the second general-purpose source register, encoded in the “Rm” field. The PC can be used, but this is deprecated.

For the “T1”, “T2 EOR, rotate right with extend”, “T2 EOR, shift or rotate by value”, “T2 EORS, rotate right with extend”, and “T2 EORS, shift or rotate by value” variants: is the second general-purpose source register, encoded in the “Rm” field.

<shift>

Is the type of shift to be applied to the second source register, encoded in “stype”:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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

<amount>

For the “A1 EOR, shift or rotate by value” and “A1 EORS, shift or rotate by value” variants: is the shift amount, in the range 1 to 31 (when <shift> = LSL or ROR) or 1 to 32 (when <shift> = LSR or ASR) encoded in the “imm5” field as <amount> modulo 32.

For the “T2 EOR, shift or rotate by value” and “T2 EORS, shift or rotate by value” variants: is the shift amount, in the range 1 to 31 (when <shift> = LSL or ROR) or 1 to 32 (when <shift> = LSR or ASR), encoded in the “imm3:imm2” field as <amount> modulo 32.

<Rdn>

Is the first general-purpose source register and the destination register, encoded in the “Rdn” field.

In T32 assembly:

- Outside an IT block, if `EORS <Rd>, <Rn>, <Rd>` has `<Rd>` and `<Rn>` both in the range R0-R7, it is assembled using encoding T1 as though `EORS <Rd>, <Rn>` had been written
- Inside an IT block, if `EOR<c> <Rd>, <Rn>, <Rd>` has `<Rd>` and `<Rn>` both in the range R0-R7, it is assembled using encoding T1 as though `EOR<c> <Rd>, <Rn>` had been written.

To prevent either of these happening, use the `.W` qualifier.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    bits(32) shifted;
    bit carry;
    (shifted, carry) = Shift_C(R[m], shift_t, shift_n, PSTATE.C);
    constant bits(32) result = R[n] EOR shifted;
    if d == 15 then          // Can only occur for A32 encoding
        if setflags then
            ALUExceptionReturn(result);
        else
            ALUWritePC(result);
    else
        R[d] = result;
        if setflags then
            PSTATE.N = result<31>;
            PSTATE.Z = IsZeroBit(result);
            PSTATE.C = carry;
            // PSTATE.V unchanged
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.51 EOR, EORS (register-shifted register)

Bitwise Exclusive-OR (register-shifted register) performs a bitwise exclusive-OR of a register value and a register-shifted register value. It writes the result to the destination register, and can optionally update the condition flags based on the result.

31–28	27–24	23–21	20	19–16	15–12	11–8	7	6–5	4	3–0
!=1111 (cond)	0 0 0 0	0 0 1 (opc)	S	Rn	Rd	Rs	0	stype	1	Rm

Encoding for the Flag setting variant

Applies when `(S == 1)`

```
EORS{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, <shift> <Rs>
```

Encoding for the Not flag setting variant

Applies when `(S == 0)`

```
EOR{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, <shift> <Rs>
```

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant integer s = UInt(Rs);
constant boolean setflags = (S == '1');
constant SRType shift_t = DecodeRegShift(stype);
if d == 15 || n == 15 || m == 15 || s == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

Is the general-purpose destination register, encoded in the “Rd” field.

<Rn>

Is the first general-purpose source register, encoded in the “Rn” field.

<Rm>

Is the second general-purpose source register, encoded in the “Rm” field.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

<shift>

Is the type of shift to be applied to the second source register, encoded in “stype”:

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

<Rs>

Is the third general-purpose source register holding a shift amount in its bottom 8 bits, encoded in the “Rs” field.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant integer shift_n = UInt(R[s]<7:0>);
    bits(32) shifted;
    bit carry;
    (shifted, carry) = Shift_C(R[m], shift_t, shift_n, PSTATE.C);
    constant bits(32) result = R[n] EOR shifted;
    R[d] = result;
    if setflags then
        PSTATE.N = result<31>;
        PSTATE.Z = IsZeroBit(result);
        PSTATE.C = carry;
        // PSTATE.V unchanged
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.52 ERET

Exception Return.

The PE branches to the address held in the register holding the preferred return address, and restores PSTATE from SPSR_<current_mode>.

The register holding the preferred return address is:

- ELR_hyp, when executing in Hyp mode.
- LR, when executing in a mode other than Hyp mode, User mode, or System mode.

The PE checks SPSR_<current_mode> for an illegal return event. See Illegal return events from AArch32 state.

Exception Return is CONSTRAINED UNPREDICTABLE in User mode and System mode.

In Debug state, the T1 encoding of ERET executes the DRPS operation.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

31–28	27 26	25–23	22 21	20	19–8	7	6–4	3–0
!=1111 (cond)	0 0	0 1 0	1 1 (op0)	0	(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)	0	1 1 0	(1)(1)(1)(0)

Encoding

```
ERET{<c>}{<q>}
```

Decode for this encoding

Listing F5-37

```
// No additional decoding required
```

T1

15–13	12 11	10	9–6	5 4	3–0	15	14–12	11	10–8	7–0
1 1 1	1 0	0	1 1 1 1	0 1	1 1 1 0 (Rn)	1	0(0)0	(1)	(1)(1)(1)	0 0 0 0 0 0 0 0 (imm8)

Encoding

```
ERET{<c>}{<q>}
```

Decode for this encoding

Listing F5-38

```
if InITBlock() && !LastInITBlock() then UNPREDICTABLE; end
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

Operation

Listing F5-39

```
if ConditionPassed() then
    EncodingSpecificOperations();
    if !Halted() then
        if PSTATE.M IN {M32_User,M32_System} then
            UNPREDICTABLE;                        // UNDEFINED or NOP
        else
            constant bits(32) new_pc_value = if PSTATE.EL == EL2 then ELR_hyp else R[14];
            AArch32.ExceptionReturn(new_pc_value, SPSR_curr[]);
        end
    else                                          // Perform DRPS operation in Debug state
        if PSTATE.M == M32_User then
            UNDEFINED;
        elsif PSTATE.M == M32_System then
            UNPREDICTABLE;                        // UNDEFINED or NOP
        else
            SynchronizeContext();
            DebugRestorePSR();
        end
    end
end
```

CONSTRAINED UNPREDICTABLE behavior

If `PSTATE.M IN {M32_User,M32_System}`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.53 ESB

Error Synchronization Barrier is an error synchronization event that might also update DISR and VDISR. This instruction can be used at all Exception levels and in Debug state.

In Debug state, this instruction behaves as if SError interrupts are masked at all Exception levels. For more information, see RAS PE architecture and Arm® Reliability, Availability, and Serviceability (RAS) System Architecture, for A-profile architecture (ARM IHI 0100).

If FEAT_RAS is not implemented, this instruction executes as a `NOP`.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

(FEAT_RAS)

Encoding

```
ESB{<c>}{<q>}
```

Decode for this encoding

Listing F5-40

```
if !IsFeatureImplemented(FEAT_RAS) then ExecuteAsNOP(); end
if cond != '1110' then UNPREDICTABLE; end // ESB must be encoded with AL condition
```

CONSTRAINED UNPREDICTABLE behavior

If `cond != '1110'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction executes unconditionally.
- The instruction executes conditionally.

T1

(FEAT_RAS)

Encoding

```
ESB{<c>}{<q>}
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for this encoding

Listing F5-41

```
if !IsFeatureImplemented(FEAT_RAS) then ExecuteAsNOP(); end
if InITBlock() then UNPREDICTABLE; end
```

CONSTRAINED UNPREDICTABLE behavior

If `InITBlock()`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction executes unconditionally.
- The instruction executes conditionally.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

Operation

Listing F5-42

```
if ConditionPassed() then
    EncodingSpecificOperations();

    SynchronizeErrors();
    AArch32.ESBOperation();
    if PSTATE.EL IN {EL0, EL1} && EL2Enabled() then
        AArch32.vESBOperation();
    elsif IsFeatureImplemented(FEAT_E3DSE) && PSTATE.EL != EL3 && !ELUsingAArch32(EL3) then
        AArch64.dESBOperation();
    end
    TakeUnmaskedSErrorInterrupts();
end
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.54 HLT

Halting Breakpoint

Halting breakpoint causes a software breakpoint to occur.

Halting breakpoint is always unconditional, even inside an IT block.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

31–28	27 26	25 23	22 21	20	19–8	7	6 4	3–0
!=1111 (cond)	0 0	0 1 0	0 0 (opc)	0	imm12	0	1 1 1	imm4

Encoding

```
HLT{<q>}  {#}<imm>
```

Decode for this encoding

Listing F5-43

```
if EDSCR.HDE == '0' || !HaltingAllowed() then UNDEFINED; end
if cond != '1110' then UNPREDICTABLE; end // HLT must be encoded with AL condition
```

CONSTRAINED UNPREDICTABLE behavior

If `cond != '1110'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction executes unconditionally.
- The instruction executes conditionally.

T1

15–12	11–8	7 6	5–0
1 0 1 1	1 0 1 0	1 0	imm6

Encoding

```
HLT{<q>}  {#}<imm>
```

Decode for this encoding

Listing F5-44

```
if EDSCR.HDE == '0' || !HaltingAllowed() then UNDEFINED; end
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<q>

See Standard assembler syntax fields. An `HLT` instruction must be unconditional.

<imm>

For the "A1" variant: is a 16-bit unsigned immediate, in the range 0 to 65535, encoded in the "imm12:imm4" field. This value is for assembly and disassembly only. It is ignored by the PE, but can be used by a debugger to store more information about the halting breakpoint.

For the "T1" variant: is a 6-bit unsigned immediate, in the range 0 to 63, encoded in the "imm6" field. This value is for assembly and disassembly only. It is ignored by the PE, but can be used by a debugger to store more information about the halting breakpoint.

Operation

Listing F5-45

```
EncodingSpecificOperations();
constant boolean is_async = FALSE;
constant FaultRecord fault = NoFault();
Halt(DebugHalt_HaltInstruction, is_async, fault);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.55 HVC

Hypervisor Call causes a Hypervisor Call exception. For more information, see Hypervisor Call (HVC) exception. Software executing at EL1 can use this instruction to call the hypervisor to request a service.

The HVC instruction is UNDEFINED:

- When EL3 is implemented and using AArch64, and SCR_EL3.HCE is set to 0.
- In Non-secure EL1 modes when EL3 is implemented and using AArch32, and SCR.HCE is set to 0.
- When EL3 is not implemented and either HCR_EL2.HCD is set to 1 or HCR.HCD is set to 1.
- When EL2 is not implemented.
- In Secure state, if EL2 is not enabled in the current Security state.
- In User mode.

The HVC instruction is CONSTRAINED UNPREDICTABLE in Hyp mode when EL3 is implemented and using AArch32, and SCR.HCE is set to 0.

On executing an HVC instruction, the HSR, Hyp Syndrome Register reports the exception as a Hypervisor Call exception, using the EC value 0x12, and captures the value of the immediate argument, see Use of the HSR.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding

```
HVC{<q>}  {#}<imm16>
```

Decode for this encoding

Listing F5-46

```
if cond != '1110' then UNPREDICTABLE; end
constant bits(16) imm16 = imm12:imm4;
```

CONSTRAINED UNPREDICTABLE behavior

If `cond != '1110'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction executes unconditionally.
- The instruction executes conditionally.

T1

Encoding

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
HVC{<q>}  {#}<imm16>
```

Decode for this encoding

Listing F5-47

```
constant bits(16) imm16 = imm4:imm12;
if InITBlock() then UNPREDICTABLE; end
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<q>

See Standard assembler syntax fields. An `HVC` instruction must be unconditional.

<imm16>

For the "A1" variant: is a 16-bit unsigned immediate, in the range 0 to 65535, encoded in the "imm12:imm4" field. This value is for assembly and disassembly only. It is reported in the HSR but otherwise is ignored by hardware. An HVC handler might interpret imm16, for example to determine the required service.

For the "T1" variant: is a 16-bit unsigned immediate, in the range 0 to 65535, encoded in the "imm4:imm12" field. This value is for assembly and disassembly only. It is reported in the HSR but otherwise is ignored by hardware. An HVC handler might interpret imm16, for example to determine the required service.

Operation

Listing F5-48

```
EncodingSpecificOperations();
if PSTATE.EL IN {EL0, EL3} || !EL2Enabled() then
    UNDEFINED;
end

bit hvc_enable;
if HaveEL(EL3) then
    if ELUsingAArch32(EL3) then
        if SCR.HCE == '0' && PSTATE.EL == EL2 then UNPREDICTABLE; end
        hvc_enable = SCR.HCE;
    else
        hvc_enable = SCR_EL3.HCE;
    end
else
    hvc_enable = if ELUsingAArch32(EL2) then NOT(HCR.HCD) else NOT(HCR_EL2.HCD);
end

if hvc_enable == '0' then
    UNDEFINED;
else
    AArch32.CallHypervisor(imm16);
end
```

CONSTRAINED UNPREDICTABLE behavior

If `ELUsingAArch32(EL3) && SCR.HCE == '0' && PSTATE.EL == EL2`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.56 ISB

Instruction Synchronization Barrier flushes the pipeline in the PE and is a context synchronization event. For more information, see Instruction Synchronization Barrier (ISB).

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

31–28	27	26	25–21	20	19–8	7–4	3–0
1 1 1 1	0	1	0 1 0 1 1	1	(1) (1) (1) (1) (1) (1) (1) (1) (0) (0) (0) (0)	0 1 1 0	option
						opcode	

Encoding

```
ISB{<c>}{<q>}  {<option>}
```

Decode for this encoding

Listing F5-49

```
// No additional decoding required
```

T1

15–13	12 11	10	9–6	5 4	3–0	15	14–12	11	10–8	7–4	3–0
1 1 1	1 0	0	1 1 1 0	1 1	(1) (1) (1) (1)	1	0 (0) 0	(1)	(1) (1) (1)	0 1 1 0	option
										opc	

Encoding

```
ISB{<c>}{<q>}  {<option>}
```

Decode for this encoding

Listing F5-50

```
// No additional decoding required
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

For the “A1” variant: see Standard assembler syntax fields. Must be `AL` or omitted.

For the “T1” variant: see Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

<option>

Specifies an optional limitation on the barrier operation. Values are:

SY Full system barrier operation, encoded as option = `0b1111`. Can be omitted.

All other encodings of option are reserved. The corresponding instructions execute as full system barrier operations, but must not be relied upon by software.

Operation

Listing F5-51

```
if ConditionPassed() then
    EncodingSpecificOperations();
    InstructionSynchronizationBarrier();
end
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.57 IT

If-Then makes up to four following instructions (the IT block) conditional. The conditions for the instructions in the IT block are the same as, or the inverse of, the condition the `IT` instruction specifies for the first instruction in the block.

The `IT` instruction itself does not affect the condition flags, but the execution of the instructions in the IT block can change the condition flags.

16-bit instructions in the IT block, other than `CMP`, `CMN` and `TST`, do not set the condition flags. An `IT` instruction with the `AL` condition can change the behavior without conditional execution.

The architecture permits exception return to an instruction in the IT block only if the restoration of the CPSR restores PSTATE.IT to a state consistent with the conditions specified by the `IT` instruction. Any other exception return to an instruction in an IT block is UNPREDICTABLE. Any branch to a target instruction in an IT block is not permitted, and if such a branch is made it is UNPREDICTABLE what condition is used when executing that target instruction and any subsequent instruction in the IT block.

Many uses of the IT instruction are deprecated for performance reasons, and an implementation might include ITD controls that can disable those uses of IT, making them UNDEFINED.

For more information see Conditional execution and Conditional instructions. The first of these sections includes more information about the ITD controls.

15–12	11–8	7–4	3–0
1 0 1 1	1 1 1 1	firstcond	!=0000

mask

Encoding

```
IT{<x>{<y>{<z>}}}{<q>}  <cond>
```

Decode for this encoding

```
if mask == '0000' then SEE "Related encodings";
if firstcond == '1111' || (firstcond == '1110' && BitCount(mask) != 1) then UNPREDICTABLE;
if InITBlock() then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `firstcond == '1111' || (firstcond == '1110' && BitCount(mask) != 1)`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The ‘1111’ condition is treated as being the same as the ‘1110’ condition, meaning always, and the ITSTATE state machine is progressed in the same way as for any other cond_base value.

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Related encodings: Miscellaneous 16-bit instructions.

Assembler Symbols

<x>

The condition for the second instruction in the IT block. If omitted, the “mask” field is set to `0b1000`. If present it is encoded in the “mask[3]” field:

T firstcond[0]
E NOT firstcond[0]

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

<y>

The condition for the third instruction in the IT block. If omitted and `<x>` is present, the “mask[2:0]” field is set to `0b100`. If `<y>` is present it is encoded in the “mask[2]” field:

T firstcond[0]
E NOT firstcond[0]

<z>

The condition for the fourth instruction in the IT block. If omitted and `<y>` is present, the “mask[1:0]” field is set to `0b10`. If `<z>` is present, the “mask[0]” field is set to 1, and it is encoded in the “mask[1]” field:

T firstcond[0]
E NOT firstcond[0]

<q>

See Standard assembler syntax fields.

<cond>

The condition for the first instruction in the IT block, encoded in the “firstcond” field. See Condition codes for the range of conditions available, and the encodings.

The conditions specified in an `IT` instruction must match those specified in the syntax of the instructions in its IT block. When assembling to A32 code, assemblers check `IT` instruction syntax for validity but do not generate assembled instructions for them. See Conditional instructions.

Operation

```
EncodingSpecificOperations();
AArch32.CheckITEnabled(mask);
PSTATE.IT<7:0> = firstcond:mask;
ShouldAdvanceIT = FALSE;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.58 LDA

Load-Acquire Word loads a word from memory and writes it to a register. The instruction also has memory ordering semantics as described in Load-Acquire, Store-Release

For more information about support for shared memory see Synchronization and semaphores. For information about memory accesses see Memory accesses.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding

```
LDA{<c>}{<q>}  <Rt>, [<Rn>]
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
if t == 15 || n == 15 then UNPREDICTABLE;
```

T1

Encoding

```
LDA{<c>}{<q>}  <Rt>, [<Rn>]
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
if t == 15 || n == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

Is the general-purpose register to be transferred, encoded in the “Rt” field.

<Rn>

Is the general-purpose base register, encoded in the “Rn” field.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) address = R[n];
    R[t] = MemO[address, 4];
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.59 LDAB

Load-Acquire Byte loads a byte from memory, zero-extends it to form a 32-bit word and writes it to a register. The instruction also has memory ordering semantics as described in Load-Acquire, Store-Release.

For more information about support for shared memory see Synchronization and semaphores. For information about memory accesses see Memory accesses.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding

```
LDAB{<c>}{<q>}  <Rt>, [<Rn>]
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
if t == 15 || n == 15 then UNPREDICTABLE;
```

T1

Encoding

```
LDAB{<c>}{<q>}  <Rt>, [<Rn>]
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
if t == 15 || n == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

Is the general-purpose register to be transferred, encoded in the “Rt” field.

<Rn>

Is the general-purpose base register, encoded in the “Rn” field.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) address = R[n];
    R[t] = ZeroExtend(MemO[address, 1], 32);
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.60 LDAEX

Load-Acquire Exclusive Word loads a word from memory, writes it to a register and:

- If the address has the Shared Memory attribute, marks the physical address as exclusive access for the executing PE in a global monitor.
- Causes the executing PE to indicate an active exclusive access in the local monitor.

The instruction also has memory ordering semantics as described in Load-Acquire, Store-Release.

For more information about support for shared memory see Synchronization and semaphores. For information about memory accesses see Memory accesses.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding

```
LDAEX{<c>}{<q>}  <Rt>, [<Rn>]
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
if t == 15 || n == 15 then UNPREDICTABLE;
```

T1

Encoding

```
LDAEX{<c>}{<q>}  <Rt>, [<Rn>]
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
if t == 15 || n == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

Is the general-purpose register to be transferred, encoded in the “Rt” field.

<Rn>

Is the general-purpose base register, encoded in the “Rn” field.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) address = R[n];
    AArch32.SetExclusiveMonitors(address, 4);
    R[t] = MemO[address, 4];
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.61 LDAEXB

Load-Acquire Exclusive Byte loads a byte from memory, zero-extends it to form a 32-bit word, writes it to a register and:

- If the address has the Shared Memory attribute, marks the physical address as exclusive access for the executing PE in a global monitor.
- Causes the executing PE to indicate an active exclusive access in the local monitor.

The instruction also has memory ordering semantics as described in Load-Acquire, Store-Release.

For more information about support for shared memory see Synchronization and semaphores. For information about memory accesses see Memory accesses.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding

```
LDAEXB{<c>}{<q>}  <Rt>, [<Rn>]
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
if t == 15 || n == 15 then UNPREDICTABLE;
```

T1

Encoding

```
LDAEXB{<c>}{<q>}  <Rt>, [<Rn>]
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
if t == 15 || n == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

Is the general-purpose register to be transferred, encoded in the “Rt” field.

<Rn>

Is the general-purpose base register, encoded in the “Rn” field.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) address = R[n];
    AArch32.SetExclusiveMonitors(address, 1);
    R[t] = ZeroExtend(MemO[address, 1], 32);
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.62 LDAEXD

Load-Acquire Exclusive Doubleword loads a doubleword from memory, writes it to two registers and:

- If the address has the Shared Memory attribute, marks the physical address as exclusive access for the executing PE in a global monitor
- Causes the executing PE to indicate an active exclusive access in the local monitor.

The instruction also acts as a barrier instruction with the ordering requirements described in Load-Acquire, Store-Release.

For more information about support for shared memory see Synchronization and semaphores. For information about memory accesses see Memory accesses.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

31–28	27	26	25	24	23	22	21	20	19–16	15–12	11	10	9	8	7	6	5	4	3–0
!=1111	0	0	0	1	1	0	1	1	Rn	Rt	(1)	(1)	1	0	1	0	0	1	(1)(1)(1)(1)
cond					size			L					ex	ord					xRt

Encoding

```
LDAEXD{<c>}{<q>}  <Rt>, <Rt2>, [<Rn>]
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer t2 = t + 1;
constant integer n = UInt(Rn);
if Rt<0> == '1' || t2 == 15 || n == 15 then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `Rt<0> == '1'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction executes with the additional decode: t<0> = ‘0’.
- The instruction executes with the additional decode: t2 = t.
- The instruction executes as described, with no change to its behavior and no additional side effects.

If `Rt == '1110'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction is handled as described in Using R15.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T1

Encoding

```
LDAEXD{<c>}{<q>}  <Rt>, <Rt2>, [<Rn>]
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer t2 = UInt(Rt2);
constant integer n = UInt(Rn);
if t == 15 || t2 == 15 || t == t2 || n == 15 then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `t == t2`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The load instruction executes but the destination register takes an UNKNOWN value.

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

For the “A1” variant: is the first general-purpose register to be transferred, encoded in the “Rt” field. `<Rt>` must be even-numbered and not R14.

For the “T1” variant: is the first general-purpose register to be transferred, encoded in the “Rt” field.

<Rt2>

For the “A1” variant: is the second general-purpose register to be transferred. `<Rt2>` must be `<R(t+1)>`.

For the “T1” variant: is the second general-purpose register to be transferred, encoded in the “Rt2” field.

<Rn>

Is the general-purpose base register, encoded in the “Rn” field.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) address = R[n];
    AArch32.SetExclusiveMonitors(address, 8);
    constant bits(64) value = MemO[address, 8];
    // Extract words from 64-bit loaded value such that R[t] is
    // loaded from address and R[t2] from address+4.
    R[t]  = if BigEndian(AccessType_GPR) then value<63:32> else value<31:0>;
    R[t2] = if BigEndian(AccessType_GPR) then value<31:0>  else value<63:32>;
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.63 LDAEXH

Load-Acquire Exclusive Halfword loads a halfword from memory, zero-extends it to form a 32-bit word, writes it to a register and:

- If the address has the Shared Memory attribute, marks the physical address as exclusive access for the executing PE in a global monitor.
- Causes the executing PE to indicate an active exclusive access in the local monitor.

The instruction also has memory ordering semantics as described in Load-Acquire, Store-Release.

For more information about support for shared memory see Synchronization and semaphores. For information about memory accesses see Memory accesses.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding

```
LDAEXH{<c>}{<q>}  <Rt>, [<Rn>]
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
if t == 15 || n == 15 then UNPREDICTABLE;
```

T1

Encoding

```
LDAEXH{<c>}{<q>}  <Rt>, [<Rn>]
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
if t == 15 || n == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

Is the general-purpose register to be transferred, encoded in the “Rt” field.

<Rn>

Is the general-purpose base register, encoded in the “Rn” field.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) address = R[n];
    AArch32.SetExclusiveMonitors(address, 2);
    R[t] = ZeroExtend(MemO[address, 2], 32);
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.64 LDAH

Load-Acquire Halfword loads a halfword from memory, zero-extends it to form a 32-bit word and writes it to a register. The instruction also has memory ordering semantics as described in Load-Acquire, Store-Release.

For more information about support for shared memory see Synchronization and semaphores. For information about memory accesses see Memory accesses.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding

```
LDAH{<c>}{<q>}  <Rt>, [<Rn>]
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
if t == 15 || n == 15 then UNPREDICTABLE;
```

T1

Encoding

```
LDAH{<c>}{<q>}  <Rt>, [<Rn>]
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
if t == 15 || n == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

Is the general-purpose register to be transferred, encoded in the “Rt” field.

<Rn>

Is the general-purpose base register, encoded in the “Rn” field.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) address = R[n];
    R[t] = ZeroExtend(MemO[address, 2], 32);
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.65 LDC (immediate)

Load data to System register (immediate) calculates an address from a base register value and an immediate offset, loads a word from memory, and writes it to the DBGDTRTXint System register. It can use offset, post-indexed, pre-indexed, or unindexed addressing. For information about memory accesses, see Memory accesses.

In an implementation that includes EL2, the permitted `LDC` access to DBGDTRTXint can be trapped to Hyp mode, meaning that an attempt to execute an `LDC` instruction in a Non-secure mode other than Hyp mode, that would be permitted in the absence of the Hyp trap controls, generates a Hyp Trap exception. For more information, see HDCR.TDA.

For simplicity, the `LDC` pseudocode does not show this possible trap to Hyp mode.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding for the Offset variant

Applies when `(P == 1 && W == 0)`

```
LDC{<c>}{<q>}  p14, c5, [<Rn>{, #{+/-}<imm>}]
```

Encoding for the Post-indexed variant

Applies when `(P == 0 && W == 1)`

```
LDC{<c>}{<q>}  p14, c5, [<Rn>], #{+/-}<imm>
```

Encoding for the Pre-indexed variant

Applies when `(P == 1 && W == 1)`

```
LDC{<c>}{<q>}  p14, c5, [<Rn>, #{+/-}<imm>]!
```

Encoding for the Unindexed variant

Applies when `(P == 0 && U == 1 && W == 0)`

```
LDC{<c>}{<q>}  p14, c5, [<Rn>], <option>
```

Decode for all variants of this encoding

```
if Rn == '1111' then SEE "LDC (literal)";
if P == '0' && U == '0' && W == '0' then UNDEFINED;
constant integer n = UInt(Rn);
constant integer cp = 14;
constant bits(32) imm32 = ZeroExtend(imm8:'00', 32);
constant boolean index = (P == '1');
constant boolean add = (U == '1');
constant boolean wback = (W == '1');
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T1

Encoding for the Offset variant

Applies when `(P == 1 && W == 0)`

```
LDC{<c>}{<q>}  p14, c5, [<Rn>{, #{+/-}<imm>}]
```

Encoding for the Post-indexed variant

Applies when `(P == 0 && W == 1)`

```
LDC{<c>}{<q>}  p14, c5, [<Rn>], #{+/-}<imm>
```

Encoding for the Pre-indexed variant

Applies when `(P == 1 && W == 1)`

```
LDC{<c>}{<q>}  p14, c5, [<Rn>, #{+/-}<imm>]!
```

Encoding for the Unindexed variant

Applies when `(P == 0 && U == 1 && W == 0)`

```
LDC{<c>}{<q>}  p14, c5, [<Rn>], <option>
```

Decode for all variants of this encoding

```
if Rn == '1111' then SEE "LDC (literal)";
if P == '0' && U == '0' && W == '0' then UNDEFINED;
constant integer n = UInt(Rn);
constant integer cp = 14;
constant bits(32) imm32 = ZeroExtend(imm8:'00', 32);
constant boolean index = (P == '1');
constant boolean add = (U == '1');
constant boolean wback = (W == '1');
```

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rn>

Is the general-purpose base register, encoded in the “Rn” field. If the PC is used, see LDC (literal).

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

+/-

Specifies the offset is added to or subtracted from the base register, defaulting to + if omitted and encoded in “U”:

U	+/-
0	-
1	+

<imm>

Is the immediate offset used for forming the address, a multiple of 4 in the range 0-1020, defaulting to 0 and encoded in the “imm8” field, as <imm>/4.

<option>

Is an 8-bit immediate, in the range 0 to 255 enclosed in { }, encoded in the “imm8” field. The value of this field is ignored when executing this instruction.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) offset_addr = if add then (R[n] + imm32) else (R[n] - imm32);
    constant bits(32) address = if index then offset_addr else R[n];

    // System register write to DBGDTRTXint.
    AArch32.SysRegWriteM(cp, ThisInstr(), address);

    if wback then R[n] = offset_addr;
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.66 LDC (literal)

Load data to System register (literal) calculates an address from the PC value and an immediate offset, loads a word from memory, and writes it to the DBGDTRTXint System register. For information about memory accesses, see Memory accesses.

In an implementation that includes EL2, the permitted `LDC` access to DBGDTRTXint can be trapped to Hyp mode, meaning that an attempt to execute an `LDC` instruction in a Non-secure mode other than Hyp mode, that would be permitted in the absence of the Hyp trap controls, generates a Hyp Trap exception. For more information, see HDCR.TDA.

For simplicity, the `LDC` pseudocode does not show this possible trap to Hyp mode.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding

Applies when `(!(P == '0' && U == '0' && W == '0'))`

```
LDC{<c>}{<q>}  p14, c5, <label> // (Preferred syntax)
```

```
LDC{<c>}{<q>}  p14, c5, [PC, #{+/-}<imm>]
```

```
LDC{<c>}{<q>}  p14, c5, [PC], <option>
```

Decode for this encoding

```
if P == '0' && U == '0' && W == '0' then UNDEFINED;
constant boolean index = (P == '1');
constant boolean add = (U == '1');
constant integer cp = 14;
constant bits(32) imm32 = ZeroExtend(imm8:'00', 32);
if W == '1' || (P == '0' && CurrentInstrSet() != InstrSet_A32) then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `W == '1'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction executes without writeback of the base address.
- The instruction uses the addressing mode described in the equivalent immediate offset instruction.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T1

Encoding

Applies when `(!(P == '0' && U == '0' && W == '0'))`

```
LDC{<c>}{<q>}  p14, c5, <label> // (Preferred syntax)
```

```
LDC{<c>}{<q>}  p14, c5, [PC, #{+/-}<imm>]
```

Decode for this encoding

```
if P == '0' && U == '0' && W == '0' then UNDEFINED;
constant boolean index = (P == '1');
constant boolean add = (U == '1');
constant integer cp = 14;
constant bits(32) imm32 = ZeroExtend(imm8:'00', 32);
if W == '1' || (P == '0' && CurrentInstrSet() != InstrSet_A32) then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `W == '1' || P == '0'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction executes without writeback of the base address.
- The instruction executes as `LDC` (immediate) with writeback to the PC. The instruction is handled as described in Using R15.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<label>

The label of the literal data item that is to be loaded into `<Rt>`. The assembler calculates the required value of the offset from the `Align(PC, 4)` value of the instruction to this label. Permitted values of the offset are multiples of 4 in the range -1020 to 1020.

If the offset is zero or positive, `imm32` is equal to the offset and `add` == `TRUE` (encoded as U == 1).

If the offset is negative, `imm32` is equal to minus the offset and `add` == `FALSE (encoded as U == 0)`.

+/-

Specifies the offset is added to or subtracted from the base register, defaulting to + if omitted and encoded in “U”:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

U	+/-
0	-
1	+

<imm>

Is the immediate offset used for forming the address, a multiple of 4 in the range 0-1020, defaulting to 0 and encoded in the “imm8” field, as <imm>/4.

<option>

Is an 8-bit immediate, in the range 0 to 255 enclosed in { }, encoded in the “imm8” field. The value of this field is ignored when executing this instruction.

The alternative syntax permits the addition or subtraction of the offset and the immediate offset to be specified separately, including permitting a subtraction of 0 that cannot be specified using the normal syntax. For more information, see Use of labels in UAL instruction syntax.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) offset_addr = (if add then (Align(PC32,4) + imm32)
                                     else (Align(PC32,4) - imm32));
    constant bits(32) address = if index then offset_addr else Align(PC32,4);

    // System register write to DBGDTRTXint.
    AArch32.SysRegWriteM(cp, ThisInstr(), address);
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.67 LDM, LDMIA, LDMFD

Load Multiple (Increment After, Full Descending) loads multiple registers from consecutive memory locations using an address from a base register. The consecutive memory locations start at this address, and the address just above the highest of those locations can optionally be written back to the base register.

The lowest-numbered register is loaded from the lowest memory address, through to the highest-numbered register from the highest memory address. See also Encoding of lists of general-purpose registers and the PC.

Armv8.2 permits the deprecation of some Load Multiple ordering behaviors in AArch32 state, for more information see FEAT_LSMAOC. The registers loaded can include the PC, causing a branch to a loaded address. This is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC. Related system instructions are LDM (User registers) and LDM (exception return).

This instruction is used by the alias POP (multiple registers).

It has encodings from the following instruction sets: A32 (A1) and T32 (T1 and T2).

A1

Encoding

```
LDM{IA}{<c>}{<q>}  <Rn>{!}, <registers> // (Preferred syntax)
```

```
LDMFD{<c>}{<q>}  <Rn>{!}, <registers> // (Alternate syntax, Full Descending stack)
```

Decode for this encoding

```
constant integer n = UInt(Rn);
constant bits(16) registers = register_list;
constant boolean wback = (W == '1');
if n == 15 || BitCount(registers) < 1 then UNPREDICTABLE;
if wback && registers<n> == '1' then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `BitCount(registers) < 1`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction executes as `LDM` with the same addressing mode but targeting an unspecified set of registers. These registers might include R15. If the instruction specifies writeback, the modification to the base address on writeback might differ from the number of registers loaded.

If `wback && registers<n> == '1'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such an instruction, the base address might be corrupted so that the instruction cannot be repeated.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T1

Encoding

```
LDM{IA}{<c>}{<q>}  <Rn>{!}, <registers> // (Preferred syntax)
```

```
LDMFD{<c>}{<q>}  <Rn>{!}, <registers> // (Alternate syntax, Full Descending stack)
```

Decode for this encoding

```
constant integer n = UInt(Rn);
constant bits(16) registers = '00000000':register_list;
constant boolean wback = (registers<n> == '0');
if BitCount(registers) < 1 then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `BitCount(registers) < 1`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction executes as `LDM` with the same addressing mode but targeting an unspecified set of registers. These registers might include R15. If the instruction specifies writeback, the modification to the base address on writeback might differ from the number of registers loaded.

T2

Encoding

```
LDM{IA}{<c>}.W  <Rn>{!}, <registers> // (Preferred syntax, if <Rn>, '!' and <registers> can be
    ↪represented in T1)
```

```
LDMFD{<c>}.W  <Rn>{!}, <registers> // (Alternate syntax, Full Descending stack, if <Rn>, '!' and
    ↪<registers> can be represented in T1)
```

```
LDM{IA}{<c>}{<q>}  <Rn>{!}, <registers> // (Preferred syntax)
```

```
LDMFD{<c>}{<q>}  <Rn>{!}, <registers> // (Alternate syntax, Full Descending stack)
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for this encoding

```
constant integer n = UInt(Rn);
constant bits(16) registers = P:M:register_list;
constant boolean wback = (W == '1');
if n == 15 || BitCount(registers) < 2 || (P == '1' && M == '1') then UNPREDICTABLE;
if wback && registers<n> == '1' then UNPREDICTABLE;
if registers<13> == '1' then UNPREDICTABLE;
if registers<15> == '1' && InITBlock() && !LastInITBlock() then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `BitCount(registers) < 1`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction executes as `LDM` with the same addressing mode but targeting an unspecified set of registers. These registers might include R15. If the instruction specifies writeback, the modification to the base address on writeback might differ from the number of registers loaded.

If `wback && registers<n> == '1'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such an instruction, the base address might be corrupted so that the instruction cannot be repeated.

If `BitCount(registers) == 1`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction loads a single register using the specified addressing modes.
- The instruction executes as `LDM` with the same addressing mode but targeting an unspecified set of registers. These registers might include R15.

If `registers<13> == '1'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction performs all of the loads using the specified addressing mode, but R13 is UNKNOWN.

If `P == '1' && M == '1'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction loads the register list and either R14 or R15, both R14 and R15, or neither of these registers.

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

IA

Is an optional suffix for the Increment After form.

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rn>

Is the general-purpose base register, encoded in the “Rn” field.

!

For the “A1” and “T2” variants: the address adjusted by the size of the data loaded is written back to the base register. If specified, it is encoded in the “W” field as 1, otherwise this field defaults to 0.

For the “T1” variant: the address adjusted by the size of the data loaded is written back to the base register. It is omitted if `<Rn>` is included in `<registers>`, otherwise it must be present.

<registers>

For the “A1” variant: is a list of one or more registers to be loaded, separated by commas and surrounded by { and }.

The PC can be in the list.

Arm deprecates using these instructions with both the LR and the PC in the list.

For the “T1” variant: is a list of one or more registers to be loaded, separated by commas and surrounded by { and }. The registers in the list must be in the range R0-R7, encoded in the “register_list” field.

For the “T2” variant: is a list of one or more registers to be loaded, separated by commas and surrounded by { and }. The registers in the list must be in the range R0-R12, encoded in the “register_list” field, and can optionally contain one of the LR or the PC. If the LR is in the list, the “M” field is set to 1, otherwise it defaults to 0. If the PC is in the list, the “P” field is set to 1, otherwise it defaults to 0.

If the PC is in the list:

- The LR must not be in the list.
- The instruction must be either outside any IT block, or the last instruction in an IT block.

Alias Conditions

Alias	Of variant	Is preferred when
`POP (multiple registers)`	`A1`	`W == '1' && Rn == '1101' && BitCount(register_list) > 1`
`POP (multiple registers)`	`T2`	`W == '1' && Rn == '1101' && BitCount(P : M : register_list) > 1`
`POP (multiple registers)`	`T2`	`BitCount(P : M : register_list) > 1`

Operation

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if ConditionPassed() then
    EncodingSpecificOperations();
    bits(32) address = R[n];
    bits(32) data;
    for i = 0 to 14
        if registers<i> == '1' then
            if i != n then
                R[i] = MemS[address,4];
            else
                data = MemS[address,4];
            address = address + 4;
    if registers<15> == '1' then
        LoadWritePC(MemS[address,4]);

    if wback && registers<n> == '1' then
        R[n] = bits(32) UNKNOWN;
    if wback && registers<n> == '0' then
        R[n] = R[n] + 4*BitCount(registers);
    if !wback && registers<n> == '1' then
        R[n] = data;
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.68 LDM (exception return)

Load Multiple (exception return) loads multiple registers from consecutive memory locations using an address from a base register. The SPSR of the current mode is copied to the CPSR. An address adjusted by the size of the data loaded can optionally be written back to the base register.

The registers loaded include the PC. The word loaded for the PC is treated as an address and a branch occurs to that address.

The PE checks the encoding that is copied to the CPSR for an illegal return event. See Illegal return events from AArch32 state.

Load Multiple (exception return) is:

- UNDEFINED in Hyp mode.
- UNPREDICTABLE in debug state, and in User mode and System mode.

Encoding

```
LDM{<amode>}{<c>}{<q>}  <Rn>{!}, <registers_with_pc>^
```

Decode for this encoding

```
constant integer n = UInt(Rn);
constant bits(15) registers = register_list;
constant boolean wback = (W == '1');
constant boolean increment = (U == '1');
constant boolean wordhigher = (P == U);
if n == 15 then UNPREDICTABLE;
if wback && registers<n> == '1' then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `wback && registers<n> == '1'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction performs all the loads using the specified addressing mode and the content of the register being written back is UNKNOWN. In addition, if an exception occurs during the execution of this instruction, the base address might be corrupted so that the instruction cannot be repeated.

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<amode>

is one of:

DA Decrement After. The consecutive memory addresses end at the address in the base register. Encoded as P = 0, U = 0.

FA Full Ascending. For this instruction, a synonym for DA.

DB Decrement Before. The consecutive memory addresses end one word below the address in the base register. Encoded as P = 1, U = 0.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

EA Empty Ascending. For this instruction, a synonym for `DB`.

IA Increment After. The consecutive memory addresses start at the address in the base register. This is the default. Encoded as P = 0, U = 1.

FD Full Descending. For this instruction, a synonym for `IA`.

IB Increment Before. The consecutive memory addresses start one word above the address in the base register. Encoded as P = 1, U = 1.

ED Empty Descending. For this instruction, a synonym for `IB`.

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rn>

Is the general-purpose base register, encoded in the “Rn” field.

!

The address adjusted by the size of the data loaded is written back to the base register. If specified, it is encoded in the “W” field as 1, otherwise this field defaults to 0.

<registers_with_pc>

Is a list of one or more registers, separated by commas and surrounded by { and }. It specifies the set of registers to be loaded. The registers are loaded with the lowest-numbered register from the lowest memory address, through to the highest-numbered register from the highest memory address. The PC must be specified in the register list, and the instruction causes a branch to the address (data) loaded into the PC. See also Encoding of lists of general-purpose registers and the PC.

Instructions with similar syntax but without the PC included in the registers list are described in LDM (User registers).

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    if PSTATE.EL == EL2 then
        UNDEFINED;
    elsif PSTATE.M IN {M32_User,M32_System} then
        UNPREDICTABLE;                        // UNDEFINED or NOP
    else
        constant integer length = 4*BitCount(registers) + 4;
        bits(32) address = if increment then R[n] else R[n]-length;
        if wordhigher then address = address+4;
        bits(32) data;

        for i = 0 to 14
            if registers<i> == '1' then
                if i != n then
                    R[i] = MemS[address,4];
                else
                    data = MemS[address,4];
                address = address + 4;
        constant bits(32) new_pc_value = MemS[address,4];

        if wback && registers<n> == '1' then
            R[n] = bits(32) UNKNOWN;
        if wback && registers<n> == '0' then
            R[n] = if increment then R[n]+length else R[n]-length;
        if !wback && registers<n> == '1' then
            R[n] = data;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        AArch32.ExceptionReturn(new_pc_value, SPSR_curr[]);
```

CONSTRAINED UNPREDICTABLE behavior

If `PSTATE.M IN {M32_User,M32_System}`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.69 LDM (User registers)

Load Multiple (User registers)

In an EL1 mode other than System mode, Load Multiple (User registers) loads multiple User mode registers from consecutive memory locations using an address from a base register. The registers loaded cannot include the PC. The PE reads the base register value normally, using the current mode to determine the correct Banked version of the register. This instruction cannot writeback to the base register.

Load Multiple (User registers) is UNDEFINED in Hyp mode, and UNPREDICTABLE in User and System modes.

Armv8.2 permits the deprecation of some Load Multiple ordering behaviors in AArch32 state, for more information see FEAT_LSMAOC.

Encoding

```
LDM{<amode>}{<c>}{<q>}  <Rn>, <registers_without_pc>^
```

Decode for this encoding

```
constant integer n = UInt(Rn);
constant bits(15) registers = register_list;
constant boolean increment = (U == '1');
constant boolean wordhigher = (P == U);
if n == 15 || BitCount(registers) < 1 then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `BitCount(registers) < 1`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction operates as an `LDM` with the same addressing mode but targeting an unspecified set of registers. These registers might include R15.

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<amode>

is one of:

DA Decrement After. The consecutive memory addresses end at the address in the base register. Encoded as P = 0, U = 0.
FA Full Ascending. For this instruction, a synonym for `DA`.
DB Decrement Before. The consecutive memory addresses end one word below the address in the base register. Encoded as P = 1, U = 0.
EA Empty Ascending. For this instruction, a synonym for `DB`.
IA Increment After. The consecutive memory addresses start at the address in the base register. This is the default. Encoded as P = 0, U = 1.
FD Full Descending. For this instruction, a synonym for `IA`.
IB Increment Before. The consecutive memory addresses start one word above the address in the base register. Encoded as P = 1, U = 1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

ED Empty Descending. For this instruction, a synonym for `IB`.

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rn>

Is the general-purpose base register, encoded in the "Rn" field.

<registers_without_pc>

Is a list of one or more registers, separated by commas and surrounded by { and }. It specifies the set of registers to be loaded by the `LDM` instruction. The registers are loaded with the lowest-numbered register from the lowest memory address, through to the highest-numbered register from the highest memory address. The PC must not be in the register list. See also Encoding of lists of general-purpose registers and the PC.

Instructions with similar syntax but with the PC included in `<registers_without_pc>` are described in LDM (exception return).

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    if PSTATE.EL == EL2 then UNDEFINED;
    elsif PSTATE.M IN {M32_User,M32_System} then UNPREDICTABLE;
    else
        constant integer length = 4*BitCount(registers);
        bits(32) address = if increment then R[n] else R[n]-length;
        if wordhigher then address = address+4;
        for i = 0 to 14
            if registers<i> == '1' then  // Load User mode register
                Rmode[i, M32_User] = MemS[address,4];  address = address + 4;
```

CONSTRAINED UNPREDICTABLE behavior

If `PSTATE.M IN {M32_User,M32_System}`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction operates as an `LDM` with the same addressing mode but targeting an unspecified set of registers. These registers might include R15.

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.70 LDMDA, LDMFA

Load Multiple Decrement After (Full Ascending) loads multiple registers from consecutive memory locations using an address from a base register. The consecutive memory locations end at this address, and the address just below the lowest of those locations can optionally be written back to the base register.

The lowest-numbered register is loaded from the lowest memory address, through to the highest-numbered register from the highest memory address. See also Encoding of lists of general-purpose registers and the PC.

Armv8.2 permits the deprecation of some Load Multiple ordering behaviors in AArch32 state, for more information see FEAT_LSMAOC. The registers loaded can include the PC, causing a branch to a loaded address. This is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC. Related system instructions are LDM (User registers) and LDM (exception return).

Encoding

```
LDMDA{<c>}{<q>}  <Rn>{!}, <registers> // (Preferred syntax)
```

```
LDMFA{<c>}{<q>}  <Rn>{!}, <registers> // (Alternate syntax, Full Ascending stack)
```

Decode for this encoding

```
constant integer n = UInt(Rn);
constant bits(16) registers = register_list;
constant boolean wback = (W == '1');
if n == 15 || BitCount(registers) < 1 then UNPREDICTABLE;
if wback && registers<n> == '1' then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `BitCount(registers) < 1`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction operates as an `LDM` with the same addressing mode but targeting an unspecified set of registers. These registers might include R15. If the instruction specifies writeback, the modification to the base address on writeback might differ from the number of registers loaded.

If `wback && registers<n> == '1'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such as instruction, the base address might be corrupted so that the instruction cannot be repeated.

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rn>

Is the general-purpose base register, encoded in the “Rn” field.

!

The address adjusted by the size of the data loaded is written back to the base register. If specified, it is encoded in the “W” field as 1, otherwise this field defaults to 0.

<registers>

Is a list of one or more registers to be loaded, separated by commas and surrounded by { and }.

The PC can be in the list.

Arm deprecates using these instructions with both the LR and the PC in the list.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    bits(32) address = R[n] - 4*BitCount(registers) + 4;
    bits(32) data;
    for i = 0 to 14
        if registers<i> == '1' then
            if i != n then
                R[i] = MemS[address,4];
            else
                data = MemS[address,4];
            address = address + 4;
    if registers<15> == '1' then
        LoadWritePC(MemS[address,4]);

    if wback && registers<n> == '1' then
        R[n] = bits(32) UNKNOWN;
    if wback && registers<n> == '0' then
        R[n] = R[n] - 4*BitCount(registers);
    if !wback && registers<n> == '1' then
        R[n] = data;
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.71 LDMDB, LDMEA

Load Multiple Decrement Before (Empty Ascending) loads multiple registers from consecutive memory locations using an address from a base register. The consecutive memory locations end just below this address, and the address of the lowest of those locations can optionally be written back to the base register.

The lowest-numbered register is loaded from the lowest memory address, through to the highest-numbered register from the highest memory address. See also Encoding of lists of general-purpose registers and the PC.

Armv8.2 permits the deprecation of some Load Multiple ordering behaviors in AArch32 state, for more information see FEAT_LSMAOC. The registers loaded can include the PC, causing a branch to a loaded address. This is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC. Related system instructions are LDM (User registers) and LDM (exception return).

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding

```
LDMDB{<c>}{<q>}  <Rn>{!}, <registers> // (Preferred syntax)
```

```
LDMEA{<c>}{<q>}  <Rn>{!}, <registers> // (Alternate syntax, Empty Ascending stack)
```

Decode for this encoding

```
constant integer n = UInt(Rn);
constant bits(16) registers = register_list;
constant boolean wback = (W == '1');
if n == 15 || BitCount(registers) < 1 then UNPREDICTABLE;
if wback && registers<n> == '1' then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `wback && registers<n> == '1'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such an instruction, the base address might be corrupted so that the instruction cannot be repeated.

If `BitCount(registers) < 1`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction executes as LDM with the same addressing mode but targeting an unspecified set of registers. These registers might include R15. If the instruction specifies writeback, the modification to the base address on writeback might differ from the number of registers loaded.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T1

15						9	8	7	6	5	4	3		0	15	14	13														0
1	1	1	0	1	0	0	1	0	0	W	1		Rn		P	M								register_list							
							opc				L																				

Encoding

```
LDMDB{<c>}{<q>}  <Rn>{!}, <registers> // (Preferred syntax)
```

```
LDMEA{<c>}{<q>}  <Rn>{!}, <registers> // (Alternate syntax, Empty Ascending stack)
```

Decode for this encoding

```
constant integer n = UInt(Rn);
constant bits(16) registers = P:M:register_list;
constant boolean wback = (W == '1');
if n == 15 || BitCount(registers) < 2 || (P == '1' && M == '1') then UNPREDICTABLE;
if wback && registers<n> == '1' then UNPREDICTABLE;
if registers<13> == '1' then UNPREDICTABLE;
if registers<15> == '1' && InITBlock() && !LastInITBlock() then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `wback && registers<n> == '1'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such an instruction, the base address might be corrupted so that the instruction cannot be repeated.

If `BitCount(registers) < 1`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction executes as `LDM` with the same addressing mode but targeting an unspecified set of registers. These registers might include R15. If the instruction specifies writeback, the modification to the base address on writeback might differ from the number of registers loaded.

If `BitCount(registers) == 1`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction loads a single register using the specified addressing modes.
- The instruction executes as `LDM` with the same addressing mode but targeting an unspecified set of registers. These registers might include R15.

If `registers<13> == '1'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction performs all of the loads using the specified addressing mode, but R13 is UNKNOWN.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

If `P == '1' && M == '1'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction loads the register list and either R14 or R15, both R14 and R15, or neither of these registers.

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rn>

Is the general-purpose base register, encoded in the "Rn" field.

!

The address adjusted by the size of the data loaded is written back to the base register. If specified, it is encoded in the "W" field as 1, otherwise this field defaults to 0.

<registers>

For the "A1" variant: is a list of one or more registers to be loaded, separated by commas and surrounded by { and }.

The PC can be in the list.

Arm deprecates using these instructions with both the LR and the PC in the list.

For the "T1" variant: is a list of one or more registers to be loaded, separated by commas and surrounded by { and }. The registers in the list must be in the range R0-R12, encoded in the "register_list" field, and can optionally contain one of the LR or the PC. If the LR is in the list, the "M" field is set to 1, otherwise it defaults to 0. If the PC is in the list, the "P" field is set to 1, otherwise it defaults to 0.

If the PC is in the list:

- The LR must not be in the list.
- The instruction must be either outside any IT block, or the last instruction in an IT block.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    bits(32) address = R[n] - 4*BitCount(registers);
    bits(32) data;
    for i = 0 to 14
        if registers<i> == '1' then
            if i != n then
                R[i] = MemS[address,4];
            else
                data = MemS[address,4];
            address = address + 4;
    if registers<15> == '1' then
        LoadWritePC(MemS[address,4]);

    if wback && registers<n> == '1' then
        R[n] = bits(32) UNKNOWN;
    if wback && registers<n> == '0' then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        R[n] = R[n] - 4*BitCount(registers);
    if !wback && registers<n> == '1' then
        R[n] = data;
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.72 LDMIB, LDMED

Load Multiple Increment Before (Empty Descending) loads multiple registers from consecutive memory locations using an address from a base register. The consecutive memory locations start just above this address, and the address of the last of those locations can optionally be written back to the base register.

The lowest-numbered register is loaded from the lowest memory address, through to the highest-numbered register from the highest memory address. See also Encoding of lists of general-purpose registers and the PC.

Armv8.2 permits the deprecation of some Load Multiple ordering behaviors in AArch32 state, for more information see FEAT_LSMAOC. The registers loaded can include the PC, causing a branch to a loaded address. This is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC. Related system instructions are LDM (User registers) and LDM (exception return).

Encoding

```
LDMIB{<c>}{<q>}  <Rn>{!}, <registers> // (Preferred syntax)
```

```
LDMED{<c>}{<q>}  <Rn>{!}, <registers> // (Alternate syntax, Empty Descending stack)
```

Decode for this encoding

```
constant integer n = UInt(Rn);
constant bits(16) registers = register_list;
constant boolean wback = (W == '1');
if n == 15 || BitCount(registers) < 1 then UNPREDICTABLE;
if wback && registers<n> == '1' then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `BitCount(registers) < 1`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction operates as an LDM with the same addressing mode but targeting an unspecified set of registers. These registers might include R15. If the instruction specifies writeback, the modification to the base address on writeback might differ from the number of registers loaded.

If `wback && registers<n> == '1'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such as instruction, the base address might be corrupted so that the instruction cannot be repeated.

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rn>

Is the general-purpose base register, encoded in the “Rn” field.

!

The address adjusted by the size of the data loaded is written back to the base register. If specified, it is encoded in the “W” field as 1, otherwise this field defaults to 0.

<registers>

Is a list of one or more registers to be loaded, separated by commas and surrounded by { and }.

The PC can be in the list.

Arm deprecates using these instructions with both the LR and the PC in the list.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    bits(32) address = R[n] + 4;
    bits(32) data;
    for i = 0 to 14
        if registers<i> == '1' then
            if i != n then
                R[i] = MemS[address,4];
            else
                data = MemS[address,4];
            address = address + 4;
    if registers<15> == '1' then
        LoadWritePC(MemS[address,4]);

    if wback && registers<n> == '1' then
        R[n] = bits(32) UNKNOWN;
    if wback && registers<n> == '0' then
        R[n] = R[n] + 4*BitCount(registers);
    if !wback && registers<n> == '1' then
        R[n] = data;
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.73 LDR (immediate)

Load Register (immediate) calculates an address from a base register value and an immediate offset, loads a word from memory, and writes it to a register. It can use offset, post-indexed, or pre-indexed addressing. For information about memory accesses see Memory accesses.

This instruction is used by the alias POP (single register).

It has encodings from the following instruction sets: A32 (A1) and T32 (T1, T2, T3, and T4).

A1

Encoding for the Offset variant

Applies when `(P == 1 && W == 0)`

```
LDR{<c>}{<q>}  <Rt>, [<Rn> {, #{+/-}<imm>}]
```

Encoding for the Post-indexed variant

Applies when `(P == 0 && W == 0)`

```
LDR{<c>}{<q>}  <Rt>, [<Rn>], #{+/-}<imm>
```

Encoding for the Pre-indexed variant

Applies when `(P == 1 && W == 1)`

```
LDR{<c>}{<q>}  <Rt>, [<Rn>, #{+/-}<imm>]!
```

Decode for all variants of this encoding

```
if Rn == '1111' then SEE "LDR (literal)";
if P == '0' && W == '1' then SEE "LDRT";
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant bits(32) imm32 = ZeroExtend(imm12, 32);
constant boolean index = (P == '1');
constant boolean add = (U == '1');
constant boolean wback = (P == '0') || (W == '1');
if wback && n == t then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `wback && n == t`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such an instruction, the base address might be corrupted so that the instruction cannot be repeated.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T1

Encoding

```
LDR{<c>}{<q>}  <Rt>, [<Rn> {, #{+}<imm>}]
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant bits(32) imm32 = ZeroExtend(imm5:'00', 32);
constant boolean index = TRUE;
constant boolean add = TRUE;
constant boolean wback = FALSE;
```

T2

Encoding

```
LDR{<c>}{<q>}  <Rt>, [SP{, #{+}<imm>}]
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = 13;
constant bits(32) imm32 = ZeroExtend(imm8:'00', 32);
constant boolean index = TRUE;
constant boolean add = TRUE;
constant boolean wback = FALSE;
```

T3

Encoding

```
LDR{<c>}{<q>}  <Rt>, [<Rn> {, #{+}<imm>}]
```

```
LDR{<c>}.W  <Rt>, [<Rn> {, #{+}<imm>}] // (<Rt>, <Rn>, <imm> can be represented in T1 or T2)
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for this encoding

```
if Rn == '1111' then SEE "LDR (literal)";
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant bits(32) imm32 = ZeroExtend(imm12, 32);
constant boolean index = TRUE;
constant boolean add = TRUE;
constant boolean wback = FALSE;
if t == 15 && InITBlock() && !LastInITBlock() then UNPREDICTABLE;
```

T4

15 … 7	6	5	4	3 … 0	15 … 12	11	10	9	8	7 … 0
1 1 1 1 1 0 0 0 0	1	0	1	!=1111	Rt	1	P	U	W	imm8
	size		L	Rn						

Encoding for the Offset variant

Applies when `(P == 1 && U == 0 && W == 0)`

```
LDR{<c>}{<q>}  <Rt>, [<Rn> {, #-<imm>}]
```

Encoding for the Post-indexed variant

Applies when `(P == 0 && W == 1)`

```
LDR{<c>}{<q>}  <Rt>, [<Rn>], #{+/-}<imm>
```

Encoding for the Pre-indexed variant

Applies when `(P == 1 && W == 1)`

```
LDR{<c>}{<q>}  <Rt>, [<Rn>, #{+/-}<imm>]!
```

Decode for all variants of this encoding

```
if Rn == '1111' then SEE "LDR (literal)";
if P == '1' && U == '1' && W == '0' then SEE "LDRT";
if P == '0' && W == '0' then UNDEFINED;
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant bits(32) imm32 = ZeroExtend(imm8, 32);
constant boolean index = (P == '1');
constant boolean add = (U == '1');
constant boolean wback = (W == '1');
if (wback && n == t) || (t == 15 && InITBlock() && !LastInITBlock()) then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `wback && n == t`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such an instruction, the base address might be corrupted so that the instruction cannot be repeated.

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

For the “A1 Offset”, “A1 Post-indexed”, and “A1 Pre-indexed” variants: is the general-purpose register to be transferred, encoded in the “Rt” field. The PC can be used. If the PC is used, the instruction branches to the address (data) loaded to the PC. This is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.

For the “T1” and “T2” variants: is the general-purpose register to be transferred, encoded in the “Rt” field.

For the “T3”, “T4 Offset”, “T4 Post-indexed”, and “T4 Pre-indexed” variants: is the general-purpose register to be transferred, encoded in the “Rt” field. The PC can be used, provided the instruction is either outside an IT block or the last instruction of an IT block. If the PC is used, the instruction branches to the address (data) loaded to the PC. This is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.

<Rn>

For the “A1 Offset”, “A1 Post-indexed”, “A1 Pre-indexed”, “T3”, “T4 Offset”, “T4 Post-indexed”, and “T4 Pre-indexed” variants: is the general-purpose base register, encoded in the “Rn” field. For PC use see LDR (literal).

For the “T1” variant: is the general-purpose base register, encoded in the “Rn” field.

+/-

Specifies the offset is added to or subtracted from the base register, defaulting to + if omitted and encoded in “U”:

U	+/-
0	-
1	+

<imm>

For the “A1 Offset”, “A1 Post-indexed”, and “A1 Pre-indexed” variants: is the 12-bit unsigned immediate byte offset, in the range 0 to 4095, defaulting to 0 if omitted, and encoded in the “imm12” field.

For the “T1” variant: is the optional positive unsigned immediate byte offset, a multiple of 4, in the range 0 to 124, defaulting to 0 and encoded in the “imm5” field as <imm>/4.

For the “T2” variant: is the optional positive unsigned immediate byte offset, a multiple of 4, in the range 0 to 1020, defaulting to 0 and encoded in the “imm8” field as <imm>/4.

For the “T3” variant: is an optional 12-bit unsigned immediate byte offset, in the range 0 to 4095, defaulting to 0 and encoded in the “imm12” field.

For the “T4 Offset”, “T4 Post-indexed”, and “T4 Pre-indexed” variants: is an 8-bit unsigned immediate byte offset, in the range 0 to 255, defaulting to 0 if omitted, and encoded in the “imm8” field.

+

Specifies the offset is added to the base register.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Alias Conditions

Alias	Of variant	Is preferred when
POP (single register)	A1 Post-indexed	P == '0' && U == '1' && W == '0' && Rn == '1101' && imm12 == '000000000100'
POP (single register)	T4 Post-indexed	Rn == '1101' && P == '0' && U == '1' && W == '1' && imm8 == '00000100'

Operation

```
if CurrentInstrSet() == InstrSet_A32 then
    if ConditionPassed() then
        EncodingSpecificOperations();
        constant bits(32) offset_addr = if add then (R[n] + imm32) else (R[n] - imm32);
        constant bits(32) address = if index then offset_addr else R[n];
        constant bits(32) data = MemU[address,4];
        if wback then R[n] = offset_addr;
        if t == 15 then
            if address<1:0> == '00' then
                LoadWritePC(data);
            else
                UNPREDICTABLE;
        else
            R[t] = data;
else
    if ConditionPassed() then
        EncodingSpecificOperations();
        constant bits(32) offset_addr = if add then (R[n] + imm32) else (R[n] - imm32);
        constant bits(32) address = if index then offset_addr else R[n];
        constant bits(32) data = MemU[address,4];
        if wback then R[n] = offset_addr;
        if t == 15 then
            if address<1:0> == '00' then
                LoadWritePC(data);
            else
                UNPREDICTABLE;
        else
            R[t] = data;
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.74 LDR (literal)

Load Register (literal) calculates an address from the PC value and an immediate offset, loads a word from memory, and writes it to a register. For information about memory accesses see Memory accesses.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1 and T2).

A1

Encoding

Applies when `(!(P == '0' && W == '1'))`

```
LDR{<c>}{<q>}  <Rt>, <label> // (Normal form)
```

```
LDR{<c>}{<q>}  <Rt>, [PC, #{+/-}<imm>] // (Alternative form)
```

Decode for this encoding

```
if P == '0' && W == '1' then SEE "LDRT";
constant integer t = UInt(Rt);
constant bits(32) imm32 = ZeroExtend(imm12, 32);
constant boolean add = (U == '1');
constant boolean wback = (P == '0') || (W == '1');
if wback then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `wback`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction executes with the additional decode: wback = FALSE;.
- The instruction treats bit[24] as the P bit, and bit[21] as the writeback (W) bit, and uses the same addressing mode as described in LDR (immediate). The instruction uses post-indexed addressing when P == ‘0’ and uses pre-indexed addressing otherwise. The instruction is handled as described in Using R15.

T1

Encoding

```
LDR{<c>}{<q>}  <Rt>, <label> // (Normal form)
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for this encoding

```
constant integer t = UInt(Rt);
constant bits(32) imm32 = ZeroExtend(imm8:'00', 32);
constant boolean add = TRUE;
```

T2

15 … 8	7	6 5	4	3 … 0	15 … 12	11 … 0
1 1 1 1 1 0 0 0	U	1 0	1	1 1 1 1	Rt	imm12
		size	L			

Encoding

```
LDR{<c>}.W  <Rt>, <label> // (Preferred syntax)
```

```
LDR{<c>}{<q>}  <Rt>, <label> // (Preferred syntax, and <Rt>, <label> can be represented in T1)
```

```
LDR{<c>}{<q>}  <Rt>, [PC, #{+/-}<imm>] // (Alternative syntax)
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant bits(32) imm32 = ZeroExtend(imm12, 32);
constant boolean add = (U == '1');
if t == 15 && InITBlock() && !LastInITBlock() then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

For the “A1” variant: is the general-purpose register to be transferred, encoded in the “Rt” field. The PC can be used. If the PC is used, the instruction branches to the address (data) loaded to the PC. This is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.

For the “T1” variant: is the general-purpose register to be transferred, encoded in the “Rt” field.

For the “T2” variant: is the general-purpose register to be transferred, encoded in the “Rt” field. The SP can be used. The PC can be used, provided the instruction is either outside an IT block or the last instruction of an IT block. If the PC is used, the instruction branches to the address (data) loaded to the PC. This is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.

<label>

For the “A1” and “T2” variants: the label of the literal data item that is to be loaded into `<Rt>`. The assembler calculates the required value of the offset from the `Align(PC, 4)` value of the instruction to this label. Permitted values of the offset are -4095 to 4095.

If the offset is zero or positive, `imm32` is equal to the offset and `add` == `TRUE`, encoded as U == 1.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

If the offset is negative, `imm32` is equal to minus the offset and `add` == `FALSE`, encoded as U == 0.

For the "T1" variant: the label of the literal data item that is to be loaded into `<Rt>`. The assembler calculates the required value of the offset from the `Align(PC, 4)` value of the instruction to this label. Permitted values of the offset are Multiples of four in the range 0 to 1020.

+/-

Specifies the offset is added to or subtracted from the base register, defaulting to + if omitted and encoded in "U":

U	+/-
0	-
1	+

<imm>

For the "A1" variant: is the 12-bit unsigned immediate byte offset, in the range 0 to 4095, defaulting to 0 if omitted, and encoded in the "imm12" field.

For the "T2" variant: is a 12-bit unsigned immediate byte offset, in the range 0 to 4095, encoded in the "imm12" field.

The alternative syntax permits the addition or subtraction of the offset and the immediate offset to be specified separately, including permitting a subtraction of 0 that cannot be specified using the normal syntax. For more information, see Use of labels in UAL instruction syntax.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) base = Align(PC32,4);
    constant bits(32) address = if add then (base + imm32) else (base - imm32);
    constant bits(32) data = MemU[address,4];
    if t == 15 then
        if address<1:0> == '00' then
            LoadWritePC(data);
        else
            UNPREDICTABLE;
    else
        R[t] = data;
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.75 LDR (register)

Load Register (register) calculates an address from a base register value and an offset register value, loads a word from memory, and writes it to a register. The offset register value can optionally be shifted. For information about memory accesses, see Memory accesses.

The T32 form of `LDR` (register) does not support register writeback.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1 and T2).

A1

31–28	27–25	24	23	22	21	20	19–16	15–12	11–7	6–5	4	3–0
!=1111	0 1 1	P	U	0	W	1	Rn	Rt	imm5	stype	0	Rm
cond				op2		op1						

Encoding for the Offset variant

Applies when `(P == 1 && W == 0)`

```
LDR{<c>}{<q>}  <Rt>, [<Rn>, {+/-}<Rm>{, <shift>}]
```

Encoding for the Post-indexed variant

Applies when `(P == 0 && W == 0)`

```
LDR{<c>}{<q>}  <Rt>, [<Rn>], {+/-}<Rm>{, <shift>}
```

Encoding for the Pre-indexed variant

Applies when `(P == 1 && W == 1)`

```
LDR{<c>}{<q>}  <Rt>, [<Rn>, {+/-}<Rm>{, <shift>}]!
```

Decode for all variants of this encoding

```
if P == '0' && W == '1' then SEE "LDRT";
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean index = (P == '1');
constant boolean add = (U == '1');
constant boolean wback = (P == '0') || (W == '1');
SRType shift_t;
integer shift_n;
(shift_t, shift_n) = DecodeImmShift(stype, imm5);
if m == 15 then UNPREDICTABLE;
if wback && (n == 15 || n == t) then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `wback && n == t`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such as instruction, the base address might be corrupted so that the instruction cannot be repeated.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T1

Encoding

```
LDR{<c>}{<q>}  <Rt>, [<Rn>, {+}<Rm>]
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant SRType shift_t  = SRType_LSL;
constant integer shift_n = 0;
constant boolean index = TRUE;
constant boolean add = TRUE;
constant boolean wback = FALSE;
```

T2

Encoding

```
LDR{<c>}{<q>}  <Rt>, [<Rn>, {+}<Rm>{, LSL #<imm>}]
```

```
LDR{<c>}.W  <Rt>, [<Rn>, {+}<Rm>] // (<Rt>, <Rn>, <Rm> can be represented in T1)
```

Decode for this encoding

```
if Rn == '1111' then SEE "LDR (literal)";
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant SRType shift_t = SRType_LSL;
constant integer shift_n = UInt(imm2);
// Armv8-A removes UNPREDICTABLE for R13
if m == 15 then UNPREDICTABLE;
if t == 15 && InITBlock() && !LastInITBlock() then UNPREDICTABLE;
constant boolean index = TRUE;
constant boolean add = TRUE;
constant boolean wback = FALSE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

For the “Offset”, “Post-indexed”, and “Pre-indexed” variants: is the general-purpose register to be transferred, encoded in the “Rt” field. The PC can be used. If the PC is used, the instruction branches to the address (data) loaded to the PC. This branch is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.

For the “T1” variant: is the general-purpose register to be transferred, encoded in the “Rt” field.

For the “T2” variant: is the general-purpose register to be transferred, encoded in the “Rt” field. The PC can be used, provided the instruction is either outside an IT block or the last instruction of an IT block. If the PC is used, the instruction branches to the address (data) loaded to the PC. This is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.

<Rn>

For the “Offset”, “Post-indexed”, and “Pre-indexed” variants: is the general-purpose base register, encoded in the “Rn” field. The PC can be used in the offset variant.

For the “T1” and “T2” variants: is the general-purpose base register, encoded in the “Rn” field.

+/-

Specifies the index register is added to or subtracted from the base register, defaulting to + if omitted and encoded in “U”:

U	+/-
0	-
1	+

<Rm>

Is the general-purpose index register, encoded in the “Rm” field.

<shift>

The shift to apply to the value read from `<Rm>`. If absent, no shift is applied. Otherwise, see Shifts applied to a register.

+

Specifies the index register is added to the base register.

<imm>

If present, the size of the left shift to apply to the value from `<Rm>`, in the range 1-3. `<imm>` is encoded in imm2. If absent, no shift is specified and imm2 is encoded as `0b00`.

Operation

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
if CurrentInstrSet() == InstrSet_A32 then
    if ConditionPassed() then
        EncodingSpecificOperations();
        constant bits(32) offset = Shift(R[m], shift_t, shift_n, PSTATE.C);
        constant bits(32) offset_addr = if add then (R[n] + offset) else (R[n] - offset);
        constant bits(32) address = if index then offset_addr else R[n];
        constant bits(32) data = MemU[address,4];
        if wback then R[n] = offset_addr;
        if t == 15 then
            if address<1:0> == '00' then
                LoadWritePC(data);
            else
                UNPREDICTABLE;
        else
            R[t] = data;
else
    if ConditionPassed() then
        EncodingSpecificOperations();
        constant bits(32) offset = Shift(R[m], shift_t, shift_n, PSTATE.C);
        constant bits(32) offset_addr = (R[n] + offset);
        constant bits(32) address = offset_addr;
        constant bits(32) data = MemU[address,4];
        if t == 15 then
            if address<1:0> == '00' then
                LoadWritePC(data);
            else
                UNPREDICTABLE;
        else
            R[t] = data;
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.76 LDRB (immediate)

Load Register Byte (immediate) calculates an address from a base register value and an immediate offset, loads a byte from memory, zero-extends it to form a 32-bit word, and writes it to a register. It can use offset, post-indexed, or pre-indexed addressing. For information about memory accesses see Memory accesses.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1, T2, and T3).

A1

Encoding for the Offset variant

Applies when `(P == 1 && W == 0)`

```
LDRB{<c>}{<q>}  <Rt>, [<Rn> {, #{+/-}<imm>}]
```

Encoding for the Post-indexed variant

Applies when `(P == 0 && W == 0)`

```
LDRB{<c>}{<q>}  <Rt>, [<Rn>], #{+/-}<imm>
```

Encoding for the Pre-indexed variant

Applies when `(P == 1 && W == 1)`

```
LDRB{<c>}{<q>}  <Rt>, [<Rn>, #{+/-}<imm>]!
```

Decode for all variants of this encoding

```
if Rn == '1111' then SEE "LDRB (literal)";
if P == '0' && W == '1' then SEE "LDRBT";
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant bits(32) imm32 = ZeroExtend(imm12, 32);
constant boolean index = (P == '1');
constant boolean add = (U == '1');
constant boolean wback = (P == '0') || (W == '1');
if t == 15 || (wback && n == t) then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `wback && n == t`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such an instruction, the base address might be corrupted so that the instruction cannot be repeated.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T1

Encoding

```
LDRB{<c>}{<q>}  <Rt>, [<Rn> {, #{+}<imm>}]
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant bits(32) imm32 = ZeroExtend(imm5, 32);
constant boolean index = TRUE;
constant boolean add = TRUE;
constant boolean wback = FALSE;
```

T2

Encoding

```
LDRB{<c>}{<q>}  <Rt>, [<Rn> {, #{+}<imm>}]
```

```
LDRB{<c>}.W  <Rt>, [<Rn> {, #{+}<imm>}] // (<Rt>, <Rn>, <imm> can be represented in T1)
```

Decode for this encoding

```
if Rt == '1111' then SEE "PLD";
if Rn == '1111' then SEE "LDRB (literal)";
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant bits(32) imm32 = ZeroExtend(imm12, 32);
constant boolean index = TRUE;
constant boolean add = TRUE;
constant boolean wback = FALSE;
// Armv8-A removes UNPREDICTABLE for R13
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T3

Encoding for the Offset variant

Applies when `(Rt != 1111 && P == 1 && U == 0 && W == 0)`

```
LDRB{<c>}{<q>}  <Rt>, [<Rn> {, #-<imm>}]
```

Encoding for the Post-indexed variant

Applies when `(P == 0 && W == 1)`

```
LDRB{<c>}{<q>}  <Rt>, [<Rn>], #{+/-}<imm>
```

Encoding for the Pre-indexed variant

Applies when `(P == 1 && W == 1)`

```
LDRB{<c>}{<q>}  <Rt>, [<Rn>, #{+/-}<imm>]!
```

Decode for all variants of this encoding

```
if Rt == '1111' && P == '1' && U == '0' && W == '0' then SEE "PLD, PLDW (immediate)";
if Rn == '1111' then SEE "LDRB (literal)";
if P == '1' && U == '1' && W == '0' then SEE "LDRBT";
if P == '0' && W == '0' then UNDEFINED;
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant bits(32) imm32 = ZeroExtend(imm8, 32);
constant boolean index = (P == '1');
constant boolean add = (U == '1');
constant boolean wback = (W == '1');
if  (t == 15 &&  W == '1') || (wback && n == t) then UNPREDICTABLE;
// Armv8-A removes UNPREDICTABLE for R13
```

CONSTRAINED UNPREDICTABLE behavior

If `wback && n == t`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such an instruction, the base address might be corrupted so that the instruction cannot be repeated.

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

Is the general-purpose register to be transferred, encoded in the “Rt” field.

<Rn>

For the “A1 Offset”, “A1 Post-indexed”, “A1 Pre-indexed”, “T2”, “T3 Offset”, “T3 Post-indexed”, and “T3 Pre-indexed” variants: is the general-purpose base register, encoded in the “Rn” field. For PC use see LDRB (literal).

For the “T1” variant: is the general-purpose base register, encoded in the “Rn” field.

+/-

Specifies the offset is added to or subtracted from the base register, defaulting to + if omitted and encoded in “U”:

U	+/-
0	-
1	+

<imm>

For the “A1 Offset”, “A1 Post-indexed”, and “A1 Pre-indexed” variants: is the 12-bit unsigned immediate byte offset, in the range 0 to 4095, defaulting to 0 if omitted, and encoded in the “imm12” field.

For the “T1” variant: is an optional 5-bit unsigned immediate byte offset, in the range 0 to 31, defaulting to 0 and encoded in the “imm5” field.

For the “T2” variant: is an optional 12-bit unsigned immediate byte offset, in the range 0 to 4095, defaulting to 0 and encoded in the “imm12” field.

For the “T3 Offset”, “T3 Post-indexed”, and “T3 Pre-indexed” variants: is an 8-bit unsigned immediate byte offset, in the range 0 to 255, defaulting to 0 if omitted, and encoded in the “imm8” field.

+

Specifies the offset is added to the base register.

Operation

```
if CurrentInstrSet() == InstrSet_A32 then
    if ConditionPassed() then
        EncodingSpecificOperations();
        constant bits(32) offset_addr = if add then (R[n] + imm32) else (R[n] - imm32);
        constant bits(32) address = if index then offset_addr else R[n];
        R[t] = ZeroExtend(MemU[address,1], 32);
        if wback then R[n] = offset_addr;
else
    if ConditionPassed() then
        EncodingSpecificOperations();
        constant bits(32) offset_addr = if add then (R[n] + imm32) else (R[n] - imm32);
        constant bits(32) address = if index then offset_addr else R[n];
        R[t] = ZeroExtend(MemU[address,1], 32);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
        if wback then R[n] = offset_addr;
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.77 LDRB (literal)

Load Register Byte (literal) calculates an address from the PC value and an immediate offset, loads a byte from memory, zero-extends it to form a 32-bit word, and writes it to a register. For information about memory accesses see Memory accesses.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding

Applies when `(!(P == '0' && W == '1'))`

```
LDRB{<c>}{<q>}  <Rt>, <label> // (Normal form)
```

```
LDRB{<c>}{<q>}  <Rt>, [PC, #{+/-}<imm>] // (Alternative form)
```

Decode for this encoding

```
if P == '0' && W == '1' then SEE "LDRBT";
constant integer t = UInt(Rt);
constant bits(32) imm32 = ZeroExtend(imm12, 32);
constant boolean add = (U == '1');
constant boolean wback = (P == '0') || (W == '1');
if t == 15 || wback then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `wback`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction executes with the additional decode: wback = FALSE;.
- The instruction treats bit[24] as the P bit, and bit[21] as the writeback (W) bit, and uses the same addressing mode as described in LDRB (immediate). The instruction uses post-indexed addressing when P == ‘0’ and uses pre-indexed addressing otherwise. The instruction is handled as described in Using R15.

T1

Encoding

```
LDRB{<c>}{<q>}  <Rt>, <label> // (Preferred syntax)
```

```
LDRB{<c>}{<q>}  <Rt>, [PC, #{+/-}<imm>] // (Alternative syntax)
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for this encoding

```
if Rt == '1111' then SEE "PLD";
constant integer t = UInt(Rt);
constant bits(32) imm32 = ZeroExtend(imm12, 32);
constant boolean add = (U == '1');
// Armv8-A removes UNPREDICTABLE for R13
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

Is the general-purpose register to be transferred, encoded in the "Rt" field.

<label>

The label of the literal data item that is to be loaded into `<Rt>`. The assembler calculates the required value of the offset from the `Align(PC, 4)` value of the instruction to this label. Permitted values of the offset are -4095 to 4095.

If the offset is zero or positive, `imm32` is equal to the offset and `add` == `TRUE`, encoded as U == 1.

If the offset is negative, `imm32` is equal to minus the offset and `add` == `FALSE`, encoded as U == 0.

+/-

Specifies the offset is added to or subtracted from the base register, defaulting to + if omitted and encoded in "U":

U	+/-
0	-
1	+

<imm>

For the "A1" variant: is the 12-bit unsigned immediate byte offset, in the range 0 to 4095, defaulting to 0 if omitted, and encoded in the "imm12" field.

For the "T1" variant: is a 12-bit unsigned immediate byte offset, in the range 0 to 4095, encoded in the "imm12" field.

The alternative syntax permits the addition or subtraction of the offset and the immediate offset to be specified separately, including permitting a subtraction of 0 that cannot be specified using the normal syntax. For more information, see Use of labels in UAL instruction syntax.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) base = Align(PC32,4);
    constant bits(32) address = if add then (base + imm32) else (base - imm32);
    R[t] = ZeroExtend(MemU[address,1], 32);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.78 LDRB (register)

Load Register Byte (register) calculates an address from a base register value and an offset register value, loads a byte from memory, zero-extends it to form a 32-bit word, and writes it to a register. The offset register value can optionally be shifted. For information about memory accesses see Memory accesses.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1 and T2).

A1

31–28	27–25	24	23	22	21	20	19–16	15–12	11–7	6–5	4	3–0
!=1111	0 1 1	P	U	1	W	1	Rn	Rt	imm5	stype	0	Rm
cond			op2			op1						

Encoding for the Offset variant

Applies when `(P == 1 && W == 0)`

```
LDRB{<c>}{<q>}  <Rt>, [<Rn>, {+/-}<Rm>{, <shift>}]
```

Encoding for the Post-indexed variant

Applies when `(P == 0 && W == 0)`

```
LDRB{<c>}{<q>}  <Rt>, [<Rn>], {+/-}<Rm>{, <shift>}
```

Encoding for the Pre-indexed variant

Applies when `(P == 1 && W == 1)`

```
LDRB{<c>}{<q>}  <Rt>, [<Rn>, {+/-}<Rm>{, <shift>}]!
```

Decode for all variants of this encoding

```
if P == '0' && W == '1' then SEE "LDRBT";
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean index = (P == '1');
constant boolean add = (U == '1');
constant boolean wback = (P == '0') || (W == '1');
SRType shift_t;
integer shift_n;
(shift_t, shift_n) = DecodeImmShift(stype, imm5);
if t == 15 || m == 15 then UNPREDICTABLE;
if wback && (n == 15 || n == t) then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `wback && n == t`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such as instruction, the base address might be corrupted so that the instruction cannot be repeated.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T1

Encoding

```
LDRB{<c>}{<q>}  <Rt>, [<Rn>, {+}<Rm>]
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean index = TRUE;
constant boolean add = TRUE;
constant boolean wback = FALSE;
constant SRType shift_t = SRType_LSL;
constant integer shift_n = 0;
```

T2

Encoding

```
LDRB{<c>}{<q>}  <Rt>, [<Rn>, {+}<Rm>{, LSL #<imm>}]
```

```
LDRB{<c>}.W  <Rt>, [<Rn>, {+}<Rm>] // (<Rt>, <Rn>, <Rm> can be represented in T1)
```

Decode for this encoding

```
if Rt == '1111' then SEE "PLD";
if Rn == '1111' then SEE "LDRB (literal)";
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean index = TRUE;
constant boolean add = TRUE;
constant boolean wback = FALSE;
constant SRType shift_t = SRType_LSL;
constant integer shift_n = UInt(imm2);
// Armv8-A removes UNPREDICTABLE for R13
if m == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

Is the general-purpose register to be transferred, encoded in the “Rt” field.

<Rn>

For the “Offset”, “Post-indexed”, and “Pre-indexed” variants: is the general-purpose base register, encoded in the “Rn” field. The PC can be used in the offset variant.

For the “T1” and “T2” variants: is the general-purpose base register, encoded in the “Rn” field.

+/-

Specifies the index register is added to or subtracted from the base register, defaulting to + if omitted and encoded in “U”:

U	+/-
0	-
1	+

<Rm>

Is the general-purpose index register, encoded in the “Rm” field.

<shift>

The shift to apply to the value read from `<Rm>`. If absent, no shift is applied. Otherwise, see Shifts applied to a register.

+

Specifies the index register is added to the base register.

<imm>

If present, the size of the left shift to apply to the value from `<Rm>`, in the range 1-3. `<imm>` is encoded in imm2. If absent, no shift is specified and imm2 is encoded as `0b00`.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) offset = Shift(R[m], shift_t, shift_n, PSTATE.C);
    constant bits(32) offset_addr = if add then (R[n] + offset) else (R[n] - offset);
    constant bits(32) address = if index then offset_addr else R[n];
    R[t] = ZeroExtend(MemU[address,1],32);
    if wback then R[n] = offset_addr;
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.79 LDRBT

Load Register Byte Unprivileged loads a byte from memory, zero-extends it to form a 32-bit word, and writes it to a register. For information about memory accesses see Memory accesses.

The memory access is restricted as if the PE were running in User mode. This makes no difference if the PE is actually running in User mode.

`LDRBT` is UNPREDICTABLE in Hyp mode.

The T32 instruction uses an offset addressing mode, that calculates the address used for the memory access from a base register value and an immediate offset, and leaves the base register unchanged.

The A32 instruction uses a post-indexed addressing mode, that uses a base register value as the address for the memory access, and calculates a new address from a base register value and an offset and writes it back to the base register. The offset can be an immediate value or an optionally-shifted register value.

It has encodings from the following instruction sets: A32 (A1 and A2) and T32 (T1).

A1

Encoding

```
LDRBT{<c>}{<q>}  <Rt>, [<Rn>] {, #{+/-}<imm>}
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant boolean postindex = TRUE;
constant boolean add = (U == '1');
constant boolean register_form = FALSE;
constant bits(32) imm32 = ZeroExtend(imm12, 32);
constant integer m = integer UNKNOWN;
constant integer shift_n = integer UNKNOWN;
constant SRType shift_t = SRType UNKNOWN;
if t == 15 || n == 15 || n == t then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `n == 15`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction uses post-indexed addressing with the base register as PC. This is handled as described in Using R15.
- The instruction uses immediate offset addressing with the base register as PC, without writeback.

If `n == t && n != 15`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such as instruction, the base address might be corrupted so that the instruction cannot be repeated.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

A2

Encoding

```
LDRBT{<c>}{<q>}  <Rt>, [<Rn>], {+/-}<Rm>{, <shift>}
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean postindex = TRUE;
constant boolean add = (U == '1');
constant boolean register_form = TRUE;
SRType shift_t;
integer shift_n;
(shift_t, shift_n) = DecodeImmShift(stype, imm5);
constant bits(32) imm32 = bits(32) UNKNOWN;
if t == 15 || n == 15 || n == t || m == 15 then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `n == t && n != 15`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such as instruction, the base address might be corrupted so that the instruction cannot be repeated.

T1

Encoding

```
LDRBT{<c>}{<q>}  <Rt>, [<Rn> {, #{+}<imm>}]
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for this encoding

```
if Rn == '1111' then SEE "LDRB (literal)";
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant boolean postindex = FALSE;
constant boolean add = TRUE;
constant boolean register_form = FALSE;
constant bits(32) imm32 = ZeroExtend(imm8, 32);
constant integer m = integer UNKNOWN;
constant integer shift_n = integer UNKNOWN;
constant SRType shift_t = SRType UNKNOWN;
// Armv8-A removes UNPREDICTABLE for R13
if t == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

For the “A1” variant: is the general-purpose register to be transferred, encoded in the “Rt” field. The PC can be used, but this is deprecated.

For the “A2” and “T1” variants: is the general-purpose register to be transferred, encoded in the “Rt” field.

<Rn>

Is the general-purpose base register, encoded in the “Rn” field.

+/-

For the “A1” variant: specifies the offset is added to or subtracted from the base register, defaulting to + if omitted and encoded in “U”:

U	+/-
0	-
1	+

For the “A2” variant: specifies the index register is added to or subtracted from the base register, defaulting to + if omitted and encoded in “U”:

U	+/-
0	-
1	+

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

<imm>

For the “A1” variant: is the 12-bit unsigned immediate byte offset, in the range 0 to 4095, defaulting to 0 if omitted, and encoded in the “imm12” field.

For the “T1” variant: is an optional 8-bit unsigned immediate byte offset, in the range 0 to 255, defaulting to 0 and encoded in the “imm8” field.

<Rm>

Is the general-purpose index register, encoded in the “Rm” field.

<shift>

The shift to apply to the value read from `<Rm>`. If absent, no shift is applied. Otherwise, see Shifts applied to a register.

+

Specifies the offset is added to the base register.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    if PSTATE.EL == EL2 then UNPREDICTABLE;                    // Hyp mode
    constant bits(32) offset = (if register_form then Shift(R[m], shift_t, shift_n, PSTATE.C)
                                else imm32);
    constant bits(32) offset_addr = if add then (R[n] + offset) else (R[n] - offset);
    constant bits(32) address = if postindex then R[n] else offset_addr;
    R[t] = ZeroExtend(MemU_unpriv[address,1],32);
    if postindex then R[n] = offset_addr;
```

CONSTRAINED UNPREDICTABLE behavior

If `PSTATE.EL == EL2`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction executes as `LDRB` (immediate).

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.80 LDRD (immediate)

Load Register Dual (immediate) calculates an address from a base register value and an immediate offset, loads two words from memory, and writes them to two registers. It can use offset, post-indexed, or pre-indexed addressing. For information about memory accesses see Memory accesses.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding for the Offset variant

Applies when `(P == 1 && W == 0)`

```
LDRD{<c>}{<q>}  <Rt>, <Rt2>, [<Rn> {, #{+/-}<imm>}]
```

Encoding for the Post-indexed variant

Applies when `(P == 0 && W == 0)`

```
LDRD{<c>}{<q>}  <Rt>, <Rt2>, [<Rn>], #{+/-}<imm>
```

Encoding for the Pre-indexed variant

Applies when `(P == 1 && W == 1)`

```
LDRD{<c>}{<q>}  <Rt>, <Rt2>, [<Rn>, #{+/-}<imm>]!
```

Decode for all variants of this encoding

```
if Rn == '1111' then SEE "LDRD (literal)";
if Rt<0> == '1' then UNPREDICTABLE;
constant integer t = UInt(Rt);
constant integer t2 = t + 1;
constant integer n = UInt(Rn);
constant bits(32) imm32 = ZeroExtend(imm4H:imm4L, 32);
constant boolean index = (P == '1');
constant boolean add = (U == '1');
constant boolean wback = (P == '0') || (W == '1');
if P == '0' && W == '1' then UNPREDICTABLE;
if wback && (n == t || n == t2) then UNPREDICTABLE;
if t2 == 15 then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `wback && (n == t || n == t2)`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such an instruction, the base address might be corrupted so that the instruction cannot be repeated.

If `P == '0' && W == '1'`, then one of the following behaviors must occur:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction executes as an `LDRD` using one of offset, post-indexed, or pre-indexed addressing.

If `Rt<0> == '1'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction executes with the additional decode: t<0> = ‘0’.
- The instruction executes with the additional decode: t2 = t.
- The instruction executes as described, with no change to its behavior and no additional side-effects. This does not apply when Rt == ‘1111’.

T1

15–9	8	7	6	5	4	3–0	15–12	11–8	7–0
1 1 1 0 1 0 0	P	U	1	W	1	!=1111	Rt	Rt2	imm8
					L	Rn			

Encoding for the Offset variant

Applies when `(P == 1 && W == 0)`

```
LDRD{<c>}{<q>}  <Rt>, <Rt2>, [<Rn> {, #{+/-}<imm>}]
```

Encoding for the Post-indexed variant

Applies when `(P == 0 && W == 1)`

```
LDRD{<c>}{<q>}  <Rt>, <Rt2>, [<Rn>], #{+/-}<imm>
```

Encoding for the Pre-indexed variant

Applies when `(P == 1 && W == 1)`

```
LDRD{<c>}{<q>}  <Rt>, <Rt2>, [<Rn>, #{+/-}<imm>]!
```

Decode for all variants of this encoding

```
if P == '0' && W == '0' then SEE "Related encodings";
if Rn == '1111' then SEE "LDRD (literal)";
constant integer t = UInt(Rt);
constant integer t2 = UInt(Rt2);
constant integer n = UInt(Rn);
constant bits(32) imm32 = ZeroExtend(imm8:'00', 32);
constant boolean index = (P == '1');
constant boolean add = (U == '1');
constant boolean wback = (W == '1');
if wback && (n == t || n == t2) then UNPREDICTABLE;
// Armv8-A removes UNPREDICTABLE for R13
if t == 15 || t2 == 15 || t == t2 then UNPREDICTABLE;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CONSTRAINED UNPREDICTABLE behavior

If `wback && (n == t || n == t2)`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such an instruction, the base address might be corrupted so that the instruction cannot be repeated.

If `t == t2`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The load instruction executes but the destination register takes an UNKNOWN value.

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Related encodings: Load/store dual, load/store exclusive, table branch.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

For the “A1 Offset”, “A1 Post-indexed”, and “A1 Pre-indexed” variants: is the first general-purpose register to be transferred, encoded in the “Rt” field. This register must be even-numbered and not R14.

For the “T1 Offset”, “T1 Post-indexed”, and “T1 Pre-indexed” variants: is the first general-purpose register to be transferred, encoded in the “Rt” field.

<Rt2>

For the “A1 Offset”, “A1 Post-indexed”, and “A1 Pre-indexed” variants: is the second general-purpose register to be transferred. This register must be `<R(t+1)>`.

For the “T1 Offset”, “T1 Post-indexed”, and “T1 Pre-indexed” variants: is the second general-purpose register to be transferred, encoded in the “Rt2” field.

<Rn>

Is the general-purpose base register, encoded in the “Rn” field. For PC use see LDRD (literal).

+/-

Specifies the offset is added to or subtracted from the base register, defaulting to + if omitted and encoded in “U”:

U	+/-
0	-
1	+

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

<imm>

For the “A1 Offset”, “A1 Post-indexed”, and “A1 Pre-indexed” variants: is the 8-bit unsigned immediate byte offset, in the range 0 to 255, defaulting to 0 if omitted, and encoded in the “imm4H:imm4L” field.

For the “T1 Offset”, “T1 Post-indexed”, and “T1 Pre-indexed” variants: is the unsigned immediate byte offset, a multiple of 4, in the range 0 to 1020, defaulting to 0 if omitted, and encoded in the “imm8” field as <imm>/4.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) offset_addr = if add then (R[n] + imm32) else (R[n] - imm32);
    constant bits(32) address = if index then offset_addr else R[n];
    if IsAligned(address, 8) then
        constant bits(64) data = MemA[address,8];
        if BigEndian(AccessType_GPR)  then
            R[t]  = data<63:32>;
            R[t2] = data<31:0>;
        else
            R[t]  = data<31:0>;
            R[t2] = data<63:32>;
    else
        constant bits(32) data1 = MemA[address,4];
        constant bits(32) data2 = MemA[address+4,4];
        R[t]  = data1;
        R[t2] = data2;
    if wback then R[n] = offset_addr;
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.81 LDRD (literal)

Load Register Dual (literal) calculates an address from the PC value and an immediate offset, loads two words from memory, and writes them to two registers. For information about memory accesses see Memory accesses.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding

```
LDRD{<c>}{<q>}  <Rt>, <Rt2>, <label> // (Normal form)
```

```
LDRD{<c>}{<q>}  <Rt>, <Rt2>, [PC, #{+/-}<imm>] // (Alternative form)
```

Decode for this encoding

```
if Rt<0> == '1' then UNPREDICTABLE;
constant integer t = UInt(Rt);
constant integer t2 = t + 1;
constant bits(32) imm32 = ZeroExtend(imm4H:imm4L, 32);
constant boolean add = (U == '1');
if t2 == 15 then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `Rt<0> == '1'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction executes with the additional decode: t<0> = ‘0’;.
- The instruction executes with the additional decode: t2 = t;.
- The instruction executes as described, with no change to its behavior and no additional side-effects. This does not apply when Rt == ‘1111’.

If `P == '0' || W == '1'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction executes as if P == 1 and W == 0.’

T1

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Encoding

Applies when `(!(P == '0' && W == '0'))`

```
LDRD{<c>}{<q>}  <Rt>, <Rt2>, <label> // (Normal form)
```

```
LDRD{<c>}{<q>}  <Rt>, <Rt2>, [PC, #{+/-}<imm>] // (Alternative form)
```

Decode for this encoding

```
if P == '0' && W == '0' then SEE "Related encodings";
constant integer t = UInt(Rt);
constant integer t2 = UInt(Rt2);
constant bits(32) imm32 = ZeroExtend(imm8:'00', 32);
constant boolean add = (U == '1');
// Armv8-A removes UNPREDICTABLE for R13
if t == 15 || t2 == 15 || t == t2 then UNPREDICTABLE;
if W == '1' then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `t == t2`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The load instruction executes but the destination register takes an UNKNOWN value.

If `W == '1'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction executes without writeback of the base address.
- The instruction uses post-indexed addressing when P == ‘0’ and uses pre-indexed addressing otherwise. The instruction is handled as described in Using R15.

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Related encodings: Load/Store dual, Load/Store-Exclusive, Load-Acquire/Store-Release, table branch.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

For the “A1” variant: is the first general-purpose register to be transferred, encoded in the “Rt” field. This register must be even-numbered and not R14.

For the “T1” variant: is the first general-purpose register to be transferred, encoded in the “Rt” field.

<Rt2>

For the “A1” variant: is the second general-purpose register to be transferred. This register must be `<R(t+1)>`.

For the “T1” variant: is the second general-purpose register to be transferred, encoded in the “Rt2” field.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

<label>

For the "A1" variant: the label of the literal data item that is to be loaded into `<Rt>`. The assembler calculates the required value of the offset from the `Align(PC, 4)` value of the instruction to this label. Any value in the range -255 to 255 is permitted. If the offset is zero or positive, `imm32` is equal to the offset and `add` == `TRUE`, encoded as U == 1. If the offset is negative, `imm32` is equal to minus the offset and `add` == `FALSE`, encoded as U == 0.

For the "T1" variant: the label of the literal data item that is to be loaded into `<Rt>`. The assembler calculates the required value of the offset from the `Align(PC, 4)` value of the instruction to this label. Permitted values of the offset are multiples of 4 in the range -1020 to 1020.

If the offset is zero or positive, `imm32` is equal to the offset and `add` == `TRUE`, encoded as U == 1.

If the offset is negative, `imm32` is equal to minus the offset and `add` == `FALSE`, encoded as U == 0.

+/-

Specifies the offset is added to or subtracted from the base register, defaulting to + if omitted and encoded in "U":

U	+/-
0	-
1	+

<imm>

For the "A1" variant: is the 8-bit unsigned immediate byte offset, in the range 0 to 255, defaulting to 0 if omitted, and encoded in the "imm4H:imm4L" field.

For the "T1" variant: is the optional 8-bit unsigned immediate byte offset, in the range 0 to 255, defaulting to 0 and encoded in the "imm8" field.

The alternative syntax permits the addition or subtraction of the offset and the immediate offset to be specified separately, including permitting a subtraction of 0 that cannot be specified using the normal syntax. For more information, see Use of labels in UAL instruction syntax.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) address = (if add then (Align(PC32,4) + imm32)
                                 else (Align(PC32,4) - imm32));
    if IsAligned(address, 8) then
        constant bits(64) data = MemA[address,8];
        if BigEndian(AccessType_GPR) then
            R[t]  = data<63:32>;
            R[t2] = data<31:0>;
        else
            R[t]  = data<31:0>;
            R[t2] = data<63:32>;
    else
        constant bits(32) data1 = MemA[address,4];
        constant bits(32) data2 = MemA[address+4,4];
        R[t]  = data1;
        R[t2] = data2;
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.82 LDRD (register)

Load Register Dual (register) calculates an address from a base register value and a register offset, loads two words from memory, and writes them to two registers. It can use offset, post-indexed, or pre-indexed addressing. For information about memory accesses see Memory accesses.

31–28	27–25	24	23	22	21	20	19–16	15–12	11–7	6–5	4	3–0
!=1111	0 0 0	P	U	0	W	0	Rn	Rt	(0)(0)(0)(0)1	1 0	1	Rm
cond						o1				op2		

Encoding for the Offset variant

Applies when `(P == 1 && W == 0)`

```
LDRD{<c>}{<q>}  <Rt>, <Rt2>, [<Rn>, {+/-}<Rm>]
```

Encoding for the Post-indexed variant

Applies when `(P == 0 && W == 0)`

```
LDRD{<c>}{<q>}  <Rt>, <Rt2>, [<Rn>], {+/-}<Rm>
```

Encoding for the Pre-indexed variant

Applies when `(P == 1 && W == 1)`

```
LDRD{<c>}{<q>}  <Rt>, <Rt2>, [<Rn>, {+/-}<Rm>]!
```

Decode for all variants of this encoding

```
if Rt<0> == '1' then UNPREDICTABLE;
constant integer t = UInt(Rt);
constant integer t2 = t + 1;
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean index = (P == '1');
constant boolean add = (U == '1');
constant boolean wback = (P == '0') || (W == '1');
if P == '0' && W == '1' then UNPREDICTABLE;
if t2 == 15 || m == 15 || m == t || m == t2 then UNPREDICTABLE;
if wback && (n == 15 || n == t || n == t2) then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `wback && (n == t || n == t2)`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such as instruction, the base address might be corrupted so that the instruction cannot be repeated.

If `P == '0' && W == '1'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction executes as an LDRD using one of offset, post-indexed, or pre-indexed addressing.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

If `m == t || m == t2`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction loads register Rm with an UNKNOWN value.

If `Rt<0> == '1'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction executes with the additional decode: t<0> = ‘0’.
- The instruction executes with the additional decode: t2 = t.
- The instruction executes as described, with no change to its behavior and no additional side-effects. This does not apply when Rt == ‘1111’.

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

Is the first general-purpose register to be transferred, encoded in the “Rt” field. This register must be even-numbered and not R14.

<Rt2>

Is the second general-purpose register to be transferred. This register must be `<R(t+1)>`.

<Rn>

Is the general-purpose base register, encoded in the “Rn” field. The PC can be used in the offset variant.

+/-

Specifies the index register is added to or subtracted from the base register, defaulting to + if omitted and encoded in “U”:

U	+/-
0	-
1	+

<Rm>

Is the general-purpose index register, encoded in the “Rm” field.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) offset_addr = if add then (R[n] + R[m]) else (R[n] - R[m]);
    constant bits(32) address = if index then offset_addr else R[n];
    if IsAligned(address, 8) then
        constant bits(64) data = MemA[address,8];
        if BigEndian(AccessType_GPR) then
            R[t]  = data<63:32>;
            R[t2] = data<31:0>;
        else
            R[t]  = data<31:0>;
            R[t2] = data<63:32>;
    else
        constant bits(32) data1 = MemA[address,4];
        constant bits(32) data2 = MemA[address+4,4];
        R[t]  = data1;
        R[t2] = data2;

    if wback then R[n] = offset_addr;
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.83 LDREX

Load Register Exclusive calculates an address from a base register value and an immediate offset, loads a word from memory, writes it to a register and:

- If the address has the Shared Memory attribute, marks the physical address as exclusive access for the executing PE in a global monitor.
- Causes the executing PE to indicate an active exclusive access in the local monitor.

For more information about support for shared memory see Synchronization and semaphores. For information about memory accesses see Memory accesses.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding

```
LDREX{<c>}{<q>}  <Rt>, [<Rn> {, {#}<imm>}]
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant bits(32) imm32 = Zeros(32); // Zero offset
if t == 15 || n == 15 then UNPREDICTABLE;
```

T1

Encoding

```
LDREX{<c>}{<q>}  <Rt>, [<Rn> {, #<imm>}]
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant bits(32) imm32 = ZeroExtend(imm8:'00', 32);
// Armv8-A removes UNPREDICTABLE for R13
if t == 15 || n == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

Is the general-purpose register to be transferred, encoded in the “Rt” field.

<Rn>

Is the general-purpose base register, encoded in the “Rn” field.

<imm>

For the “A1” variant: the immediate offset added to the value of `<Rn>` to calculate the address. `<imm>` can only be 0 or omitted.

For the “T1” variant: the immediate offset added to the value of `<Rn>` to calculate the address. `<imm>` can be omitted, meaning an offset of 0. Values are multiples of 4 in the range 0-1020.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) address = R[n] + imm32;
    AArch32.SetExclusiveMonitors(address,4);
    R[t] = MemA[address,4];
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.84 LDREXB

Load Register Exclusive Byte derives an address from a base register value, loads a byte from memory, zero-extends it to form a 32-bit word, writes it to a register and:

- If the address has the Shared Memory attribute, marks the physical address as exclusive access for the executing PE in a global monitor.
- Causes the executing PE to indicate an active exclusive access in the local monitor.

For more information about support for shared memory see Synchronization and semaphores. For information about memory accesses see Memory accesses.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding

```
LDREXB{<c>}{<q>}  <Rt>, [<Rn>]
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
if t == 15 || n == 15 then UNPREDICTABLE;
```

T1

Encoding

```
LDREXB{<c>}{<q>}  <Rt>, [<Rn>]
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
// Armv8-A removes UNPREDICTABLE for R13
if t == 15 || n == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

Is the general-purpose register to be transferred, encoded in the “Rt” field.

<Rn>

Is the general-purpose base register, encoded in the “Rn” field.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) address = R[n];
    AArch32.SetExclusiveMonitors(address,1);
    R[t] = ZeroExtend(MemA[address,1], 32);
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.85 LDREXD

Load Register Exclusive Doubleword derives an address from a base register value, loads a 64-bit doubleword from memory, writes it to two registers and:

- If the address has the Shared Memory attribute, marks the physical address as exclusive access for the executing PE in a global monitor.
- Causes the executing PE to indicate an active exclusive access in the local monitor.

For more information about support for shared memory see Synchronization and semaphores. For information about memory accesses see Memory accesses.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

31–28	27	26	25	24	23	22	21	20	19–16	15–12	11	10	9	8	7	6	5	4	3–0
!=1111	0	0	0	1	1	0	1	1	Rn	Rt	(1)	(1)	1	1	1	0	0	1	(1)(1)(1)(1)
cond					size			L					ex	ord					xRt

Encoding

```
LDREXD{<c>}{<q>}  <Rt>, <Rt2>, [<Rn>]
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer t2 = t + 1;
constant integer n = UInt(Rn);
if Rt<0> == '1' || t2 == 15 || n == 15 then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `Rt<0> == '1'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction executes with the additional decode: t<0> = ‘0’.
- The instruction executes with the additional decode: t2 = t.
- The instruction executes as described, with no change to its behavior and no additional side effects.

If `Rt == '1110'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction is handled as described in Using R15.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T1

15–13	12–9	8–5	4	3–0	15–12	11–8	7–6	5–4	3–0
1 1 1	0 1 0 0	0 1 1 0	1	Rn	Rt	Rt2	0 1	1 1	(1) (1) (1) (1)
		op0	L					sz	Rd

Encoding

```
LDREXD{<c>}{<q>}  <Rt>, <Rt2>, [<Rn>]
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer t2 = UInt(Rt2);
constant integer n = UInt(Rn);
if t == 15 || t2 == 15 || t == t2 || n == 15 then UNPREDICTABLE;
// Armv8-A removes UNPREDICTABLE for R13
```

CONSTRAINED UNPREDICTABLE behavior

If `t == t2`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The load instruction executes but the destination register takes an UNKNOWN value.

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

For the “A1” variant: is the first general-purpose register to be transferred, encoded in the “Rt” field. `<Rt>` must be even-numbered and not R14.

For the “T1” variant: is the first general-purpose register to be transferred, encoded in the “Rt” field.

<Rt2>

For the “A1” variant: is the second general-purpose register to be transferred. `<Rt2>` must be `<R(t+1)>`.

For the “T1” variant: is the second general-purpose register to be transferred, encoded in the “Rt2” field.

<Rn>

Is the general-purpose base register, encoded in the “Rn” field.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) address = R[n];
    AArch32.SetExclusiveMonitors(address,8);
    constant bits(64) value = MemA[address,8];

    // Extract words from 64-bit loaded value such that R[t] is
    // loaded from address and R[t2] from address+4.
    R[t]  = if BigEndian(AccessType_GPR) then value<63:32> else value<31:0>;
    R[t2] = if BigEndian(AccessType_GPR) then value<31:0> else value<63:32>;
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.86 LDREXH

Load Register Exclusive Halfword derives an address from a base register value, loads a halfword from memory, zero-extends it to form a 32-bit word, writes it to a register and:

- If the address has the Shared Memory attribute, marks the physical address as exclusive access for the executing PE in a global monitor.
- Causes the executing PE to indicate an active exclusive access in the local monitor.

For more information about support for shared memory see Synchronization and semaphores. For information about memory accesses see Memory accesses.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding

```
LDREXH{<c>}{<q>}  <Rt>, [<Rn>]
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
if t == 15 || n == 15 then UNPREDICTABLE;
```

T1

Encoding

```
LDREXH{<c>}{<q>}  <Rt>, [<Rn>]
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
// Armv8-A removes UNPREDICTABLE for R13
if t == 15 || n == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

Is the general-purpose register to be transferred, encoded in the “Rt” field.

<Rn>

Is the general-purpose base register, encoded in the “Rn” field.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) address = R[n];
    AArch32.SetExclusiveMonitors(address,2);
    R[t] = ZeroExtend(MemA[address,2], 32);
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.87 LDRH (immediate)

Load Register Halfword (immediate) calculates an address from a base register value and an immediate offset, loads a halfword from memory, zero-extends it to form a 32-bit word, and writes it to a register. It can use offset, post-indexed, or pre-indexed addressing. For information about memory accesses see Memory accesses.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1, T2, and T3).

A1

31–28	27–25	24	23	22	21	20	19–16	15–12	11–8	7	6–5	4	3–0
!=1111	0 0 0	P	U	1	W	1	!=1111	Rt	imm4H	1	0 1	1	imm4L
cond						o1	Rn				op2		

Encoding for the Offset variant

Applies when `(P == 1 && W == 0)`

```
LDRH{<c>}{<q>}  <Rt>, [<Rn> {, #{+/-}<imm>}]
```

Encoding for the Post-indexed variant

Applies when `(P == 0 && W == 0)`

```
LDRH{<c>}{<q>}  <Rt>, [<Rn>], #{+/-}<imm>
```

Encoding for the Pre-indexed variant

Applies when `(P == 1 && W == 1)`

```
LDRH{<c>}{<q>}  <Rt>, [<Rn>, #{+/-}<imm>]!
```

Decode for all variants of this encoding

```
if Rn == '1111' then SEE "LDRH (literal)";
if P == '0' && W == '1' then SEE "LDRHT";
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant bits(32) imm32 = ZeroExtend(imm4H:imm4L, 32);
constant boolean index = (P == '1');
constant boolean add = (U == '1');
constant boolean wback = (P == '0') || (W == '1');
if t == 15 || (wback && n == t) then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `wback && n == t`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such an instruction, the base address might be corrupted so that the instruction cannot be repeated.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T1

Encoding

```
LDRH{<c>}{<q>}  <Rt>, [<Rn> {, #{+}<imm>}]
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant bits(32) imm32 = ZeroExtend(imm5:'0', 32);
constant boolean index = TRUE;
constant boolean add = TRUE;
constant boolean wback = FALSE;
```

T2

Encoding

```
LDRH{<c>}{<q>}  <Rt>, [<Rn> {, #{+}<imm>}]
```

```
LDRH{<c>}.W  <Rt>, [<Rn> {, #{+}<imm>}] // (<Rt>, <Rn>, <imm> can be represented in T1)
```

Decode for this encoding

```
if Rt == '1111' then SEE "PLD (immediate)";
if Rn == '1111' then SEE "LDRH (literal)";
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant bits(32) imm32 = ZeroExtend(imm12, 32);
constant boolean index = TRUE;
constant boolean add = TRUE;
constant boolean wback = FALSE;
// Armv8-A removes UNPREDICTABLE for R13
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T3

Encoding for the Offset variant

Applies when `(Rt != 1111 && P == 1 && U == 0 && W == 0)`

```
LDRH{<c>}{<q>}  <Rt>, [<Rn> {, #-<imm>}]
```

Encoding for the Post-indexed variant

Applies when `(P == 0 && W == 1)`

```
LDRH{<c>}{<q>}  <Rt>, [<Rn>], #{+/-}<imm>
```

Encoding for the Pre-indexed variant

Applies when `(P == 1 && W == 1)`

```
LDRH{<c>}{<q>}  <Rt>, [<Rn>, #{+/-}<imm>]!
```

Decode for all variants of this encoding

```
if Rn == '1111' then SEE "LDRH (literal)";
if Rt == '1111' && P == '1' && U == '0' && W == '0' then SEE "PLDW (immediate)";
if P == '1' && U == '1' && W == '0' then SEE "LDRHT";
if P == '0' && W == '0' then UNDEFINED;
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant bits(32) imm32 = ZeroExtend(imm8, 32);
constant boolean index = (P == '1');
constant boolean add = (U == '1');
constant boolean wback = (W == '1');
if (t == 15 && W == '1') || (wback && n == t) then UNPREDICTABLE;
// Armv8-A removes UNPREDICTABLE for R13
```

CONSTRAINED UNPREDICTABLE behavior

If `wback && n == t`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such an instruction, the base address might be corrupted so that the instruction cannot be repeated.

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

Is the general-purpose register to be transferred, encoded in the “Rt” field.

<Rn>

For the “A1 Offset”, “A1 Post-indexed”, “A1 Pre-indexed”, “T2”, “T3 Offset”, “T3 Post-indexed”, and “T3 Pre-indexed” variants: is the general-purpose base register, encoded in the “Rn” field. For PC use see LDRH (literal).

For the “T1” variant: is the general-purpose base register, encoded in the “Rn” field.

+/-

Specifies the offset is added to or subtracted from the base register, defaulting to + if omitted and encoded in “U”:

U	+/-
0	-
1	+

<imm>

For the “A1 Offset”, “A1 Post-indexed”, and “A1 Pre-indexed” variants: is the 8-bit unsigned immediate byte offset, in the range 0 to 255, defaulting to 0 if omitted, and encoded in the “imm4H:imm4L” field.

For the “T1” variant: is the optional positive unsigned immediate byte offset, a multiple of 2, in the range 0 to 62, defaulting to 0 and encoded in the “imm5” field as <imm>/2.

For the “T2” variant: is an optional 12-bit unsigned immediate byte offset, in the range 0 to 4095, defaulting to 0 and encoded in the “imm12” field.

For the “T3 Offset”, “T3 Post-indexed”, and “T3 Pre-indexed” variants: is an 8-bit unsigned immediate byte offset, in the range 0 to 255, defaulting to 0 if omitted, and encoded in the “imm8” field.

+

Specifies the offset is added to the base register.

Operation

```
if CurrentInstrSet() == InstrSet_A32 then
    if ConditionPassed() then
        EncodingSpecificOperations();
        constant bits(32) offset_addr = if add then (R[n] + imm32) else (R[n] - imm32);
        constant bits(32) address = if index then offset_addr else R[n];
        constant bits(16) data = MemU[address,2];
        if wback then R[n] = offset_addr;
        R[t] = ZeroExtend(data, 32);
else
    if ConditionPassed() then
        EncodingSpecificOperations();
        constant bits(32) offset_addr = if add then (R[n] + imm32) else (R[n] - imm32);
        constant bits(32) address = if index then offset_addr else R[n];
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
constant bits(16) data = MemU[address,2];
if wback then R[n] = offset_addr;
R[t] = ZeroExtend(data, 32);
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.88 LDRH (literal)

Load Register Halfword (literal) calculates an address from the PC value and an immediate offset, loads a halfword from memory, zero-extends it to form a 32-bit word, and writes it to a register. For information about memory accesses see Memory accesses.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding

Applies when `(!(P == '0' && W == '1'))`

```
LDRH{<c>}{<q>}  <Rt>, <label> // (Normal form)
```

```
LDRH{<c>}{<q>}  <Rt>, [PC, #{+/-}<imm>] // (Alternative form)
```

Decode for this encoding

```
if P == '0' && W == '1' then SEE "LDRHT";
constant integer t = UInt(Rt);
constant bits(32) imm32 = ZeroExtend(imm4H:imm4L, 32);
constant boolean add = (U == '1');
constant boolean wback = (P == '0') || (W == '1');
if t == 15 || wback then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `wback`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction executes with the additional decode: wback = FALSE;.
- The instruction treats bit[24] as the P bit, and bit[21] as the writeback (W) bit, and uses the same addressing mode as described in LDRH (immediate). The instruction uses post-indexed addressing when P == ‘0’ and uses pre-indexed addressing otherwise. The instruction is handled as described in Using R15.

T1

Encoding

```
LDRH{<c>}{<q>}  <Rt>, <label> // (Preferred syntax)
```

```
LDRH{<c>}{<q>}  <Rt>, [PC, #{+/-}<imm>] // (Alternative syntax)
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for this encoding

```
if Rt == '1111' then SEE "PLD (literal)";
constant integer t = UInt(Rt);
constant bits(32) imm32 = ZeroExtend(imm12, 32);
constant boolean add = (U == '1');
// Armv8-A removes UNPREDICTABLE for R13
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

Is the general-purpose register to be transferred, encoded in the "Rt" field.

<label>

For the "A1" variant: the label of the literal data item that is to be loaded into `<Rt>`. The assembler calculates the required value of the offset from the `Align(PC, 4)` value of the instruction to this label. Any value in the range -255 to 255 is permitted. If the offset is zero or positive, `imm32` is equal to the offset and `add` == `TRUE`, encoded as U == 1. If the offset is negative, `imm32` is equal to minus the offset and `add` == `FALSE`, encoded as U == 0.

For the "T1" variant: the label of the literal data item that is to be loaded into `<Rt>`. The assembler calculates the required value of the offset from the `Align(PC, 4)` value of the instruction to this label. Permitted values of the offset are -4095 to 4095.

If the offset is zero or positive, `imm32` is equal to the offset and `add` == `TRUE`, encoded as U == 1.

If the offset is negative, `imm32` is equal to minus the offset and `add` == `FALSE`, encoded as U == 0.

+/-

Specifies the offset is added to or subtracted from the base register, defaulting to + if omitted and encoded in "U":

U	+/-
0	-
1	+

<imm>

For the "A1" variant: is the 8-bit unsigned immediate byte offset, in the range 0 to 255, defaulting to 0 if omitted, and encoded in the "imm4H:imm4L" field.

For the "T1" variant: is a 12-bit unsigned immediate byte offset, in the range 0 to 4095, encoded in the "imm12" field.

The alternative syntax permits the addition or subtraction of the offset and the immediate offset to be specified separately, including permitting a subtraction of 0 that cannot be specified using the normal syntax. For more information, see Use of labels in UAL instruction syntax.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) base = Align(PC32,4);
    constant bits(32) address = if add then (base + imm32) else (base - imm32);
    constant bits(16) data = MemU[address,2];
    R[t] = ZeroExtend(data, 32);
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.89 LDRH (register)

Load Register Halfword (register) calculates an address from a base register value and an offset register value, loads a halfword from memory, zero-extends it to form a 32-bit word, and writes it to a register. The offset register value can be shifted left by 0, 1, 2, or 3 bits. For information about memory accesses see Memory accesses.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1 and T2).

A1

31–28	27–25	24	23	22	21	20	19–16	15–12	11–7	6–5	4	3–0
!=1111	0 0 0	P	U	0	W	1	Rn	Rt	(0)(0)(0)(0)1	0 1	1	Rm
cond						o1				op2		

Encoding for the Offset variant

Applies when `(P == 1 && W == 0)`

```
LDRH{<c>}{<q>}  <Rt>, [<Rn>, {+/-}<Rm>]
```

Encoding for the Post-indexed variant

Applies when `(P == 0 && W == 0)`

```
LDRH{<c>}{<q>}  <Rt>, [<Rn>], {+/-}<Rm>
```

Encoding for the Pre-indexed variant

Applies when `(P == 1 && W == 1)`

```
LDRH{<c>}{<q>}  <Rt>, [<Rn>, {+/-}<Rm>]!
```

Decode for all variants of this encoding

```
if P == '0' && W == '1' then SEE "LDRHT";
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean index = (P == '1');
constant boolean add = (U == '1');
constant boolean wback = (P == '0') || (W == '1');
constant SRType shift_t = SRType_LSL;
constant integer shift_n = 0;
if t == 15 || m == 15 then UNPREDICTABLE;
if wback && (n == 15 || n == t) then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `wback && n == t`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such as instruction, the base address might be corrupted so that the instruction cannot be repeated.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T1

Encoding

```
LDRH{<c>}{<q>}  <Rt>, [<Rn>, {+}<Rm>]
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean index = TRUE;
constant boolean add = TRUE;
constant boolean wback = FALSE;
constant SRType shift_t = SRType_LSL;
constant integer shift_n = 0;
```

T2

Encoding

```
LDRH{<c>}{<q>}  <Rt>, [<Rn>, {+}<Rm>{, LSL #<imm>}]
```

```
LDRH{<c>}.W  <Rt>, [<Rn>, {+}<Rm>] // (<Rt>, <Rn>, <Rm> can be represented in T1)
```

Decode for this encoding

```
if Rn == '1111' then SEE "LDRH (literal)";
if Rt == '1111' then SEE "PLDW (register)";
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean index = TRUE;
constant boolean add = TRUE;
constant boolean wback = FALSE;
constant SRType shift_t = SRType_LSL;
constant integer shift_n = UInt(imm2);
// Armv8-A removes UNPREDICTABLE for R13
if m == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

Is the general-purpose register to be transferred, encoded in the “Rt” field.

<Rn>

For the “Offset”, “Post-indexed”, and “Pre-indexed” variants: is the general-purpose base register, encoded in the “Rn” field. The PC can be used in the offset variant.

For the “T1” and “T2” variants: is the general-purpose base register, encoded in the “Rn” field.

+/-

Specifies the index register is added to or subtracted from the base register, defaulting to + if omitted and encoded in “U”:

U	+/-
0	-
1	+

<Rm>

Is the general-purpose index register, encoded in the “Rm” field.

+

Specifies the index register is added to the base register.

<imm>

If present, the size of the left shift to apply to the value from `<Rm>`, in the range 1-3. `<imm>` is encoded in imm2. If absent, no shift is specified and imm2 is encoded as `0b00`.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) offset = Shift(R[m], shift_t, shift_n, PSTATE.C);
    constant bits(32) offset_addr = if add then (R[n] + offset) else (R[n] - offset);
    constant bits(32) address = if index then offset_addr else R[n];
    constant bits(16) data = MemU[address,2];
    if wback then R[n] = offset_addr;
    R[t] = ZeroExtend(data, 32);
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.90 LDRHT

Load Register Halfword Unprivileged loads a halfword from memory, zero-extends it to form a 32-bit word, and writes it to a register. For information about memory accesses see Memory accesses.

The memory access is restricted as if the PE were running in User mode. This makes no difference if the PE is actually running in User mode.

`LDRHT` is UNPREDICTABLE in Hyp mode.

The T32 instruction uses an offset addressing mode, that calculates the address used for the memory access from a base register value and an immediate offset, and leaves the base register unchanged.

The A32 instruction uses a post-indexed addressing mode, that uses a base register value as the address for the memory access, and calculates a new address from a base register value and an offset and writes it back to the base register. The offset can be an immediate value or a register value.

It has encodings from the following instruction sets: A32 (A1 and A2) and T32 (T1).

A1

Encoding

```
LDRHT{<c>}{<q>}  <Rt>, [<Rn>] {, #{+/-}<imm>}
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant boolean postindex = TRUE;
constant boolean add = (U == '1');
constant boolean register_form = FALSE;
constant bits(32) imm32 = ZeroExtend(imm4H:imm4L, 32);
constant integer m = integer UNKNOWN;
if t == 15 || n == 15 || n == t then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `n == 15`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction uses post-indexed addressing with the base register as PC. This is handled as described in Using R15.
- The instruction is treated as if bit[24] == '1' and bit[21] == '0'. The instruction uses immediate offset addressing with the base register as PC, without writeback.

If `n == t && n != 15`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such as instruction, the base address might be corrupted so that the instruction cannot be repeated.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

A2

31–28	27	26	25	24	23	22	21	20	19–16	15–12	11–8	7	6	5	4	3–0
!=1111	0	0	0	0	U	0	1	1	Rn	Rt	(0)(0)(0)(0)	1	0	1	1	Rm
cond				P		W		o1						op2		

Encoding

```
LDRHT{<c>}{<q>}  <Rt>, [<Rn>], {+/-}<Rm>
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean postindex = TRUE;
constant boolean add = (U == '1');
constant boolean register_form = TRUE;
constant bits(32) imm32 = bits(32) UNKNOWN;
if t == 15 || n == 15 || n == t || m == 15 then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `n == t && n != 15`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such as instruction, the base address might be corrupted so that the instruction cannot be repeated.

T1

15–13	12–9	8	7	6	5	4	3–0	15–12	11–8	7–0
1 1 1	1 1 0 0	0	0	0	1	1	!=1111	Rt	1 1 1 0	imm8
				size		L	Rn			

Encoding

```
LDRHT{<c>}{<q>}  <Rt>, [<Rn> {, #{+}<imm>}]
```

Decode for this encoding

```
if Rn == '1111' then SEE "LDRH (literal)";
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant boolean postindex = FALSE;
constant boolean add = TRUE;
constant boolean register_form = FALSE;
constant bits(32) imm32 = ZeroExtend(imm8, 32);
constant integer m = integer UNKNOWN;
// Armv8-A removes UNPREDICTABLE for R13
if t == 15 then UNPREDICTABLE;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

Is the general-purpose register to be transferred, encoded in the “Rt” field.

<Rn>

Is the general-purpose base register, encoded in the “Rn” field.

+/-

For the “A1” variant: specifies the offset is added to or subtracted from the base register, defaulting to + if omitted and encoded in “U”:

U	+/-
0	-
1	+

For the “A2” variant: specifies the index register is added to or subtracted from the base register, defaulting to + if omitted and encoded in “U”:

U	+/-
0	-
1	+

<imm>

For the “A1” variant: is the 8-bit unsigned immediate byte offset, in the range 0 to 255, defaulting to 0 if omitted, and encoded in the “imm4H:imm4L” field.

For the “T1” variant: is an optional 8-bit unsigned immediate byte offset, in the range 0 to 255, defaulting to 0 and encoded in the “imm8” field.

<Rm>

Is the general-purpose index register, encoded in the “Rm” field.

+

Specifies the offset is added to the base register.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    if PSTATE.EL == EL2 then UNPREDICTABLE;                    // Hyp mode
    constant bits(32) offset = if register_form then R[m] else imm32;
    constant bits(32) offset_addr = if add then (R[n] + offset) else (R[n] - offset);
    constant bits(32) address = if postindex then R[n] else offset_addr;
    constant bits(16) data = MemU_unpriv[address,2];
    if postindex then R[n] = offset_addr;
    R[t] = ZeroExtend(data, 32);
```

CONSTRAINED UNPREDICTABLE behavior

If `PSTATE.EL == EL2`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction executes as `LDRH` (immediate).

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.91 LDRSB (immediate)

Load Register Signed Byte (immediate) calculates an address from a base register value and an immediate offset, loads a byte from memory, sign-extends it to form a 32-bit word, and writes it to a register. It can use offset, post-indexed, or pre-indexed addressing. For information about memory accesses see Memory accesses.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1 and T2).

A1

31–28	27–25	24	23	22	21	20	19–16	15–12	11–8	7	6–5	4	3–0
!=1111	0 0 0	P	U	1	W	1	!=1111	Rt	imm4H	1	1 0	1	imm4L
cond						o1	Rn				op2		

Encoding for the Offset variant

Applies when `(P == 1 && W == 0)`

```
LDRSB{<c>}{<q>}  <Rt>, [<Rn> {, #{+/-}<imm>}]
```

Encoding for the Post-indexed variant

Applies when `(P == 0 && W == 0)`

```
LDRSB{<c>}{<q>}  <Rt>, [<Rn>], #{+/-}<imm>
```

Encoding for the Pre-indexed variant

Applies when `(P == 1 && W == 1)`

```
LDRSB{<c>}{<q>}  <Rt>, [<Rn>, #{+/-}<imm>]!
```

Decode for all variants of this encoding

```
if Rn == '1111' then SEE "LDRSB (literal)";
if P == '0' && W == '1' then SEE "LDRSBT";
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant bits(32) imm32 = ZeroExtend(imm4H:imm4L, 32);
constant boolean index = (P == '1');
constant boolean add = (U == '1');
constant boolean wback = (P == '0') || (W == '1');
if t == 15 || (wback && n == t) then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `wback && n == t`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such an instruction, the base address might be corrupted so that the instruction cannot be repeated.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T1

15–13	12–9	8–7	6–5	4	3–0	15–12	11–0
1 1 1	1 1 0 0	1 1	0 0	1	!=1111	!=1111	imm12
			size		Rn	Rt	

Encoding

```
LDRSB{<c>}{<q>}  <Rt>, [<Rn> {, #{+}<imm>}]
```

Decode for this encoding

```
if Rt == '1111' then SEE "PLI";
if Rn == '1111' then SEE "LDRSB (literal)";
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant bits(32) imm32 = ZeroExtend(imm12, 32);
constant boolean index = TRUE;
constant boolean add = TRUE;
constant boolean wback = FALSE;
// Armv8-A removes UNPREDICTABLE for R13
```

T2

15–7	6–5	4	3–0	15–12	11	10	9	8	7–0
1 1 1 1 1 0 0 1 0	0 0	1	!=1111	Rt	1	P	U	W	imm8
	size		Rn						

Encoding for the Offset variant

Applies when `(Rt != 1111 && P == 1 && U == 0 && W == 0)`

```
LDRSB{<c>}{<q>}  <Rt>, [<Rn> {, #-<imm>}]
```

Encoding for the Post-indexed variant

Applies when `(P == 0 && W == 1)`

```
LDRSB{<c>}{<q>}  <Rt>, [<Rn>], #{+/-}<imm>
```

Encoding for the Pre-indexed variant

Applies when `(P == 1 && W == 1)`

```
LDRSB{<c>}{<q>}  <Rt>, [<Rn>, #{+/-}<imm>]!
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for all variants of this encoding

```
if Rt == '1111' && P == '1' && U == '0' && W == '0' then SEE "PLI";
if Rn == '1111' then SEE "LDRSB (literal)";
if P == '1' && U == '1' && W == '0' then SEE "LDRSBT";
if P == '0' && W == '0' then UNDEFINED;
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant bits(32) imm32 = ZeroExtend(imm8, 32);
constant boolean index = (P == '1');
constant boolean add = (U == '1');
constant boolean wback = (W == '1');
if (t == 15 && W == '1') || (wback && n == t) then UNPREDICTABLE;
// Armv8-A removes UNPREDICTABLE for R13
```

CONSTRAINED UNPREDICTABLE behavior

If `wback && n == t`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such an instruction, the base address might be corrupted so that the instruction cannot be repeated.

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

Is the general-purpose register to be transferred, encoded in the “Rt” field.

<Rn>

Is the general-purpose base register, encoded in the “Rn” field. For PC use see LDRSB (literal).

+/-

Specifies the offset is added to or subtracted from the base register, defaulting to + if omitted and encoded in “U”:

U	+/-
0	-
1	+

<imm>

For the “A1 Offset”, “A1 Post-indexed”, and “A1 Pre-indexed” variants: is the 8-bit unsigned immediate byte offset, in the range 0 to 255, defaulting to 0 if omitted, and encoded in the “imm4H:imm4L” field.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

For the “T1” variant: is an optional 12-bit unsigned immediate byte offset, in the range 0 to 4095, defaulting to 0 and encoded in the “imm12” field.

For the “T2 Offset”, “T2 Post-indexed”, and “T2 Pre-indexed” variants: is an 8-bit unsigned immediate byte offset, in the range 0 to 255, defaulting to 0 if omitted, and encoded in the “imm8” field.

+

Specifies the offset is added to the base register.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) offset_addr = if add then (R[n] + imm32) else (R[n] - imm32);
    constant bits(32) address = if index then offset_addr else R[n];
    R[t] = SignExtend(MemU[address,1], 32);
    if wback then R[n] = offset_addr;
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.92 LDRSB (literal)

Load Register Signed Byte (literal) calculates an address from the PC value and an immediate offset, loads a byte from memory, sign-extends it to form a 32-bit word, and writes it to a register. For information about memory accesses see Memory accesses.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding

Applies when `(!(P == '0' && W == '1'))`

```
LDRSB{<c>}{<q>}  <Rt>, <label> // (Normal form)
```

```
LDRSB{<c>}{<q>}  <Rt>, [PC, #{+/-}<imm>] // (Alternative form)
```

Decode for this encoding

```
if P == '0' && W == '1' then SEE "LDRSBT";
constant integer t = UInt(Rt);
constant bits(32) imm32 = ZeroExtend(imm4H:imm4L, 32);
constant boolean add = (U == '1');
constant boolean wback = (P == '0') || (W == '1');
if t == 15 || wback then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `wback`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction executes with the additional decode: wback = FALSE;.
- The instruction treats bit[24] as the P bit, and bit[21] as the writeback (W) bit, and uses the same addressing mode as described in LDRSB (immediate). The instruction uses post-indexed addressing when P == ‘0’ and uses pre-indexed addressing otherwise. The instruction is handled as described in Using R15.

T1

Encoding

```
LDRSB{<c>}{<q>}  <Rt>, <label> // (Preferred syntax)
```

```
LDRSB{<c>}{<q>}  <Rt>, [PC, #{+/-}<imm>] // (Alternative syntax)
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for this encoding

```
if Rt == '1111' then SEE "PLI";
constant integer t = UInt(Rt);
constant bits(32) imm32 = ZeroExtend(imm12, 32);
constant boolean add = (U == '1');
// Armv8-A removes UNPREDICTABLE for R13
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

Is the general-purpose register to be transferred, encoded in the "Rt" field.

<label>

For the "A1" variant: the label of the literal data item that is to be loaded into `<Rt>`. The assembler calculates the required value of the offset from the `Align(PC, 4)` value of the instruction to this label. Any value in the range -255 to 255 is permitted. If the offset is zero or positive, `imm32` is equal to the offset and `add` == `TRUE`, encoded as U == 1. If the offset is negative, `imm32` is equal to minus the offset and `add` == `FALSE`, encoded as U == 0.

For the "T1" variant: the label of the literal data item that is to be loaded into `<Rt>`. The assembler calculates the required value of the offset from the `Align(PC, 4)` value of the instruction to this label. Permitted values of the offset are -4095 to 4095.

If the offset is zero or positive, `imm32` is equal to the offset and `add` == `TRUE`, encoded as U == 1.

If the offset is negative, `imm32` is equal to minus the offset and `add` == `FALSE`, encoded as U == 0.

+/-

Specifies the offset is added to or subtracted from the base register, defaulting to + if omitted and encoded in "U":

U	+/-
0	-
1	+

<imm>

For the "A1" variant: is the 8-bit unsigned immediate byte offset, in the range 0 to 255, defaulting to 0 if omitted, and encoded in the "imm4H:imm4L" field.

For the "T1" variant: is a 12-bit unsigned immediate byte offset, in the range 0 to 4095, encoded in the "imm12" field.

The alternative syntax permits the addition or subtraction of the offset and the immediate offset to be specified separately, including permitting a subtraction of 0 that cannot be specified using the normal syntax. For more information, see Use of labels in UAL instruction syntax.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) base = Align(PC32,4);
    constant bits(32) address = if add then (base + imm32) else (base - imm32);
    R[t] = SignExtend(MemU[address,1], 32);
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.93 LDRSB (register)

Load Register Signed Byte (register) calculates an address from a base register value and an offset register value, loads a byte from memory, sign-extends it to form a 32-bit word, and writes it to a register. The offset register value can be shifted left by 0, 1, 2, or 3 bits. For information about memory accesses see Memory accesses.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1 and T2).

A1

31–28	27–25	24	23	22	21	20	19–16	15–12	11–7	6–5	4	3–0
!=1111	0 0 0	P	U	0	W	1	Rn	Rt	(0)(0)(0)(0)1	1 0	1	Rm
cond						o1				op2		

Encoding for the Offset variant

Applies when `(P == 1 && W == 0)`

```
LDRSB{<c>}{<q>}  <Rt>, [<Rn>, {+/-}<Rm>]
```

Encoding for the Post-indexed variant

Applies when `(P == 0 && W == 0)`

```
LDRSB{<c>}{<q>}  <Rt>, [<Rn>], {+/-}<Rm>
```

Encoding for the Pre-indexed variant

Applies when `(P == 1 && W == 1)`

```
LDRSB{<c>}{<q>}  <Rt>, [<Rn>, {+/-}<Rm>]!
```

Decode for all variants of this encoding

```
if P == '0' && W == '1' then SEE "LDRSBT";
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean index = (P == '1');
constant boolean add = (U == '1');
constant boolean wback = (P == '0') || (W == '1');
constant SRType shift_t = SRType_LSL;
constant integer shift_n = 0;
if t == 15 || m == 15 then UNPREDICTABLE;
if wback && (n == 15 || n == t) then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `wback && n == t`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such as instruction, the base address might be corrupted so that the instruction cannot be repeated.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T1

Encoding

```
LDRSB{<c>}{<q>}  <Rt>, [<Rn>, {+}<Rm>]
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean index = TRUE;
constant boolean add = TRUE;
constant boolean wback = FALSE;
constant SRType shift_t = SRType_LSL;
constant integer shift_n = 0;
```

T2

Encoding

```
LDRSB{<c>}{<q>}  <Rt>, [<Rn>, {+}<Rm>{, LSL #<imm>}]
```

```
LDRSB{<c>}.W  <Rt>, [<Rn>, {+}<Rm>] // (<Rt>, <Rn>, <Rm> can be represented in T1)
```

Decode for this encoding

```
if Rt == '1111' then SEE "PLI";
if Rn == '1111' then SEE "LDRSB (literal)";
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean index = TRUE;
constant boolean add = TRUE;
constant boolean wback = FALSE;
constant SRType shift_t = SRType_LSL;
constant integer shift_n = UInt(imm2);
// Armv8-A removes UNPREDICTABLE for R13
if m == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

Is the general-purpose register to be transferred, encoded in the “Rt” field.

<Rn>

For the “Offset”, “Post-indexed”, and “Pre-indexed” variants: is the general-purpose base register, encoded in the “Rn” field. The PC can be used in the offset variant.

For the “T1” and “T2” variants: is the general-purpose base register, encoded in the “Rn” field.

+/-

Specifies the index register is added to or subtracted from the base register, defaulting to + if omitted and encoded in “U”:

U	+/-
0	-
1	+

<Rm>

Is the general-purpose index register, encoded in the “Rm” field.

+

Specifies the index register is added to the base register.

<imm>

If present, the size of the left shift to apply to the value from `<Rm>`, in the range 1-3. `<imm>` is encoded in imm2. If absent, no shift is specified and imm2 is encoded as `0b00`.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) offset = Shift(R[m], shift_t, shift_n, PSTATE.C);
    constant bits(32) offset_addr = if add then (R[n] + offset) else (R[n] - offset);
    constant bits(32) address = if index then offset_addr else R[n];
    R[t] = SignExtend(MemU[address,1], 32);
    if wback then R[n] = offset_addr;
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.94 LDRSBT

Load Register Signed Byte Unprivileged loads a byte from memory, sign-extends it to form a 32-bit word, and writes it to a register. For information about memory accesses see Memory accesses.

The memory access is restricted as if the PE were running in User mode. This makes no difference if the PE is actually running in User mode.

`LDRSBT` is UNPREDICTABLE in Hyp mode.

The T32 instruction uses an offset addressing mode, that calculates the address used for the memory access from a base register value and an immediate offset, and leaves the base register unchanged.

The A32 instruction uses a post-indexed addressing mode, that uses a base register value as the address for the memory access, and calculates a new address from a base register value and an offset and writes it back to the base register. The offset can be an immediate value or a register value.

It has encodings from the following instruction sets: A32 (A1 and A2) and T32 (T1).

A1

Encoding

```
LDRSBT{<c>}{<q>}  <Rt>, [<Rn>] {, #{+/-}<imm>}
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant boolean postindex = TRUE;
constant boolean add = (U == '1');
constant boolean register_form = FALSE;
constant bits(32) imm32 = ZeroExtend(imm4H:imm4L, 32);
constant integer m = integer UNKNOWN;
if t == 15 || n == 15 || n == t then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `n == 15`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction uses post-indexed addressing with the base register as PC. This is handled as described in Using R15.
- The instruction is treated as if bit[24] == '1' and bit[21] == '0'. The instruction uses immediate offset addressing with the base register as PC, without writeback.

If `n == t && n != 15`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such as instruction, the base address might be corrupted so that the instruction cannot be repeated.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

A2

31–28	27	26	25	24	23	22	21	20	19–16	15–12	11–8	7	6	5	4	3–0
!=1111	0	0	0	0	U	0	1	1	Rn	Rt	(0)(0)(0)(0)	1	1	0	1	Rm
cond				P		W		o1						op2		

Encoding

```
LDRSBT{<c>}{<q>}  <Rt>, [<Rn>], {+/-}<Rm>
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean postindex = TRUE;
constant boolean add = (U == '1');
constant boolean register_form = TRUE;
constant bits(32) imm32 = bits(32) UNKNOWN;
if t == 15 || n == 15 || n == t || m == 15 then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `n == t && n != 15`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such as instruction, the base address might be corrupted so that the instruction cannot be repeated.

T1

15–13	12–9	8	7	6	5	4	3–0	15–12	11–8	7–0
1 1 1	1 1 0 0	1	0	0	0	1	!=1111	Rt	1 1 1 0	imm8
				size			Rn			

Encoding

```
LDRSBT{<c>}{<q>}  <Rt>, [<Rn> {, #{+}<imm>}]
```

Decode for this encoding

```
if Rn == '1111' then SEE "LDRSB (literal)";
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant boolean postindex = FALSE;
constant boolean add = TRUE;
constant boolean register_form = FALSE;
constant bits(32) imm32 = ZeroExtend(imm8, 32);
constant integer m = integer UNKNOWN;
// Armv8-A removes UNPREDICTABLE for R13
if t == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

Is the general-purpose register to be transferred, encoded in the “Rt” field.

<Rn>

Is the general-purpose base register, encoded in the “Rn” field.

+/-

For the “A1” variant: specifies the offset is added to or subtracted from the base register, defaulting to + if omitted and encoded in “U”:

U	+/-
0	-
1	+

For the “A2” variant: specifies the index register is added to or subtracted from the base register, defaulting to + if omitted and encoded in “U”:

U	+/-
0	-
1	+

<imm>

For the “A1” variant: is the 8-bit unsigned immediate byte offset, in the range 0 to 255, defaulting to 0 if omitted, and encoded in the “imm4H:imm4L” field.

For the “T1” variant: is an optional 8-bit unsigned immediate byte offset, in the range 0 to 255, defaulting to 0 and encoded in the “imm8” field.

<Rm>

Is the general-purpose index register, encoded in the “Rm” field.

+

Specifies the offset is added to the base register.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    if PSTATE.EL == EL2 then UNPREDICTABLE;                    // Hyp mode
    constant bits(32) offset = if register_form then R[m] else imm32;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
constant bits(32) offset_addr = if add then (R[n] + offset) else (R[n] - offset);
constant bits(32) address = if postindex then R[n] else offset_addr;
R[t] = SignExtend(MemU_unpriv[address,1], 32);
if postindex then R[n] = offset_addr;
```

CONSTRAINED UNPREDICTABLE behavior

If `PSTATE.EL == EL2`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction executes as `LDRSB` (immediate).

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.95 LDRSH (immediate)

Load Register Signed Halfword (immediate) calculates an address from a base register value and an immediate offset, loads a halfword from memory, sign-extends it to form a 32-bit word, and writes it to a register. It can use offset, post-indexed, or pre-indexed addressing. For information about memory accesses see Memory accesses.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1 and T2).

A1

Encoding for the Offset variant

Applies when `(P == 1 && W == 0)`

```
LDRSH{<c>}{<q>}  <Rt>, [<Rn> {, #{+/-}<imm>}]
```

Encoding for the Post-indexed variant

Applies when `(P == 0 && W == 0)`

```
LDRSH{<c>}{<q>}  <Rt>, [<Rn>], #{+/-}<imm>
```

Encoding for the Pre-indexed variant

Applies when `(P == 1 && W == 1)`

```
LDRSH{<c>}{<q>}  <Rt>, [<Rn>, #{+/-}<imm>]!
```

Decode for all variants of this encoding

```
if Rn == '1111' then SEE "LDRSH (literal)";
if P == '0' && W == '1' then SEE "LDRSHT";
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant bits(32) imm32 = ZeroExtend(imm4H:imm4L, 32);
constant boolean index = (P == '1');
constant boolean add = (U == '1');
constant boolean wback = (P == '0') || (W == '1');
if t == 15 || (wback && n == t) then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `wback && n == t`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such an instruction, the base address might be corrupted so that the instruction cannot be repeated.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T1

15–13	12–9	8–7	6–5	4	3–0	15–12	11–0
1 1 1	1 1 0 0	1 1	0 1	1	!=1111	!=1111	imm12
			size		Rn	Rt	

Encoding

```
LDRSH{<c>}{<q>}  <Rt>, [<Rn> {, #{+}<imm>}]
```

Decode for this encoding

```
if Rn == '1111' then SEE "LDRSH (literal)";
if Rt == '1111' then SEE "Related instructions";
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant bits(32) imm32 = ZeroExtend(imm12, 32);
constant boolean index = TRUE;
constant boolean add = TRUE;
constant boolean wback = FALSE;
// Armv8-A removes UNPREDICTABLE for R13
```

T2

15–7	6–5	4	3–0	15–12	11	10	9	8	7–0
1 1 1 1 1 0 0 1 0	0 1	1	!=1111	Rt	1	P	U	W	imm8
	size		Rn						

Encoding for the Offset variant

Applies when `(Rt != 1111 && P == 1 && U == 0 && W == 0)`

```
LDRSH{<c>}{<q>}  <Rt>, [<Rn> {, #-<imm>}]
```

Encoding for the Post-indexed variant

Applies when `(P == 0 && W == 1)`

```
LDRSH{<c>}{<q>}  <Rt>, [<Rn>], #{+/-}<imm>
```

Encoding for the Pre-indexed variant

Applies when `(P == 1 && W == 1)`

```
LDRSH{<c>}{<q>}  <Rt>, [<Rn>, #{+/-}<imm>]!
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for all variants of this encoding

```
if Rn == '1111' then SEE "LDRSH (literal)";
if Rt == '1111' && P == '1' && U == '0' && W == '0' then SEE "Related instructions";
if P == '1' && U == '1' && W == '0' then SEE "LDRSHT";
if P == '0' && W == '0' then UNDEFINED;
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant bits(32) imm32 = ZeroExtend(imm8, 32);
constant boolean index = (P == '1');
constant boolean add = (U == '1');
constant boolean wback = (W == '1');
if (t == 15 && W == '1') || (wback && n == t) then UNPREDICTABLE;
// Armv8-A removes UNPREDICTABLE for R13
```

CONSTRAINED UNPREDICTABLE behavior

If `wback && n == t`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such an instruction, the base address might be corrupted so that the instruction cannot be repeated.

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Related instructions: Load/store single.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

Is the general-purpose register to be transferred, encoded in the “Rt” field.

<Rn>

Is the general-purpose base register, encoded in the “Rn” field. For PC use see LDRSH (literal).

+/-

Specifies the offset is added to or subtracted from the base register, defaulting to + if omitted and encoded in “U”:

U	+/-
0	-
1	+

<imm>

For the “A1 Offset”, “A1 Post-indexed”, and “A1 Pre-indexed” variants: is the 8-bit unsigned immediate byte offset, in the range 0 to 255, defaulting to 0 if omitted, and encoded in the “imm4H:imm4L” field.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

For the “T1” variant: is an optional 12-bit unsigned immediate byte offset, in the range 0 to 4095, defaulting to 0 and encoded in the “imm12” field.

For the “T2 Offset”, “T2 Post-indexed”, and “T2 Pre-indexed” variants: is an 8-bit unsigned immediate byte offset, in the range 0 to 255, defaulting to 0 if omitted, and encoded in the “imm8” field.

+

Specifies the offset is added to the base register.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) offset_addr = if add then (R[n] + imm32) else (R[n] - imm32);
    constant bits(32) address = if index then offset_addr else R[n];
    constant bits(16) data = MemU[address,2];
    if wback then R[n] = offset_addr;
    R[t] = SignExtend(data, 32);
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.96 LDRSH (literal)

Load Register Signed Halfword (literal) calculates an address from the PC value and an immediate offset, loads a halfword from memory, sign-extends it to form a 32-bit word, and writes it to a register. For information about memory accesses see Memory accesses.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding

Applies when `(!(P == '0' && W == '1'))`

```
LDRSH{<c>}{<q>}  <Rt>, <label> // (Normal form)
```

```
LDRSH{<c>}{<q>}  <Rt>, [PC, #{+/-}<imm>] // (Alternative form)
```

Decode for this encoding

```
if P == '0' && W == '1' then SEE "LDRSHT";
constant integer t = UInt(Rt);
constant bits(32) imm32 = ZeroExtend(imm4H:imm4L, 32);
constant boolean add = (U == '1');
constant boolean wback = (P == '0') || (W == '1');
if t == 15 || wback then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `wback`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction executes with the additional decode: wback = FALSE;.
- The instruction treats bit[24] as the P bit, and bit[21] as the writeback (W) bit, and uses the same addressing mode as described in LDRSH (immediate). The instruction uses post-indexed addressing when P == ‘0’ and uses pre-indexed addressing otherwise. The instruction is handled as described in Using R15.

T1

Encoding

```
LDRSH{<c>}{<q>}  <Rt>, <label> // (Preferred syntax)
```

```
LDRSH{<c>}{<q>}  <Rt>, [PC, #{+/-}<imm>] // (Alternative syntax)
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for this encoding

```
if Rt == '1111' then SEE "Related instructions";
constant integer t = UInt(Rt);
constant bits(32) imm32 = ZeroExtend(imm12, 32);
constant boolean add = (U == '1');
// Armv8-A removes UNPREDICTABLE for R13
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Related instructions: Load, signed (literal).

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

Is the general-purpose register to be transferred, encoded in the "Rt" field.

<label>

For the "A1" variant: the label of the literal data item that is to be loaded into `<Rt>`. The assembler calculates the required value of the offset from the `Align(PC, 4)` value of the instruction to this label. Any value in the range -255 to 255 is permitted. If the offset is zero or positive, `imm32` is equal to the offset and `add` == `TRUE`, encoded as U == 1. If the offset is negative, `imm32` is equal to minus the offset and `add` == `FALSE`, encoded as U == 0.

For the "T1" variant: the label of the literal data item that is to be loaded into `<Rt>`. The assembler calculates the required value of the offset from the `Align(PC, 4)` value of the instruction to this label. Permitted values of the offset are -4095 to 4095.

If the offset is zero or positive, `imm32` is equal to the offset and `add` == `TRUE`, encoded as U == 1.

If the offset is negative, `imm32` is equal to minus the offset and `add` == `FALSE`, encoded as U == 0.

+/-

Specifies the offset is added to or subtracted from the base register, defaulting to + if omitted and encoded in "U":

U	+/-
0	-
1	+

<imm>

For the "A1" variant: is the 8-bit unsigned immediate byte offset, in the range 0 to 255, defaulting to 0 if omitted, and encoded in the "imm4H:imm4L" field.

For the "T1" variant: is a 12-bit unsigned immediate byte offset, in the range 0 to 4095, encoded in the "imm12" field.

The alternative syntax permits the addition or subtraction of the offset and the immediate offset to be specified separately, including permitting a subtraction of 0 that cannot be specified using the normal syntax. For more information, see Use of labels in UAL instruction syntax.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) base = Align(PC32,4);
    constant bits(32) address = if add then (base + imm32) else (base - imm32);
    constant bits(16) data = MemU[address,2];
    R[t] = SignExtend(data, 32);
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.97 LDRSH (register)

Load Register Signed Halfword (register) calculates an address from a base register value and an offset register value, loads a halfword from memory, sign-extends it to form a 32-bit word, and writes it to a register. The offset register value can be shifted left by 0, 1, 2, or 3 bits. For information about memory accesses see Memory accesses.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1 and T2).

A1

31–28	27–25	24	23	22	21	20	19–16	15–12	11–7	6	5	4	3–0
!=1111 (cond)	0 0 0	P	U	0	W	1 (o1)	Rn	Rt	(0)(0)(0)(0)1	1	1 (op2)	1	Rm

Encoding for the Offset variant

Applies when `(P == 1 && W == 0)`

```
LDRSH{<c>}{<q>}  <Rt>, [<Rn>, {+/-}<Rm>]
```

Encoding for the Post-indexed variant

Applies when `(P == 0 && W == 0)`

```
LDRSH{<c>}{<q>}  <Rt>, [<Rn>], {+/-}<Rm>
```

Encoding for the Pre-indexed variant

Applies when `(P == 1 && W == 1)`

```
LDRSH{<c>}{<q>}  <Rt>, [<Rn>, {+/-}<Rm>]!
```

Decode for all variants of this encoding

```
if P == '0' && W == '1' then SEE "LDRSHT";
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean index = (P == '1');
constant boolean add = (U == '1');
constant boolean wback = (P == '0') || (W == '1');
constant SRType shift_t = SRType_LSL;
constant integer shift_n = 0;
if t == 15 || m == 15 then UNPREDICTABLE;
if wback && (n == 15 || n == t) then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `wback && n == t`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such as instruction, the base address might be corrupted so that the instruction cannot be repeated.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T1

Encoding

```
LDRSH{<c>}{<q>}  <Rt>, [<Rn>, {+}<Rm>]
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean index = TRUE;
constant boolean add = TRUE;
constant boolean wback = FALSE;
constant SRType shift_t = SRType_LSL;
constant integer shift_n = 0;
```

T2

Encoding

```
LDRSH{<c>}{<q>}  <Rt>, [<Rn>, {+}<Rm>{, LSL #<imm>}]
```

```
LDRSH{<c>}.W  <Rt>, [<Rn>, {+}<Rm>] // (<Rt>, <Rn>, <Rm> can be represented in T1)
```

Decode for this encoding

```
if Rn == '1111' then SEE "LDRSH (literal)";
if Rt == '1111' then SEE "Related instructions";
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean index = TRUE;
constant boolean add = TRUE;
constant boolean wback = FALSE;
constant SRType shift_t = SRType_LSL;
constant integer shift_n = UInt(imm2);
// Armv8-A removes UNPREDICTABLE for R13
if m == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Related instructions: Load/store, signed (register offset).

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

Is the general-purpose register to be transferred, encoded in the “Rt” field.

<Rn>

For the “Offset”, “Post-indexed”, and “Pre-indexed” variants: is the general-purpose base register, encoded in the “Rn” field. The PC can be used in the offset variant.

For the “T1” and “T2” variants: is the general-purpose base register, encoded in the “Rn” field.

+/-

Specifies the index register is added to or subtracted from the base register, defaulting to + if omitted and encoded in “U”:

U	+/-
0	-
1	+

<Rm>

Is the general-purpose index register, encoded in the “Rm” field.

+

Specifies the index register is added to the base register.

<imm>

If present, the size of the left shift to apply to the value from `<Rm>`, in the range 1-3. `<imm>` is encoded in imm2. If absent, no shift is specified and imm2 is encoded as `0b00`.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) offset = Shift(R[m], shift_t, shift_n, PSTATE.C);
    constant bits(32) offset_addr = if add then (R[n] + offset) else (R[n] - offset);
    constant bits(32) address = if index then offset_addr else R[n];
    constant bits(16) data = MemU[address,2];
    if wback then R[n] = offset_addr;
    R[t] = SignExtend(data, 32);
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.98 LDRSHT

Load Register Signed Halfword Unprivileged loads a halfword from memory, sign-extends it to form a 32-bit word, and writes it to a register. For information about memory accesses see Memory accesses.

The memory access is restricted as if the PE were running in User mode. This makes no difference if the PE is actually running in User mode.

`LDRSHT` is UNPREDICTABLE in Hyp mode.

The T32 instruction uses an offset addressing mode, that calculates the address used for the memory access from a base register value and an immediate offset, and leaves the base register unchanged.

The A32 instruction uses a post-indexed addressing mode, that uses a base register value as the address for the memory access, and calculates a new address from a base register value and an offset and writes it back to the base register. The offset can be an immediate value or a register value.

It has encodings from the following instruction sets: A32 (A1 and A2) and T32 (T1).

A1

Encoding

```
LDRSHT{<c>}{<q>}  <Rt>, [<Rn>] {, #{+/-}<imm>}
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant boolean postindex = TRUE;
constant boolean add = (U == '1');
constant boolean register_form = FALSE;
constant bits(32) imm32 = ZeroExtend(imm4H:imm4L, 32);
constant integer m = integer UNKNOWN;
if t == 15 || n == 15 || n == t then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `n == 15`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction uses post-indexed addressing with the base register as PC. This is handled as described in Using R15.
- The instruction is treated as if bit[24] == '1' and bit[21] == '0'. The instruction uses immediate offset addressing with the base register as PC, without writeback.

If `n == t && n != 15`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such as instruction, the base address might be corrupted so that the instruction cannot be repeated.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

A2

31–28	27	26	25	24	23	22	21	20	19–16	15–12	11–8	7	6	5	4	3–0
!=1111	0	0	0	0	U	0	1	1	Rn	Rt	(0)(0)(0)(0)	1	1	1	1	Rm
cond				P		W		o1						op2		

Encoding

```
LDRSHT{<c>}{<q>}  <Rt>, [<Rn>], {+/-}<Rm>
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean postindex = TRUE;
constant boolean add = (U == '1');
constant boolean register_form = TRUE;
constant bits(32) imm32 = bits(32) UNKNOWN;
if t == 15 || n == 15 || n == t || m == 15 then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `n == t && n != 15`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such as instruction, the base address might be corrupted so that the instruction cannot be repeated.

T1

15–13	12–9	8–7	6–5	4	3–0	15–12	11–8	7–0
1 1 1	1 1 0 0	1 0	0 1	1	!=1111	Rt	1 1 1 0	imm8
			size		Rn			

Encoding

```
LDRSHT{<c>}{<q>}  <Rt>, [<Rn> {, #{+}<imm>}]
```

Decode for this encoding

```
if Rn == '1111' then SEE "LDRSH (literal)";
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant boolean postindex = FALSE;
constant boolean add = TRUE;
constant boolean register_form = FALSE;
constant bits(32) imm32 = ZeroExtend(imm8, 32);
constant integer m = integer UNKNOWN;
// Armv8-A removes UNPREDICTABLE for R13
if t == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

Is the general-purpose register to be transferred, encoded in the “Rt” field.

<Rn>

Is the general-purpose base register, encoded in the “Rn” field.

+/-

For the “A1” variant: specifies the offset is added to or subtracted from the base register, defaulting to + if omitted and encoded in “U”:

U	+/-
0	-
1	+

For the “A2” variant: specifies the index register is added to or subtracted from the base register, defaulting to + if omitted and encoded in “U”:

U	+/-
0	-
1	+

<imm>

For the “A1” variant: is the 8-bit unsigned immediate byte offset, in the range 0 to 255, defaulting to 0 if omitted, and encoded in the “imm4H:imm4L” field.

For the “T1” variant: is an optional 8-bit unsigned immediate byte offset, in the range 0 to 255, defaulting to 0 and encoded in the “imm8” field.

<Rm>

Is the general-purpose index register, encoded in the “Rm” field.

+

Specifies the offset is added to the base register.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    if PSTATE.EL == EL2 then UNPREDICTABLE;                    // Hyp mode
    constant bits(32) offset = if register_form then R[m] else imm32;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
constant bits(32) offset_addr = if add then (R[n] + offset) else (R[n] - offset);
constant bits(32) address = if postindex then R[n] else offset_addr;
constant bits(16) data = MemU_unpriv[address,2];
if postindex then R[n] = offset_addr;
R[t] = SignExtend(data, 32);
```

CONSTRAINED UNPREDICTABLE behavior

If `PSTATE.EL == EL2`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction executes as `LDRSH` (immediate).

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.99 LDRT

Load Register Unprivileged loads a word from memory, and writes it to a register. For information about memory accesses see Memory accesses.

The memory access is restricted as if the PE were running in User mode. This makes no difference if the PE is actually running in User mode.

`LDRT` is UNPREDICTABLE in Hyp mode.

The T32 instruction uses an offset addressing mode, that calculates the address used for the memory access from a base register value and an immediate offset, and leaves the base register unchanged.

The A32 instruction uses a post-indexed addressing mode, that uses a base register value as the address for the memory access, and calculates a new address from a base register value and an offset and writes it back to the base register. The offset can be an immediate value or an optionally-shifted register value.

It has encodings from the following instruction sets: A32 (A1 and A2) and T32 (T1).

A1

Encoding

```
LDRT{<c>}{<q>}  <Rt>, [<Rn>] {, #{+/-}<imm>}
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant boolean postindex = TRUE;
constant boolean add = (U == '1');
constant boolean register_form = FALSE;
constant bits(32) imm32 = ZeroExtend(imm12, 32);
constant integer m = integer UNKNOWN;
constant integer shift_n = integer UNKNOWN;
constant SRType shift_t = SRType UNKNOWN;
if t == 15 || n == 15 || n == t then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `n == 15`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction uses post-indexed addressing with the base register as PC. This is handled as described in Using R15.
- The instruction is treated as if bit[24] == ‘1’ and bit[21] == ‘0’. The instruction uses immediate offset addressing with the base register as PC, without writeback.

If `n == t && n != 15`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such as instruction, the base address might be corrupted so that the instruction cannot be repeated.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

A2

Encoding

```
LDRT{<c>}{<q>}  <Rt>, [<Rn>], {+/-}<Rm>{, <shift>}
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean postindex = TRUE;
constant boolean add = (U == '1');
constant boolean register_form = TRUE;
SRType shift_t;
integer shift_n;
(shift_t, shift_n) = DecodeImmShift(stype, imm5);
constant bits(32) imm32 = bits(32) UNKNOWN;
if t == 15 || n == 15 || n == t || m == 15 then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `n == t && n != 15`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The instruction performs all of the loads using the specified addressing mode and the content of the register that is written back is UNKNOWN. In addition, if an exception occurs during such as instruction, the base address might be corrupted so that the instruction cannot be repeated.

T1

Encoding

```
LDRT{<c>}{<q>}  <Rt>, [<Rn> {, #{+}<imm>}]
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for this encoding

```
if Rn == '1111' then SEE "LDR (literal)";
constant integer t = UInt(Rt);
constant integer n = UInt(Rn);
constant boolean postindex = FALSE;
constant boolean add = TRUE;
constant boolean register_form = FALSE;
constant bits(32) imm32 = ZeroExtend(imm8, 32);
constant integer m = integer UNKNOWN;
constant integer shift_n = integer UNKNOWN;
constant SRType shift_t = SRType UNKNOWN;
// Armv8-A removes UNPREDICTABLE for R13
if t == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rt>

For the “A1” variant: is the general-purpose register to be transferred, encoded in the “Rt” field. The PC can be used, but this is deprecated.

For the “A2” and “T1” variants: is the general-purpose register to be transferred, encoded in the “Rt” field.

<Rn>

Is the general-purpose base register, encoded in the “Rn” field.

+/-

For the “A1” variant: specifies the offset is added to or subtracted from the base register, defaulting to + if omitted and encoded in “U”:

U	+/-
0	-
1	+

For the “A2” variant: specifies the index register is added to or subtracted from the base register, defaulting to + if omitted and encoded in “U”:

U	+/-
0	-
1	+

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

<imm>

For the “A1” variant: is the 12-bit unsigned immediate byte offset, in the range 0 to 4095, defaulting to 0 if omitted, and encoded in the “imm12” field.

For the “T1” variant: is an optional 8-bit unsigned immediate byte offset, in the range 0 to 255, defaulting to 0 and encoded in the “imm8” field.

<Rm>

Is the general-purpose index register, encoded in the “Rm” field.

<shift>

The shift to apply to the value read from `<Rm>`. If absent, no shift is applied. Otherwise, see Shifts applied to a register.

+

Specifies the offset is added to the base register.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    if PSTATE.EL == EL2 then UNPREDICTABLE;                    // Hyp mode
    constant bits(32) offset = (if register_form then Shift(R[m], shift_t, shift_n, PSTATE.C)
                                else imm32);
    constant bits(32) offset_addr = if add then (R[n] + offset) else (R[n] - offset);
    constant bits(32) address = if postindex then R[n] else offset_addr;
    constant bits(32) data = MemU_unpriv[address,4];
    if postindex then R[n] = offset_addr;
    R[t] = data;
```

CONSTRAINED UNPREDICTABLE behavior

If `PSTATE.EL == EL2`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.
- The instruction executes as `LDR` (immediate).

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.100 LSL (immediate)

Logical Shift Left (immediate) shifts a register value left by an immediate number of bits, shifting in zeros, and writes the result to the destination register.

This is an alias of MOV, MOVS (register). This means:

- The encodings in this description are named to match the encodings of MOV, MOVS (register).
- The description of MOV, MOVS (register) gives the operational pseudocode, any CONSTRAINED UNPREDICTABLE behavior, and any operational information for this instruction.

It has encodings from the following instruction sets: A32 (A1) and T32 (T2 and T3).

A1

Encoding for the MOV, shift or rotate by value variant

```
LSL{<c>}{<q>}  {<Rd>, }<Rm>, #<imm>
```

is equivalent to

```
MOV{<c>}{<q>}  <Rd>, <Rm>, LSL #<imm>
```

and is always the preferred disassembly.

T2

Encoding for the T2 variant

```
LSL<c>{<q>}  {<Rd>, }<Rm>, #<imm>
```

is equivalent to

```
MOV<c>{<q>}  <Rd>, <Rm>, LSL #<imm>
```

and is the preferred disassembly when `InITBlock()`.

T3

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Encoding for the MOV, shift or rotate by value variant

Applies when `(!(imm3 == '000' && imm2 == '00'))`

```
LSL{<c>}{<q>}  {<Rd>, }<Rm>, #<imm>
```

```
LSL<c>.W  {<Rd>, }<Rm>, #<imm> // (!InITBlock() && UInt(Rd) < 8 && UInt(Rm) < 8)
```

is equivalent to

```
MOV{<c>}{<q>}  <Rd>, <Rm>, LSL #<imm>
```

and is always the preferred disassembly.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

For the “A1 MOV, shift or rotate by value” variant: is the general-purpose destination register, encoded in the “Rd” field. Arm deprecates using the PC as the destination register, but if the PC is used, the instruction is a branch to the address calculated by the operation. This is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.

For the “T2” and “T3 MOV, shift or rotate by value” variants: is the general-purpose destination register, encoded in the “Rd” field.

<Rm>

For the “A1 MOV, shift or rotate by value” variant: is the general-purpose source register, encoded in the “Rm” field. The PC can be used, but this is deprecated.

For the “T2” and “T3 MOV, shift or rotate by value” variants: is the general-purpose source register, encoded in the “Rm” field.

<imm>

For the “A1 MOV, shift or rotate by value” variant: is the shift amount, in the range 0 to 31, encoded in the “imm5” field as <imm> modulo 32.

For the “T2” variant: is the shift amount, in the range 1 to 31, encoded in the “imm5” field as <amount> modulo 32.

For the “T3 MOV, shift or rotate by value” variant: is the shift amount, in the range 0 to 31, encoded in the “imm3:imm2” field as <imm> modulo 32.

Operation

The description of MOV, MOVS (register) gives the operational pseudocode for this instruction.

Operational Information

The description of MOV, MOVS (register) gives the operational information for this instruction.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.101 LSL (register)

Logical Shift Left (register) shifts a register value left by a variable number of bits, shifting in zeros, and writes the result to the destination register. The variable number of bits is read from the bottom byte of a register.

This is an alias of MOV, MOVS (register-shifted register). This means:

- The encodings in this description are named to match the encodings of MOV, MOVS (register-shifted register).
- The description of MOV, MOVS (register-shifted register) gives the operational pseudocode, any CONSTRAINED UNPREDICTABLE behavior, and any operational information for this instruction.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1 and T2).

A1

Encoding for the Not flag setting variant

```
LSL{<c>}{<q>}  {<Rd>, }<Rm>, <Rs>
```

is equivalent to

```
MOV{<c>}{<q>}  <Rd>, <Rm>, LSL <Rs>
```

and is always the preferred disassembly.

T1

Encoding for the Logical shift left variant

```
LSL<c>{<q>}  {<Rdm>, }<Rdm>, <Rs>
```

is equivalent to

```
MOV<c>{<q>}  <Rdm>, <Rdm>, LSL <Rs>
```

and is the preferred disassembly when `InITBlock()`.

T2

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Encoding for the Not flag setting variant

```
LSL{<c>}{<q>}  {<Rd>, }<Rm>, <Rs>
```

```
LSL<c>.W  {<Rd>, }<Rm>, <Rs> // (InITBlock() && (UInt(Rd) < 8 && Rd == Rm && UInt(Rs) < 8))
```

is equivalent to

```
MOV{<c>}{<q>}  <Rd>, <Rm>, LSL <Rs>
```

and is always the preferred disassembly.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

Is the general-purpose destination register, encoded in the “Rd” field.

<Rm>

Is the first general-purpose source register, encoded in the “Rm” field.

<Rs>

Is the second general-purpose source register holding a shift amount in its bottom 8 bits, encoded in the “Rs” field.

<Rdm>

Is the first general-purpose source register and the destination register, encoded in the “Rdm” field.

Operation

The description of MOV, MOVS (register-shifted register) gives the operational pseudocode for this instruction.

Operational Information

The description of MOV, MOVS (register-shifted register) gives the operational information for this instruction.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.102 LSLS (immediate)

Logical Shift Left, setting flags (immediate) shifts a register value left by an immediate number of bits, shifting in zeros, and writes the result to the destination register.

If the destination register is not the PC, this instruction updates the condition flags based on the result.

The field descriptions for `<Rd>` identify the encodings where the PC is permitted as the destination register. Arm deprecates any use of these encodings. However, when the destination register is the PC:

- The PE branches to the address written to the PC, and restores PSTATE from SPSR_<current_mode>.
- The PE checks SPSR_<current_mode> for an illegal return event. See Illegal return events from AArch32 state.
- The instruction is UNDEFINED in Hyp mode.
- The instruction is CONSTRAINED UNPREDICTABLE in User mode and System mode.

This is an alias of MOV, MOVS (register). This means:

- The encodings in this description are named to match the encodings of MOV, MOVS (register).
- The description of MOV, MOVS (register) gives the operational pseudocode, any CONSTRAINED UNPREDICTABLE behavior, and any operational information for this instruction.

It has encodings from the following instruction sets: A32 (A1) and T32 (T2 and T3).

A1

Encoding for the MOVS, shift or rotate by value variant

```
LSLS{<c>}{<q>}  {<Rd>, }<Rm>, #<imm>
```

is equivalent to

```
MOVS{<c>}{<q>}  <Rd>, <Rm>, LSL #<imm>
```

and is always the preferred disassembly.

T2

Encoding for the T2 variant

```
LSLS{<q>}  {<Rd>, }<Rm>, #<imm>
```

is equivalent to

```
MOVS<c>{<q>}  <Rd>, <Rm>, LSL #<imm>
```

and is the preferred disassembly when `!InITBlock()`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T3

15–13	12–9	8–5	4	3–0	15	14–12	11–8	7–6	5–4	3–0
1 1 1	0 1 0 1	0 0 1 0	1	1 1 1 1	(0)	imm3	Rd	imm2	0 0	Rm
		op1	S	Rn					stype	

Encoding for the MOVS, shift or rotate by value variant

Applies when `(!(imm3 == '000' && imm2 == '00'))`

```
LSLS{<c>}{<q>}  {<Rd>, }<Rm>, #<imm>
```

```
LSLS.W  {<Rd>, }<Rm>, #<imm> // (!InITBlock() && UInt(Rd) < 8 && UInt(Rm) < 8)
```

is equivalent to

```
MOVS{<c>}{<q>}  <Rd>, <Rm>, LSL #<imm>
```

and is always the preferred disassembly.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

For the “A1 MOVS, shift or rotate by value” variant: is the general-purpose destination register, encoded in the “Rd” field. Arm deprecates using the PC as the destination register, but if the PC is used, the instruction performs an exception return, that restores PSTATE from SPSR_<current_mode>.

For the “T2” and “T3 MOVS, shift or rotate by value” variants: is the general-purpose destination register, encoded in the “Rd” field.

<Rm>

For the “A1 MOVS, shift or rotate by value” variant: is the general-purpose source register, encoded in the “Rm” field. The PC can be used, but this is deprecated.

For the “T2” and “T3 MOVS, shift or rotate by value” variants: is the general-purpose source register, encoded in the “Rm” field.

<imm>

For the “A1 MOVS, shift or rotate by value” variant: is the shift amount, in the range 0 to 31, encoded in the “imm5” field as <imm> modulo 32.

For the “T2” variant: is the shift amount, in the range 1 to 31, encoded in the “imm5” field as <amount> modulo 32.

For the “T3 MOVS, shift or rotate by value” variant: is the shift amount, in the range 0 to 31, encoded in the “imm3:imm2” field as <imm> modulo 32.

Operation

The description of MOV, MOVS (register) gives the operational pseudocode for this instruction.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Operational Information

The description of MOV, MOVS (register) gives the operational information for this instruction.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.103 LSLS (register)

Logical Shift Left, setting flags (register) shifts a register value left by a variable number of bits, shifting in zeros, writes the result to the destination register, and updates the condition flags based on the result. The variable number of bits is read from the bottom byte of a register.

This is an alias of MOV, MOVS (register-shifted register). This means:

- The encodings in this description are named to match the encodings of MOV, MOVS (register-shifted register).
- The description of MOV, MOVS (register-shifted register) gives the operational pseudocode, any CONSTRAINED UNPREDICTABLE behavior, and any operational information for this instruction.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1 and T2).

A1

Encoding for the Flag setting variant

```
LSLS{<c>}{<q>}  {<Rd>, }<Rm>, <Rs>
```

is equivalent to

```
MOVS{<c>}{<q>}  <Rd>, <Rm>, LSL <Rs>
```

and is always the preferred disassembly.

T1

Encoding for the Logical shift left variant

```
LSLS{<q>}  {<Rdm>, }<Rdm>, <Rs>
```

is equivalent to

```
MOVS{<q>}  <Rdm>, <Rdm>, LSL <Rs>
```

and is the preferred disassembly when `!InITBlock()`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T2

15–13	12–8	7	6–5	4	3–0	15–12	11–8	7–4	3–0
1 1 1	1 1 0 1 0	0	0 0	1	Rm	1 1 1 1	Rd	0 0 0 0	Rs
			stype	S					

Encoding for the Flag setting variant

```
LSLS{<c>}{<q>}  {<Rd>, }<Rm>, <Rs>
```

```
LSLS.W  {<Rd>, }<Rm>, <Rs> // (!InITBlock() && (UInt(Rd) < 8 && Rd == Rm && UInt(Rs) < 8))
```

is equivalent to

```
MOVS{<c>}{<q>}  <Rd>, <Rm>, LSL <Rs>
```

and is always the preferred disassembly.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

Is the general-purpose destination register, encoded in the “Rd” field.

<Rm>

Is the first general-purpose source register, encoded in the “Rm” field.

<Rs>

Is the second general-purpose source register holding a shift amount in its bottom 8 bits, encoded in the “Rs” field.

<Rdm>

Is the first general-purpose source register and the destination register, encoded in the “Rdm” field.

Operation

The description of MOV, MOVS (register-shifted register) gives the operational pseudocode for this instruction.

Operational Information

The description of MOV, MOVS (register-shifted register) gives the operational information for this instruction.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.104 LSR (immediate)

Logical Shift Right (immediate) shifts a register value right by an immediate number of bits, shifting in zeros, and writes the result to the destination register.

This is an alias of MOV, MOVS (register). This means:

- The encodings in this description are named to match the encodings of MOV, MOVS (register).
- The description of MOV, MOVS (register) gives the operational pseudocode, any CONSTRAINED UNPREDICTABLE behavior, and any operational information for this instruction.

It has encodings from the following instruction sets: A32 (A1) and T32 (T2 and T3).

A1

Encoding for the MOV, shift or rotate by value variant

```
LSR{<c>}{<q>}  {<Rd>, }<Rm>, #<imm>
```

is equivalent to

```
MOV{<c>}{<q>}  <Rd>, <Rm>, LSR #<imm>
```

and is always the preferred disassembly.

T2

Encoding for the T2 variant

```
LSR<c>{<q>}  {<Rd>, }<Rm>, #<imm>
```

is equivalent to

```
MOV<c>{<q>}  <Rd>, <Rm>, LSR #<imm>
```

and is the preferred disassembly when `InITBlock()`.

T3

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Encoding for the MOV, shift or rotate by value variant

```
LSR{<c>}{<q>}  {<Rd>, }<Rm>, #<imm>
```

```
LSR<c>.W  {<Rd>, }<Rm>, #<imm> // (!InITBlock() && UInt(Rd) < 8 && UInt(Rm) < 8)
```

is equivalent to

```
MOV{<c>}{<q>}  <Rd>, <Rm>, LSR #<imm>
```

and is always the preferred disassembly.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

For the "A1 MOV, shift or rotate by value" variant: is the general-purpose destination register, encoded in the "Rd" field. Arm deprecates using the PC as the destination register, but if the PC is used, the instruction is a branch to the address calculated by the operation. This is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.

For the "T2" and "T3 MOV, shift or rotate by value" variants: is the general-purpose destination register, encoded in the "Rd" field.

<Rm>

For the "A1 MOV, shift or rotate by value" variant: is the general-purpose source register, encoded in the "Rm" field. The PC can be used, but this is deprecated.

For the "T2" and "T3 MOV, shift or rotate by value" variants: is the general-purpose source register, encoded in the "Rm" field.

<imm>

For the "A1 MOV, shift or rotate by value" and "T2" variants: is the shift amount, in the range 1 to 32, encoded in the "imm5" field as <imm> modulo 32.

For the "T3 MOV, shift or rotate by value" variant: is the shift amount, in the range 1 to 32, encoded in the "imm3:imm2" field as <imm> modulo 32.

Operation

The description of MOV, MOVS (register) gives the operational pseudocode for this instruction.

Operational Information

The description of MOV, MOVS (register) gives the operational information for this instruction.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.105 LSR (register)

Logical Shift Right (register) shifts a register value right by a variable number of bits, shifting in zeros, and writes the result to the destination register. The variable number of bits is read from the bottom byte of a register.

This is an alias of MOV, MOVS (register-shifted register). This means:

- The encodings in this description are named to match the encodings of MOV, MOVS (register-shifted register).
- The description of MOV, MOVS (register-shifted register) gives the operational pseudocode, any CONSTRAINED UNPREDICTABLE behavior, and any operational information for this instruction.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1 and T2).

A1

Encoding for the Not flag setting variant

```
LSR{<c>}{<q>}  {<Rd>, }<Rm>, <Rs>
```

is equivalent to

```
MOV{<c>}{<q>}  <Rd>, <Rm>, LSR <Rs>
```

and is always the preferred disassembly.

T1

Encoding for the Logical shift right variant

```
LSR<c>{<q>}  {<Rdm>, }<Rdm>, <Rs>
```

is equivalent to

```
MOV<c>{<q>}  <Rdm>, <Rdm>, LSR <Rs>
```

and is the preferred disassembly when `InITBlock()`.

T2

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Encoding for the Not flag setting variant

```
LSR{<c>}{<q>}  {<Rd>, }<Rm>, <Rs>
```

```
LSR<c>.W  {<Rd>, }<Rm>, <Rs> // (InITBlock() && (UInt(Rd) < 8 && Rd == Rm && UInt(Rs) < 8))
```

is equivalent to

```
MOV{<c>}{<q>}  <Rd>, <Rm>, LSR <Rs>
```

and is always the preferred disassembly.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

Is the general-purpose destination register, encoded in the “Rd” field.

<Rm>

Is the first general-purpose source register, encoded in the “Rm” field.

<Rs>

Is the second general-purpose source register holding a shift amount in its bottom 8 bits, encoded in the “Rs” field.

<Rdm>

Is the first general-purpose source register and the destination register, encoded in the “Rdm” field.

Operation

The description of MOV, MOVS (register-shifted register) gives the operational pseudocode for this instruction.

Operational Information

The description of MOV, MOVS (register-shifted register) gives the operational information for this instruction.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.106 LSRS (immediate)

Logical Shift Right, setting flags (immediate) shifts a register value right by an immediate number of bits, shifting in zeros, and writes the result to the destination register.

If the destination register is not the PC, this instruction updates the condition flags based on the result.

The field descriptions for `<Rd>` identify the encodings where the PC is permitted as the destination register. Arm deprecates any use of these encodings. However, when the destination register is the PC:

- The PE branches to the address written to the PC, and restores PSTATE from SPSR_<current_mode>.
- The PE checks SPSR_<current_mode> for an illegal return event. See Illegal return events from AArch32 state.
- The instruction is UNDEFINED in Hyp mode.
- The instruction is CONSTRAINED UNPREDICTABLE in User mode and System mode.

This is an alias of MOV, MOVS (register). This means:

- The encodings in this description are named to match the encodings of MOV, MOVS (register).
- The description of MOV, MOVS (register) gives the operational pseudocode, any CONSTRAINED UNPREDICTABLE behavior, and any operational information for this instruction.

It has encodings from the following instruction sets: A32 (A1) and T32 (T2 and T3).

A1

Encoding for the MOVS, shift or rotate by value variant

```
LSRS{<c>}{<q>}  {<Rd>, }<Rm>, #<imm>
```

is equivalent to

```
MOVS{<c>}{<q>}  <Rd>, <Rm>, LSR #<imm>
```

and is always the preferred disassembly.

T2

Encoding for the T2 variant

```
LSRS{<q>}  {<Rd>, }<Rm>, #<imm>
```

is equivalent to

```
MOVS<c>{<q>}  <Rd>, <Rm>, LSR #<imm>
```

and is the preferred disassembly when `!InITBlock()`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T3

15–13	12–9	8–5	4	3–0	15	14–12	11–8	7–6	5–4	3–0
1 1 1	0 1 0 1	0 0 1 0	1	1 1 1 1	(0)	imm3	Rd	imm2	0 1	Rm
		op1	S	Rn					stype	

Encoding for the MOVS, shift or rotate by value variant

```
LSRS{<c>}{<q>}  {<Rd>, }<Rm>, #<imm>
```

```
LSRS.W  {<Rd>, }<Rm>, #<imm> // (!InITBlock() && UInt(Rd) < 8 && UInt(Rm) < 8)
```

is equivalent to

```
MOVS{<c>}{<q>}  <Rd>, <Rm>, LSR #<imm>
```

and is always the preferred disassembly.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

For the “A1 MOVS, shift or rotate by value” variant: is the general-purpose destination register, encoded in the “Rd” field. Arm deprecates using the PC as the destination register, but if the PC is used, the instruction performs an exception return, that restores PSTATE from SPSR_<current_mode>.

For the “T2” and “T3 MOVS, shift or rotate by value” variants: is the general-purpose destination register, encoded in the “Rd” field.

<Rm>

For the “A1 MOVS, shift or rotate by value” variant: is the general-purpose source register, encoded in the “Rm” field. The PC can be used, but this is deprecated.

For the “T2” and “T3 MOVS, shift or rotate by value” variants: is the general-purpose source register, encoded in the “Rm” field.

<imm>

For the “A1 MOVS, shift or rotate by value” and “T2” variants: is the shift amount, in the range 1 to 32, encoded in the “imm5” field as <imm> modulo 32.

For the “T3 MOVS, shift or rotate by value” variant: is the shift amount, in the range 1 to 32, encoded in the “imm3:imm2” field as <imm> modulo 32.

Operation

The description of MOV, MOVS (register) gives the operational pseudocode for this instruction.

Operational Information

The description of MOV, MOVS (register) gives the operational information for this instruction.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.107 LSRS (register)

Logical Shift Right, setting flags (register) shifts a register value right by an immediate number of bits, shifting in zeros, writes the result to the destination register, and updates the condition flags based on the result. The variable number of bits is read from the bottom byte of a register.

This is an alias of MOV, MOVS (register-shifted register). This means:

- The encodings in this description are named to match the encodings of MOV, MOVS (register-shifted register).
- The description of MOV, MOVS (register-shifted register) gives the operational pseudocode, any CONSTRAINED UNPREDICTABLE behavior, and any operational information for this instruction.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1 and T2).

A1

Encoding for the Flag setting variant

```
LSRS{<c>}{<q>}  {<Rd>, }<Rm>, <Rs>
```

is equivalent to

```
MOVS{<c>}{<q>}  <Rd>, <Rm>, LSR <Rs>
```

and is always the preferred disassembly.

T1

Encoding for the Logical shift right variant

```
LSRS{<q>}  {<Rdm>, }<Rdm>, <Rs>
```

is equivalent to

```
MOVS{<q>}  <Rdm>, <Rdm>, LSR <Rs>
```

and is the preferred disassembly when `!InITBlock()`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T2

15–13	12–8	7	6–5	4	3–0	15–12	11–8	7–4	3–0
1 1 1	1 1 0 1 0	0	0 1	1	Rm	1 1 1 1	Rd	0 0 0 0	Rs
			stype	S					

Encoding for the Flag setting variant

```
LSRS{<c>}{<q>}  {<Rd>, }<Rm>, <Rs>
```

```
LSRS.W  {<Rd>, }<Rm>, <Rs> // (!InITBlock() && (UInt(Rd) < 8 && Rd == Rm && UInt(Rs) < 8))
```

is equivalent to

```
MOVS{<c>}{<q>}  <Rd>, <Rm>, LSR <Rs>
```

and is always the preferred disassembly.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

Is the general-purpose destination register, encoded in the “Rd” field.

<Rm>

Is the first general-purpose source register, encoded in the “Rm” field.

<Rs>

Is the second general-purpose source register holding a shift amount in its bottom 8 bits, encoded in the “Rs” field.

<Rdm>

Is the first general-purpose source register and the destination register, encoded in the “Rdm” field.

Operation

The description of MOV, MOVS (register-shifted register) gives the operational pseudocode for this instruction.

Operational Information

The description of MOV, MOVS (register-shifted register) gives the operational information for this instruction.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.108 MCR

Move to System register from general-purpose register or execute a System instruction. This instruction copies the value of a general-purpose register to a System register, or executes a System instruction.

The System register and System instruction descriptions identify valid encodings for this instruction. Other encodings are UNDEFINED. For more information see About the AArch32 System register interface and General behavior of System registers.

In an implementation that includes EL2, MCR accesses to System registers can be trapped to Hyp mode, meaning that an attempt to execute an MCR instruction in a Non-secure mode other than Hyp mode, that would be permitted in the absence of the Hyp trap controls, generates a Hyp Trap exception. For more information, see EL2 configurable instruction enables, disables, and traps.

Because of the range of possible traps to Hyp mode, the MCR pseudocode does not show these possible traps.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

31–28	27 26	25 24	23–21	20	19–16	15–12	11	10 9	8	7–5	4	3–0
!=1111	1 1	1 0	opc1	0	CRn	Rt	1	1 1		opc2	1	CRm
cond				L					cp15			

Encoding

```
MCR{<c>}{<q>}  <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer cp = if cp15 == '0' then 14 else 15;
// Armv8-A removes UNPREDICTABLE for R13
if t == 15  then UNPREDICTABLE;
```

T1

15–13	12	11 10	9 8	7–5	4	3–0	15–12	11	10 9	8	7–5	4	3–0
1 1 1	0	1 1	1 0	opc1	0	CRn	Rt	1	1 1		opc2	1	CRm
					L					cp15			

Encoding

```
MCR{<c>}{<q>}  <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer cp = if cp15 == '0' then 14 else 15;
// Armv8-A removes UNPREDICTABLE for R13
if t == 15  then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<coproc>

Is the System register encoding space, encoded in “cp15”:

cp15	<coproc>
0	p14
1	p15

<opc1>

Is the opc1 parameter within the System register encoding space, in the range 0 to 7, encoded in the “opc1” field.

<Rt>

Is the general-purpose register to be transferred, encoded in the “Rt” field.

<CRn>

Is the CRn parameter within the System register encoding space, in the range c0 to c15, encoded in the “CRn” field.

<CRm>

Is the CRm parameter within the System register encoding space, in the range c0 to c15, encoded in the “CRm” field.

<opc2>

Is the opc2 parameter within the System register encoding space, in the range 0 to 7, encoded in the “opc2” field.

The possible values of { `<coproc>`, `<opc1>`, `<CRn>`, `<CRm>`, `<opc2>` } encode the entire System register and System instruction encoding space. Not all of this space is allocated, and the System register and System instruction descriptions identify the allocated encodings.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    AArch32.SysRegWrite(cp, ThisInstr(), t);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.109 MCRR

Move to System register from two general-purpose registers. This instruction copies the values of two general-purpose registers to a System register.

The System register descriptions identify valid encodings for this instruction. Other encodings are UNDEFINED. For more information see About the AArch32 System register interface and General behavior of System registers.

In an implementation that includes EL2, MCRR accesses to System registers can be trapped to Hyp mode, meaning that an attempt to execute an MCRR instruction in a Non-secure mode other than Hyp mode, that would be permitted in the absence of the Hyp trap controls, generates a Hyp Trap exception. For more information, see EL2 configurable instruction enables, disables, and traps.

Because of the range of possible traps to Hyp mode, the MCRR pseudocode does not show these possible traps.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding

```
MCRR{<c>}{<q>}  <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer t2 = UInt(Rt2);
constant integer cp = if cp15 == '0' then 14 else 15;
if t == 15 || t2 == 15 then UNPREDICTABLE;
// Armv8-A removes UNPREDICTABLE for R13
```

T1

Encoding

```
MCRR{<c>}{<q>}  <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer t2 = UInt(Rt2);
constant integer cp = if cp15 == '0' then 14 else 15;
if t == 15 || t2 == 15 then UNPREDICTABLE;
// Armv8-A removes UNPREDICTABLE for R13
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<coproc>

Is the System register encoding space, encoded in “cp15”:

cp15	<coproc>
0	p14
1	p15

<opc1>

Is the opc1 parameter within the System register encoding space, in the range 0 to 15, encoded in the “opc1” field.

<Rt>

Is the first general-purpose register that is transferred into, encoded in the “Rt” field.

<Rt2>

Is the second general-purpose register that is transferred into, encoded in the “Rt2” field.

<CRm>

Is the CRm parameter within the System register encoding space, in the range c0 to c15, encoded in the “CRm” field.

The possible values of { `<coproc>`, `<opc1>`, `<CRm>` } encode the entire System register encoding space. Not all of this space is allocated, and the System register descriptions identify the allocated encodings.

For the permitted uses of these instructions, as described in this manual, `<Rt2>` transfers bits[63:32] of the selected System register, while `<Rt>` transfers bits[31:0].

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    AArch32.SysRegWrite64(cp, ThisInstr(), t, t2);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.110 MLA, MLAS

Multiply Accumulate multiplies two register values, and adds a third register value. The least significant 32 bits of the result are written to the destination register. These 32 bits do not depend on whether the source register values are considered to be signed values or unsigned values.

In an A32 instruction, the condition flags can optionally be updated based on the result. Use of this option adversely affects performance on many implementations.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding for the Flag setting variant

Applies when `(S == 1)`

```
MLAS{<c>}{<q>}  <Rd>, <Rn>, <Rm>, <Ra>
```

Encoding for the Not flag setting variant

Applies when `(S == 0)`

```
MLA{<c>}{<q>}  <Rd>, <Rn>, <Rm>, <Ra>
```

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant integer a = UInt(Ra);
constant boolean setflags = (S == '1');
if d == 15 || n == 15 || m == 15 || a == 15 then UNPREDICTABLE;
```

T1

Encoding

```
MLA{<c>}{<q>}  <Rd>, <Rn>, <Rm>, <Ra>
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for this encoding

```
if Ra == '1111' then SEE "MUL";
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant integer a = UInt(Ra);
constant boolean setflags = FALSE;
// Armv8-A removes UNPREDICTABLE for R13
if d == 15 || n == 15 || m  == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

Is the general-purpose destination register, encoded in the “Rd” field.

<Rn>

Is the first general-purpose source register holding the multiplicand, encoded in the “Rn” field.

<Rm>

Is the second general-purpose source register holding the multiplier, encoded in the “Rm” field.

<Ra>

Is the third general-purpose source register holding the addend, encoded in the “Ra” field.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant integer operand1 = SInt(R[n]); // operand1 = UInt(R[n]) produces the same final results
    constant integer operand2 = SInt(R[m]); // operand2 = UInt(R[m]) produces the same final results
    constant integer addend   = SInt(R[a]); // addend   = UInt(R[a]) produces the same final results
    constant integer result = operand1 * operand2 + addend;
    R[d] = result<31:0>;
    if setflags then
        PSTATE.N = result<31>;
        PSTATE.Z = IsZeroBit(result<31:0>);
        // PSTATE.C, PSTATE.V unchanged
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.111 MLS

Multiply and Subtract multiplies two register values, and subtracts the product from a third register value. The least significant 32 bits of the result are written to the destination register. These 32 bits do not depend on whether the source register values are considered to be signed values or unsigned values.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

31–28	27	26	25	24	23	22	21	20	19–16	15–12	11–8	7	6	5	4	3–0
!=1111 (cond)	0	0	0	0	0	1	1	0 (S)	Rd	Ra	Rm	1	0	0	1	Rn

opc

Encoding

```
MLS{<c>}{<q>}  <Rd>, <Rn>, <Rm>, <Ra>
```

Decode for this encoding

```
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant integer a = UInt(Ra);
if d == 15 || n == 15 || m == 15 || a == 15 then UNPREDICTABLE;
```

T1

15–13	12–7	6–4	3–0	15–12	11–8	7–6	5–4	3–0
1 1 1	1 1 0 1 1 0	0 0 0 (op1)	Rn	Ra	Rd	0 0	0 1 (op2)	Rm

Encoding

```
MLS{<c>}{<q>}  <Rd>, <Rn>, <Rm>, <Ra>
```

Decode for this encoding

```
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant integer a = UInt(Ra);
if d == 15 || n == 15 || m == 15 || a == 15 then UNPREDICTABLE;
// Armv8-A removes UNPREDICTABLE for R13
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

Is the general-purpose destination register, encoded in the “Rd” field.

<Rn>

Is the first general-purpose source register holding the multiplicand, encoded in the “Rn” field.

<Rm>

Is the second general-purpose source register holding the multiplier, encoded in the “Rm” field.

<Ra>

Is the third general-purpose source register holding the minuend, encoded in the “Ra” field.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant integer operand1 = SInt(R[n]); // operand1 = UInt(R[n]) produces the same final results
    constant integer operand2 = SInt(R[m]); // operand2 = UInt(R[m]) produces the same final results
    constant integer addend   = SInt(R[a]); // addend   = UInt(R[a]) produces the same final results
    constant integer result = addend - operand1 * operand2;
    R[d] = result<31:0>;
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.112 MOV, MOVS (immediate)

Move (immediate) writes an immediate value to the destination register.

If the destination register is not the PC, the MOVS variant of the instruction updates the condition flags based on the result.

The field descriptions for `<Rd>` identify the encodings where the PC is permitted as the destination register. Arm deprecates any use of these encodings. However, when the destination register is the PC:

- The MOV variant of the instruction is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- The MOVS variant of the instruction performs an exception return without the use of the stack. In this case:
  - The PE branches to the address written to the PC, and restores PSTATE from SPSR_<current_mode>.
  - The PE checks SPSR_<current_mode> for an illegal return event. See Illegal return events from AArch32 state.
  - The instruction is UNDEFINED in Hyp mode.
  - The instruction is CONSTRAINED UNPREDICTABLE in User mode and System mode.

It has encodings from the following instruction sets: A32 (A1 and A2) and T32 (T1, T2, and T3).

A1

Encoding for the MOV variant

Applies when `(S == 0)`

```
MOV{<c>}{<q>}  <Rd>, #<const>
```

Encoding for the MOVS variant

Applies when `(S == 1)`

```
MOVS{<c>}{<q>}  <Rd>, #<const>
```

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant boolean setflags = (S == '1');
bits(32) imm32;
bit carry;
(imm32, carry) = A32ExpandImm_C(imm12, PSTATE.C);
```

A2

Encoding

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MOV{<c>}{<q>}  <Rd>, #<imm16> // (<imm16> can not be represented in A1)
```

```
MOVW{<c>}{<q>}  <Rd>, #<imm16> // (<imm16> can be represented in A1)
```

Decode for this encoding

```
constant integer d = UInt(Rd);
constant boolean setflags = FALSE;
constant bits(32) imm32 = ZeroExtend(imm4:imm12, 32);
constant bit carry = bit UNKNOWN;
if d == 15 then UNPREDICTABLE;
```

T1

Encoding

```
MOV<c>{<q>}  <Rd>, #<imm8> // (InITBlock())
```

```
MOVS{<q>}  <Rd>, #<imm8> // (Outside IT block)
```

Decode for this encoding

```
constant integer d = UInt(Rd);
constant boolean setflags = !InITBlock();
constant bits(32) imm32 = ZeroExtend(imm8, 32);
constant bit carry = PSTATE.C;
```

T2

Encoding for the MOV variant

Applies when `(S == 0)`

```
MOV{<c>}{<q>}  <Rd>, #<const>
```

```
MOV<c>.W  <Rd>, #<const> // (Inside IT block, and <Rd>, <const> can be represented in T1)
```

Encoding for the MOVS variant

Applies when `(S == 1)`

```
MOVS{<c>}{<q>}  <Rd>, #<const>
```

```
MOVS.W  <Rd>, #<const> // (Outside IT block, and <Rd>, <const> can be represented in T1)
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant boolean setflags = (S == '1');
bits(32) imm32;
bit carry;
(imm32, carry) = T32ExpandImm_C(i:imm3:imm8, PSTATE.C);
// Armv8-A removes UNPREDICTABLE for R13
if d == 15 then UNPREDICTABLE;
```

T3

15–11	10	9–8	7	6–4	3–0	15	14–12	11–8	7–0
1 1 1 1 0	i	1 0	0	1 0 0	imm4	0	imm3	Rd	imm8
			o1						

Encoding

```
MOV{<c>}{<q>}  <Rd>, #<imm16> // (<imm16> cannot be represented in T1 or T2)
```

```
MOVW{<c>}{<q>}  <Rd>, #<imm16> // (<imm16> can be represented in T1 or T2)
```

Decode for this encoding

```
constant integer d = UInt(Rd);
constant boolean setflags = FALSE;
constant bits(32) imm32 = ZeroExtend(imm4:i:imm3:imm8, 32);
constant bit carry = bit UNKNOWN;
// Armv8-A removes UNPREDICTABLE for R13
if d == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

For the “A1 MOV” and “A1 MOVS” variants: is the general-purpose destination register, encoded in the “Rd” field. Arm deprecates using the PC as the destination register, but if the PC is used:

- For the MOV variant, the instruction is a branch to the address calculated by the operation. This is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- For the MOVS variant, the instruction performs an exception return, that restores PSTATE from SPSR_<current_mode>.

For the “A2”, “T1”, “T2 MOV”, “T2 MOVS”, and “T3” variants: is the general-purpose destination register, encoded in the “Rd” field.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

<const>

For the “A1 MOV” and “A1 MOVS” variants: an immediate value. See Modified immediate constants in A32 instructions for the range of values.

For the “T2 MOV” and “T2 MOVS” variants: an immediate value. See Modified immediate constants in T32 instructions for the range of values.

<imm16>

For the “A2” variant: is a 16-bit unsigned immediate, in the range 0 to 65535, encoded in the “imm4:imm12” field.

For the “T3” variant: is a 16-bit unsigned immediate, in the range 0 to 65535, encoded in the “imm4:i:imm3:imm8” field.

<imm8>

Is a 8-bit unsigned immediate, in the range 0 to 255, encoded in the “imm8” field.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) result = imm32;
    if d == 15 then          // Can only occur for encoding A1
        if setflags then
            ALUExceptionReturn(result);
        else
            ALUWritePC(result);
    else
        R[d] = result;
        if setflags then
            PSTATE.N = result<31>;
            PSTATE.Z = IsZeroBit(result);
            PSTATE.C = carry;
            // PSTATE.V unchanged
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.113 MOV, MOVS (register)

Move (register) copies a value from a register to the destination register.

If the destination register is not the PC, the MOVS variant of the instruction updates the condition flags based on the result.

The field descriptions for `<Rd>` identify the encodings where the PC is permitted as the destination register. If the destination register is the PC:

- The MOV variant of the instruction is a branch. In the T32 instruction set (encoding T1) this is a simple branch, and in the A32 instruction set it is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- The MOVS variant of the instruction performs an exception return without the use of the stack. In this case:
  - The PE branches to the address written to the PC, and restores PSTATE from SPSR_<current_mode>.
  - The PE checks SPSR_<current_mode> for an illegal return event. See Illegal return events from AArch32 state.
  - The instruction is UNDEFINED in Hyp mode.
  - The instruction is CONSTRAINED UNPREDICTABLE in User mode and System mode.

This instruction is used by the aliases ASRS (immediate), ASR (immediate), LSLS (immediate), LSL (immediate), LSRS (immediate), LSR (immediate), RORS (immediate), ROR (immediate), RRXS, and RRX.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1, T2, and T3).

A1

31–28	27–23	22–21	20	19–16	15–12	11–7	6–5	4	3–0
!=1111	0 0 0 1 1	0 1	S	(0)(0)(0)(0)	Rd	imm5	stype	0	Rm
cond		opc		Rn					

Encoding for the MOV, rotate right with extend variant

Applies when `(S == 0 && imm5 == 00000 && stype == 11)`

```
MOV{<c>}{<q>}  <Rd>, <Rm>, RRX
```

Encoding for the MOV, shift or rotate by value variant

Applies when `(S == 0 && !(imm5 == 00000 && stype == 11))`

```
MOV{<c>}{<q>}  <Rd>, <Rm> {, <shift> #<amount>}
```

Encoding for the MOVS, rotate right with extend variant

Applies when `(S == 1 && imm5 == 00000 && stype == 11)`

```
MOVS{<c>}{<q>}  <Rd>, <Rm>, RRX
```

Encoding for the MOVS, shift or rotate by value variant

Applies when `(S == 1 && !(imm5 == 00000 && stype == 11))`

```
MOVS{<c>}{<q>}  <Rd>, <Rm> {, <shift> #<amount>}
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant integer m = UInt(Rm);
constant boolean setflags = (S == '1');
SRType shift_t;
integer shift_n;
(shift_t, shift_n) = DecodeImmShift(stype, imm5);
```

T1

15					10	9	8	7	6			3	2		0
0	1	0	0	0	1	1	0	D	Rm				Rd		

op

Encoding

```
MOV{<c>}{<q>}  <Rd>, <Rm>
```

Decode for this encoding

```
constant integer d = UInt(D:Rd);
constant integer m = UInt(Rm);
constant boolean setflags = FALSE;
constant SRType shift_t = SRType_LSL;
constant integer shift_n = 0;
if d == 15 && InITBlock() && !LastInITBlock() then UNPREDICTABLE;
```

T2

15		13	12	11	10			6	5		3	2		0
0	0	0	!=11		imm5				Rm			Rd		

op

Encoding

```
MOV<c>{<q>}  <Rd>, <Rm> {, <shift> #<amount>} // (InITBlock())
```

```
MOVS{<q>}  <Rd>, <Rm> {, <shift> #<amount>} // (Outside IT block)
```

Decode for this encoding

```
constant integer d = UInt(Rd);
constant integer m = UInt(Rm);
constant boolean setflags = !InITBlock();
SRType shift_t;
integer shift_n;
(shift_t, shift_n) = DecodeImmShift(op, imm5);
if op == '00' && imm5 == '00000' && InITBlock() then UNPREDICTABLE;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CONSTRAINED UNPREDICTABLE behavior

If `op == '00' && imm5 == '00000' && InITBlock()`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as if it passed its condition code check.
- The instruction executes as NOP, as if it failed its condition code check.
- The instruction executes as MOV Rd, Rm.

T3

15–9	8–5	4	3–0	15	14–12	11–8	7–6	5–4	3–0
1 1 1 0 1 0 1	0 0 1 0 (op1)	S	1 1 1 1 (Rn)	(0)	imm3	Rd	imm2	stype	Rm

Encoding for the MOV, rotate right with extend variant

Applies when `(S == 0 && imm3 == 000 && imm2 == 00 && stype == 11)`

```
MOV{<c>}{<q>}  <Rd>, <Rm>, RRX
```

Encoding for the MOV, shift or rotate by value variant

Applies when `(S == 0 && !(imm3 == 000 && imm2 == 00 && stype == 11))`

```
MOV{<c>}{<q>}  <Rd>, <Rm> {, <shift> #<amount>}
```

```
MOV{<c>}.W  <Rd>, <Rm> {, LSL #0} // (<Rd>, <Rm> can be represented in T1)
```

```
MOV<c>.W  <Rd>, <Rm> {, <shift> #<amount>} // (Inside IT block, and <Rd>, <Rm>, <shift>, <amount> can be
    ↪represented in T2)
```

Encoding for the MOVS, rotate right with extend variant

Applies when `(S == 1 && imm3 == 000 && imm2 == 00 && stype == 11)`

```
MOVS{<c>}{<q>}  <Rd>, <Rm>, RRX
```

Encoding for the MOVS, shift or rotate by value variant

Applies when `(S == 1 && !(imm3 == 000 && imm2 == 00 && stype == 11))`

```
MOVS{<c>}{<q>}  <Rd>, <Rm> {, <shift> #<amount>}
```

```
MOVS.W  <Rd>, <Rm> {, <shift> #<amount>} // (Outside IT block, and <Rd>, <Rm>, <shift>, <amount> can be
    ↪represented in T1 or T2)
```

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant integer m = UInt(Rm);
constant boolean setflags = (S == '1');
SRType shift_t;
integer shift_n;
(shift_t, shift_n) = DecodeImmShift(stype, imm3:imm2);
// Armv8-A removes UNPREDICTABLE for R13
if d == 15 || m == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

For the "A1 MOV, rotate right with extend", "A1 MOV, shift or rotate by value", "A1 MOVS, rotate right with extend", and "A1 MOVS, shift or rotate by value" variants: is the general-purpose destination register, encoded in the "Rd" field. If the PC is used:

- For the MOV variant, the instruction is a branch to the address calculated by the operation. This is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC. Arm deprecates use of the instruction if `<Rn>` is the PC.
- For the MOVS variant, the instruction performs an exception return, that restores PSTATE from SPSR_<current_mode>. Arm deprecates use of the instruction if <Rn> is not the LR, or if the optional shift or `RRX` argument is specified.

For the "T1" variant: is the general-purpose destination register, encoded in the "D:Rd" field. If the PC is used:

- The instruction causes a branch to the address moved to the PC. This is a simple branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- The instruction must either be outside an IT block or the last instruction of an IT block.

For the "T2", "T3 MOV, rotate right with extend", "T3 MOV, shift or rotate by value", "T3 MOVS, rotate right with extend", and "T3 MOVS, shift or rotate by value" variants: is the general-purpose destination register, encoded in the "Rd" field.

<Rm>

For the "A1 MOV, rotate right with extend", "A1 MOV, shift or rotate by value", "A1 MOVS, rotate right with extend", "A1 MOVS, shift or rotate by value", and "T1" variants: is the general-purpose source register, encoded in the "Rm" field. The PC can be used. Arm deprecates use of the instruction if `<Rd>` is the PC.

For the "T2", "T3 MOV, rotate right with extend", "T3 MOV, shift or rotate by value", "T3 MOVS, rotate right with extend", and "T3 MOVS, shift or rotate by value" variants: is the general-purpose source register, encoded in the "Rm" field.

<shift>

For the "A1 MOV, shift or rotate by value", "A1 MOVS, shift or rotate by value", "T3 MOV, shift or rotate by value", and "T3 MOVS, shift or rotate by value" variants: is the type of shift to be applied to the source register, encoded in "stype":

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

For the "T2" variant: is the type of shift to be applied to the source register, encoded in "op":

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

op	<shift>
00	LSL
01	LSR
10	ASR

<amount>

For the “A1 MOV, shift or rotate by value” and “A1 MOVS, shift or rotate by value” variants: is the shift amount, in the range 0 to 31 (when <shift> = LSL), or 1 to 31 (when <shift> = ROR) or 1 to 32 (when <shift> = LSR or ASR), encoded in the “imm5” field as <amount> modulo 32.

For the “T2” variant: is the shift amount, in the range 1 to 31 (when <shift> = LSL or ROR) or 1 to 32 (when <shift> = LSR or ASR), encoded in the “imm5” field as <amount> modulo 32.

For the “T3 MOV, shift or rotate by value” and “T3 MOVS, shift or rotate by value” variants: is the shift amount, in the range 0 to 31 (when <shift> = LSL) or 1 to 31 (when <shift> = ROR), or 1 to 32 (when <shift> = LSR or ASR), encoded in the “imm3:imm2” field as <amount> modulo 32.

Alias Conditions

Alias	Of variant	Is preferred when
ASRS (immediate)	T3 MOVS, shift or rotate by value, A1 MOVS, shift or rotate by value	S == '1' && stype == '10'
ASRS (immediate)	T2	op == '10' && !InITBlock()
ASRS (immediate)	T3 MOVS, shift or rotate by value	Unconditionally
ASR (immediate)	T3 MOV, shift or rotate by value, A1 MOV, shift or rotate by value	S == '0' && stype == '10'
ASR (immediate)	T2	op == '10' && InITBlock()
ASR (immediate)	T3 MOV, shift or rotate by value	Unconditionally
LSLS (immediate)	A1 MOVS, shift or rotate by value	S == '1' && imm5 != '00000' && stype == '00'
LSLS (immediate)	T2	op == '00' && imm5 != '00000' && !InITBlock()
LSLS (immediate)	T3 MOVS, shift or rotate by value	(S == '1' && stype == '00' && !(imm3 == '000')) || (S == '1' && stype == '00' && !(imm2 == '00'))
LSLS (immediate)	T3 MOVS, shift or rotate by value	Unconditionally
LSL (immediate)	A1 MOV, shift or rotate by value	S == '0' && imm5 != '00000' && stype == '00'
LSL (immediate)	T2	op == '00' && imm5 != '00000' && InITBlock()
LSL (immediate)	T3 MOV, shift or rotate by value	(S == '0' && stype == '00' && !(imm3 == '000')) || (S == '0' && stype == '00' && !(imm2 == '00'))

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Alias	Of variant	Is preferred when
LSL (immediate)	T3 MOV, shift or rotate by value	Unconditionally
LSRS (immediate)	T3 MOVS, shift or rotate by value, A1 MOVS, shift or rotate by value	S == '1' && stype == '01'
LSRS (immediate)	T2	op == '01' && !InITBlock()
LSRS (immediate)	T3 MOVS, shift or rotate by value	Unconditionally
LSR (immediate)	T3 MOV, shift or rotate by value, A1 MOV, shift or rotate by value	S == '0' && stype == '01'
LSR (immediate)	T2	op == '01' && InITBlock()
LSR (immediate)	T3 MOV, shift or rotate by value	Unconditionally
RORS (immediate)	A1 MOVS, shift or rotate by value	S == '1' && imm5 != '00000' && stype == '11'
RORS (immediate)	T3 MOVS, shift or rotate by value	(S == '1' && stype == '11' && !(imm3 == '000')) || (S == '1' && stype == '11' && !(imm2 == '00'))
ROR (immediate)	A1 MOV, shift or rotate by value	S == '0' && imm5 != '00000' && stype == '11'
ROR (immediate)	T3 MOV, shift or rotate by value	(S == '0' && stype == '11' && !(imm3 == '000')) || (S == '0' && stype == '11' && !(imm2 == '00'))
RRXS	A1 MOVS, rotate right with extend	S == '1' && imm5 == '00000' && stype == '11'
RRXS	T3 MOVS, rotate right with extend	S == '1' && imm3 == '000' && imm2 == '00' && stype == '11'
RRX	A1 MOV, rotate right with extend	S == '0' && imm5 == '00000' && stype == '11'
RRX	T3 MOV, rotate right with extend	S == '0' && imm3 == '000' && imm2 == '00' && stype == '11'

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    bits(32) shifted;
    bit carry;
    (shifted, carry) = Shift_C(R[m], shift_t, shift_n, PSTATE.C);
    constant bits(32) result = shifted;
    if d == 15 then
        if setflags then
            ALUExceptionReturn(result);
        else
            ALUWritePC(result);
    else
        R[d] = result;
        if setflags then
            PSTATE.N = result<31>;
            PSTATE.Z = IsZeroBit(result);
            PSTATE.C = carry;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
            // PSTATE.V unchanged
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.114 MOV, MOVS (register-shifted register)

Move (register-shifted register) copies a register-shifted register value to the destination register. It can optionally update the condition flags based on the value.

This instruction is used by the aliases ASRS (register), ASR (register), LSLS (register), LSL (register), LSRS (register), LSR (register), RORS (register), and ROR (register).

It has encodings from the following instruction sets: A32 (A1) and T32 (T1 and T2).

A1

Encoding for the Flag setting variant

Applies when `(S == 1)`

```
MOVS{<c>}{<q>}  <Rd>, <Rm>, <shift> <Rs>
```

Encoding for the Not flag setting variant

Applies when `(S == 0)`

```
MOV{<c>}{<q>}  <Rd>, <Rm>, <shift> <Rs>
```

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant integer m = UInt(Rm);
constant integer s = UInt(Rs);
constant boolean setflags = (S == '1');
constant SRType shift_t = DecodeRegShift(stype);
if d == 15 || m == 15 || s == 15 then UNPREDICTABLE;
```

T1

Encoding for the Arithmetic shift right variant

Applies when `(op == 0100)`

```
MOV<c>{<q>}  <Rdm>, <Rdm>, ASR <Rs> // (InITBlock())
```

```
MOVS{<q>}  <Rdm>, <Rdm>, ASR <Rs> // (Outside IT block)
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Encoding for the Logical shift left variant

Applies when `(op == 0010)`

```
MOV<c>{<q>}  <Rdm>, <Rdm>, LSL <Rs> // (InITBlock())
```

```
MOVS{<q>}  <Rdm>, <Rdm>, LSL <Rs> // (Outside IT block)
```

Encoding for the Logical shift right variant

Applies when `(op == 0011)`

```
MOV<c>{<q>}  <Rdm>, <Rdm>, LSR <Rs> // (InITBlock())
```

```
MOVS{<q>}  <Rdm>, <Rdm>, LSR <Rs> // (Outside IT block)
```

Encoding for the Rotate right variant

Applies when `(op == 0111)`

```
MOV<c>{<q>}  <Rdm>, <Rdm>, ROR <Rs> // (InITBlock())
```

```
MOVS{<q>}  <Rdm>, <Rdm>, ROR <Rs> // (Outside IT block)
```

Decode for all variants of this encoding

```
if ! op IN {'0010', '0011', '0100', '0111'} then SEE "Related encodings";
constant integer d = UInt(Rdm);
constant integer m = UInt(Rdm);
constant integer s = UInt(Rs);
constant boolean setflags = !InITBlock();
constant SRType shift_t = DecodeRegShift(op<2>:op<0>);
```

T2

15 … 7	6 5	4	3 … 0	15 … 12	11 … 8	7 … 4	3 … 0
1 1 1 1 1 0 1 0 0	stype	S	Rm	1 1 1 1	Rd	0 0 0 0	Rs

Encoding for the Flag setting variant

Applies when `(S == 1)`

```
MOVS{<c>}{<q>}  <Rd>, <Rm>, <shift> <Rs>
```

```
MOVS.W  <Rd>, <Rm>, <shift> <Rs> // (Outside IT block, and <Rd>, <Rm>, <shift>, <Rs> can be represented
    ↪in T1)
```

Encoding for the Not flag setting variant

Applies when `(S == 0)`

```
MOV{<c>}{<q>}  <Rd>, <Rm>, <shift> <Rs>
```

```
MOV<c>.W  <Rd>, <Rm>, <shift> <Rs> // (Inside IT block, and <Rd>, <Rm>, <shift>, <Rs> can be represented
    ↪in T1)
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant integer m = UInt(Rm);
constant integer s = UInt(Rs);
constant boolean setflags = (S == '1');
constant SRType shift_t = DecodeRegShift(stype);
// Armv8-A removes UNPREDICTABLE for R13
if d == 15 || m == 15 || s == 15 then UNPREDICTABLE;
```

Related encodings: In encoding T1, for an `op` field value that is not described above, see Data-processing (two low registers).

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

Is the general-purpose destination register, encoded in the “Rd” field.

<Rm>

Is the general-purpose source register, encoded in the “Rm” field.

<shift>

Is the type of shift to be applied to the second source register, encoded in “stype”:

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

<Rs>

Is the general-purpose source register holding a shift amount in its bottom 8 bits, encoded in the “Rs” field.

<Rdm>

Is the general-purpose source register and the destination register, encoded in the “Rdm” field.

Alias Conditions

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Alias	Of variant	Is preferred when
ASRS (register)	A1 Flag setting	S == '1' && stype == '10'
ASRS (register)	T1 Arithmetic shift right	op == '0100' && !InITBlock()
ASRS (register)	T2 Flag setting	stype == '10' && S == '1'
ASRS (register)	T2 Flag setting	Unconditionally
ASR (register)	A1 Not flag setting	S == '0' && stype == '10'
ASR (register)	T1 Arithmetic shift right	op == '0100' && InITBlock()
ASR (register)	T2 Not flag setting	stype == '10' && S == '0'
ASR (register)	T2 Not flag setting	Unconditionally
LSLS (register)	A1 Flag setting	S == '1' && stype == '00'
LSLS (register)	T1 Logical shift left	op == '0010' && !InITBlock()
LSLS (register)	T2 Flag setting	stype == '00' && S == '1'
LSLS (register)	T2 Flag setting	Unconditionally
LSL (register)	A1 Not flag setting	S == '0' && stype == '00'
LSL (register)	T1 Logical shift left	op == '0010' && InITBlock()
LSL (register)	T2 Not flag setting	stype == '00' && S == '0'
LSL (register)	T2 Not flag setting	Unconditionally
LSRS (register)	A1 Flag setting	S == '1' && stype == '01'
LSRS (register)	T1 Logical shift right	op == '0011' && !InITBlock()
LSRS (register)	T2 Flag setting	stype == '01' && S == '1'
LSRS (register)	T2 Flag setting	Unconditionally
LSR (register)	A1 Not flag setting	S == '0' && stype == '01'
LSR (register)	T1 Logical shift right	op == '0011' && InITBlock()
LSR (register)	T2 Not flag setting	stype == '01' && S == '0'
LSR (register)	T2 Not flag setting	Unconditionally
RORS (register)	A1 Flag setting	S == '1' && stype == '11'
RORS (register)	T1 Rotate right	op == '0111' && !InITBlock()
RORS (register)	T2 Flag setting	stype == '11' && S == '1'
RORS (register)	T2 Flag setting	Unconditionally
ROR (register)	A1 Not flag setting	S == '0' && stype == '11'
ROR (register)	T1 Rotate right	op == '0111' && InITBlock()
ROR (register)	T2 Not flag setting	stype == '11' && S == '0'
ROR (register)	T2 Not flag setting	Unconditionally

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant integer shift_n = UInt(R[s]<7:0>);
    bits(32) result;
    bit carry;
    (result, carry) = Shift_C(R[m], shift_t, shift_n, PSTATE.C);
    R[d] = result;
    if setflags then
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
PSTATE.N = result<31>;
PSTATE.Z = IsZeroBit(result);
PSTATE.C = carry;
// PSTATE.V unchanged
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.115 MOVT

Move Top writes an immediate value to the top halfword of the destination register. It does not affect the contents of the bottom halfword.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding

```
MOVT{<c>}{<q>}  <Rd>, #<imm16>
```

Decode for this encoding

```
constant integer d = UInt(Rd);
constant bits(16) imm16 = imm4:imm12;
if d == 15 then UNPREDICTABLE;
```

T1

Encoding

```
MOVT{<c>}{<q>}  <Rd>, #<imm16>
```

Decode for this encoding

```
constant integer d = UInt(Rd);
constant bits(16) imm16 = imm4:i:imm3:imm8;
// Armv8-A removes UNPREDICTABLE for R13
if d == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

<Rd>

Is the general-purpose destination register, encoded in the “Rd” field.

<imm16>

For the “A1” variant: is a 16-bit unsigned immediate, in the range 0 to 65535, encoded in the “imm4:imm12” field.

For the “T1” variant: is a 16-bit unsigned immediate, in the range 0 to 65535, encoded in the “imm4:i:imm3:imm8” field.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    R[d]<31:16> = imm16;
    // R[d]<15:0> unchanged
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.116 MRC

Move to general-purpose register from System register. This instruction copies the value of a System register to a general-purpose register.

The System register descriptions identify valid encodings for this instruction. Other encodings are UNDEFINED. For more information see About the AArch32 System register interface and General behavior of System registers.

In an implementation that includes EL2, MRC accesses to system control registers can be trapped to Hyp mode, meaning that an attempt to execute an MRC instruction in a Non-secure mode other than Hyp mode, that would be permitted in the absence of the Hyp trap controls, generates a Hyp Trap exception. For more information, see EL2 configurable instruction enables, disables, and traps.

Because of the range of possible traps to Hyp mode, the MRC pseudocode does not show these possible traps.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding

```
MRC{<c>}{<q>}  <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer cp = if cp15 == '0' then 14 else 15;
// Armv8-A removes UNPREDICTABLE for R13
```

T1

Encoding

```
MRC{<c>}{<q>}  <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer cp = if cp15 == '0' then 14 else 15;
// Armv8-A removes UNPREDICTABLE for R13
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<coproc>

Is the System register encoding space, encoded in “cp15”:

cp15	<coproc>
0	p14
1	p15

<opc1>

Is the opc1 parameter within the System register encoding space, in the range 0 to 7, encoded in the “opc1” field.

<Rt>

Is the general-purpose register to be transferred or `APSR_nzcv` (encoded as `0b1111`), encoded in the “Rt” field. If `APSR_nzcv` is used, bits [31:28] of the transferred value are written to the PSTATE condition flags.

<CRn>

Is the CRn parameter within the System register encoding space, in the range c0 to c15, encoded in the “CRn” field.

<CRm>

Is the CRm parameter within the System register encoding space, in the range c0 to c15, encoded in the “CRm” field.

<opc2>

Is the opc2 parameter within the System register encoding space, in the range 0 to 7, encoded in the “opc2” field.

The possible values of { `<coproc>`, `<opc1>`, `<CRn>`, `<CRm>`, `<opc2>` } encode the entire System register and System instruction encoding space. Not all of this space is allocated, and the System register and System instruction descriptions identify the allocated encodings.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();

    if t != 15 || AArch32.SysRegReadCanWriteAPSR(cp, ThisInstr()) then
        AArch32.SysRegRead(cp, ThisInstr(), t);
    else
        UNPREDICTABLE;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.117 MRRC

Move to two general-purpose registers from System register. This instruction copies the value of a System register to two general-purpose registers.

The System register descriptions identify valid encodings for this instruction. Other encodings are UNDEFINED. For more information see About the AArch32 System register interface and General behavior of System registers.

In an implementation that includes EL2, MRRC accesses to System registers can be trapped to Hyp mode, meaning that an attempt to execute an MRRC instruction in a Non-secure mode other than Hyp mode, that would be permitted in the absence of the Hyp trap controls, generates a Hyp Trap exception. For more information, see EL2 configurable instruction enables, disables, and traps.

Because of the range of possible traps to Hyp mode, the MRRC pseudocode does not show these possible traps.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding

```
MRRC{<c>}{<q>}  <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer t2 = UInt(Rt2);
constant integer cp = if cp15 == '0' then 14 else 15;
// Armv8-A removes UNPREDICTABLE for R13
if t == 15 || t2 == 15 || t == t2 then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `t == t2`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The value in the destination register is UNKNOWN.

T1

Encoding

```
MRRC{<c>}{<q>}  <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for this encoding

```
constant integer t = UInt(Rt);
constant integer t2 = UInt(Rt2);
constant integer cp = if cp15 == '0' then 14 else 15;
// Armv8-A removes UNPREDICTABLE for R13
if t == 15 || t2 == 15 || t == t2 then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `t == t2`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.
- The value in the destination register is UNKNOWN.

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<coproc>

Is the System register encoding space, encoded in “cp15”:

cp15	<coproc>
0	p14
1	p15

<opc1>

Is the opc1 parameter within the System register encoding space, in the range 0 to 15, encoded in the “opc1” field.

<Rt>

Is the first general-purpose register that is transferred into, encoded in the “Rt” field.

<Rt2>

Is the second general-purpose register that is transferred into, encoded in the “Rt2” field.

<CRm>

Is the CRm parameter within the System register encoding space, in the range c0 to c15, encoded in the “CRm” field.

The possible values of { `<coproc>`, `<opc1>`, `<CRm>` } encode the entire System register encoding space. Not all of this space is allocated, and the System register descriptions identify the allocated encodings.

For the permitted uses of these instructions, as described in this manual, `<Rt2>` transfers bits[63:32] of the selected System register, while `<Rt>` transfers bits[31:0].

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    AArch32.SysRegRead64(cp, ThisInstr(), t, t2);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.118 MRS

Move Special register to general-purpose register moves the value of the APSR, CPSR, or SPSR_<current_mode> into a general-purpose register.

Arm recommends the `APSR` form when only the N, Z, C, V, Q, and GE[3:0] bits are being written. For more information, see APSR.

An `MRS` that accesses the SPSRs is UNPREDICTABLE if executed in User mode or System mode.

An `MRS` that is executed in User mode and accesses the CPSR returns an UNKNOWN value for the CPSR.{E, A, I, F, M} fields.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

31–28	27	26	25	23–22	21	20	19–16	15–12	11	10	9	8	7	6	5	4	3–0
!=1111 (cond)	0	0	0	1 0	R	0	0	(1) (1) (1) (1) (mask)	Rd		(0)	(0)	0	0	0	0	(0) (0) (0) (0) (Rn)

B, m

Encoding

```
MRS{<c>}{<q>}  <Rd>, <spec_reg>
```

Decode for this encoding

Listing F5-52

```
constant integer d = UInt(Rd);
constant boolean read_spsr = (R == '1');
if d == 15 then UNPREDICTABLE; end
```

T1

15–13	12	11	10	9–6	5	4	3–0	15	14–12	11–8	7	6	5	4–0
1 1 1	1	0	0	1 1 1 1	1	R	(1) (1) (1) (1)	1	0 (0) 0	Rd	(0)	(0)	0	(0) (0) (0) (0) (0)

Encoding

```
MRS{<c>}{<q>}  <Rd>, <spec_reg>
```

Decode for this encoding

Listing F5-53

```
constant integer d = UInt(Rd);
constant boolean read_spsr = (R == '1');
// Armv8-A removes UNPREDICTABLE for R13
if d == 15 then UNPREDICTABLE; end
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

Is the general-purpose destination register, encoded in the “Rd” field.

<spec_reg>

Is the special register to be accessed, encoded in “R”:

R	<spec_reg>
0	CPSR|APSR
1	SPSR

Operation

Listing F5-54

```
if ConditionPassed() then
    EncodingSpecificOperations();
    if read_spsr then
        if PSTATE.M IN {M32_User,M32_System} then
            UNPREDICTABLE;
        else
            R[d] = SPSR_curr[];
        end
    else
        // CPSR has same bit assignments as SPSR, but with the IT, J, SS, IL, and T bits masked out.
        constant bits(32) maskval = '11111000 11101111 00000011 11011111';
        bits(32) psr_val = GetPSRFromPSTATE(AArch32_NonDebugState, 32) AND maskval;
        if PSTATE.EL == EL0 then
            // If accessed from User mode return UNKNOWN values for E, A, I, F bits, bits<9:6>,
            // and for the M field, bits<4:0>
            psr_val<22> = bits(1) UNKNOWN;
            psr_val<9:6> = bits(4) UNKNOWN;
            psr_val<4:0> = bits(5) UNKNOWN;
        end
        R[d] = psr_val;
    end
end
```

CONSTRAINED UNPREDICTABLE behavior

If `PSTATE.M IN {M32_User, M32_System} && read_spsr`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.119 MRS (Banked register)

Move Banked or Special register to general-purpose register

Move to Register from Banked or Special register moves the value from the Banked general-purpose register or Saved Program Status Registers (SPSRs) of the specified mode, or the value of ELR_hyp, to a general-purpose register.

MRS (Banked register) is UNPREDICTABLE if executed in User mode.

When EL3 is using AArch64, if an MRS (Banked register) instruction that is executed in a Secure EL1 mode would access SPSR_mon, SP_mon, or LR_mon, it is trapped to EL3.

The effect of using an MRS (Banked register) instruction with a register argument that is not valid for the current mode is UNPREDICTABLE. For more information see Usage restrictions on the Banked register transfer instructions.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

31–28	27	26	25	23–22	21	20	19–16	15–12	11	10	9	8	7	6	5	4	3–0
!=1111 (cond)	0	0	0	1 0	R	0	0	M1	Rd		(0)	(0)	1 (B)	M	0	0	0 0	(0) (0) (0) (0) (Rn)

Encoding

```
MRS{<c>}{<q>}  <Rd>, <banked_reg>
```

Decode for this encoding

```
constant integer d = UInt(Rd);
constant boolean read_spsr = (R == '1');
if d == 15 then UNPREDICTABLE;
constant bits(5) SYSm = M:M1;
```

T1

15–13	12	11	10	9–6	5	4	3–0	15	14–12	11–8	7	6	5	4	3–0
1 1 1	1	0	0	1 1 1 1	1	R	M1	1	0(0)0	Rd	(0)	(0)	1	M	(0) (0) (0) (0)

Encoding

```
MRS{<c>}{<q>}  <Rd>, <banked_reg>
```

Decode for this encoding

```
constant integer d = UInt(Rd);
constant boolean read_spsr = (R == '1');
// Armv8-A removes UNPREDICTABLE for R13
if d == 15 then UNPREDICTABLE;
constant bits(5) SYSm = M:M1;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

Is the general-purpose destination register, encoded in the “Rd” field.

<banked_reg>

Is the name of the banked register to be transferred to or from, encoded in “R:M:M1”:

R	M	M1	<banked_reg>
0	0	0000	R8_usr
0	0	0001	R9_usr
0	0	0010	R10_usr
0	0	0011	R11_usr
0	0	0100	R12_usr
0	0	0101	SP_usr
0	0	0110	LR_usr
0	0	0111	UNPREDICTABLE
0	0	1000	R8_fiq
0	0	1001	R9_fiq
0	0	1010	R10_fiq
0	0	1011	R11_fiq
0	0	1100	R12_fiq
0	0	1101	SP_fiq
0	0	1110	LR_fiq
0	0	1111	UNPREDICTABLE
0	1	0000	LR_irq
0	1	0001	SP_irq
0	1	0010	LR_svc
0	1	0011	SP_svc
0	1	0100	LR_abt
0	1	0101	SP_abt
0	1	0110	LR_und
0	1	0111	SP_und
0	1	10xx	UNPREDICTABLE
0	1	1100	LR_mon
0	1	1101	SP_mon

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

R	M	M1	<banked_reg>
0	1	1110	ELR_hyp
0	1	1111	SP_hyp
1	0	0xxx	UNPREDICTABLE
1	0	10xx	UNPREDICTABLE
1	0	110x	UNPREDICTABLE
1	0	1110	SPSR_fiq
1	0	1111	UNPREDICTABLE
1	1	0000	SPSR_irq
1	1	0001	UNPREDICTABLE
1	1	0010	SPSR_svc
1	1	0011	UNPREDICTABLE
1	1	0100	SPSR_abt
1	1	0101	UNPREDICTABLE
1	1	0110	SPSR_und
1	1	0111	UNPREDICTABLE
1	1	10xx	UNPREDICTABLE
1	1	1100	SPSR_mon
1	1	1101	UNPREDICTABLE
1	1	1110	SPSR_hyp
1	1	1111	UNPREDICTABLE

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    if PSTATE.EL == EL0 then
        constant Constraint c = ConstrainUnpredictable(Unpredictable_BankedRegister);
        case c of
            when Constraint_UNDEF
                UNDEFINED;
            when Constraint_NOP
                ExecuteAsNOP();
            when Constraint_UNKNOWN
                R[d] = bits(32) UNKNOWN;
    else
        constant bits(5) mode = PSTATE.M;
        if read_spsr then
            // Check for CONSTRAINED UNPREDICTABLE cases
            boolean valid;
            Constraint c = Constraint_NONE;
            (valid, c)   = SPSRaccessValid(SYSm, mode);

            if valid then
                case SYSm of
                    when '01110'  R[d] = SPSR_fiq<31:0>;
                    when '10000'  R[d] = SPSR_irq<31:0>;
                    when '10010'  R[d] = SPSR_svc<31:0>;
                    when '10100'  R[d] = SPSR_abt<31:0>;
                    when '10110'  R[d] = SPSR_und<31:0>;
                    when '11100'
                        if !ELUsingAArch32(EL3) then AArch64.MonitorModeTrap();
                        R[d] = SPSR_mon;
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
                when '11110'  R[d] = SPSR_hyp<31:0>;
        else
            case c of
                when Constraint_UNDEF
                    UNDEFINED;
                when Constraint_NOP
                    ExecuteAsNOP();
                when Constraint_UNKNOWN
                    R[d] = bits(32) UNKNOWN;
                when Constraint_ANYREG
                    R[d] = ReadAnyAllocatedSPSR();
    else
        // Check for CONSTRAINED UNPREDICTABLE cases
        boolean valid;
        Constraint c = Constraint_NONE;
        (valid, c)   = BankedRegisterAccessValid(SYSm, mode);

        if valid then
            case SYSm of
                when '00xxx'                        // Access the User mode registers
                    constant integer m = UInt(SYSm<2:0>) + 8;
                    R[d] = Rmode[m,M32_User];
                when '01xxx'                        // Access the FIQ mode registers
                    constant integer m = UInt(SYSm<2:0>) + 8;
                    R[d] = Rmode[m,M32_FIQ];
                when '1000x'                        // Access the IRQ mode registers
                    // LR when SYSm<0> == 0, otherwise SP
                    constant integer m = 14 - UInt(SYSm<0>);
                    R[d] = Rmode[m,M32_IRQ];
                when '1001x'                        // Access the Supervisor mode registers
                    // LR when SYSm<0> == 0, otherwise SP
                    constant integer m = 14 - UInt(SYSm<0>);
                    R[d] = Rmode[m,M32_Svc];
                when '1010x'                        // Access the Abort mode registers
                    // LR when SYSm<0> == 0, otherwise SP
                    constant integer m = 14 - UInt(SYSm<0>);
                    R[d] = Rmode[m,M32_Abort];
                when '1011x'                        // Access the Undefined mode registers
                    // LR when SYSm<0> == 0, otherwise SP
                    constant integer m = 14 - UInt(SYSm<0>);
                    R[d] = Rmode[m,M32_Undef];
                when '1110x'                        // Access Monitor registers
                    if !ELUsingAArch32(EL3) then AArch64.MonitorModeTrap();
                    // LR when SYSm<0> == 0, otherwise SP
                    constant integer m = 14 - UInt(SYSm<0>);
                    R[d] = Rmode[m,M32_Monitor];
                when '11110'                        // Access ELR_hyp register
                    R[d] = ELR_hyp;
                when '11111'                        // Access SP_hyp register
                    R[d] = Rmode[13,M32_Hyp];
        else
            case c of
                when Constraint_UNDEF
                    UNDEFINED;
                when Constraint_NOP
                    ExecuteAsNOP();
                when Constraint_UNKNOWN
                    R[d] = bits(32) UNKNOWN;
                when Constraint_ANYREG
                    R[d] = ReadAnyAllocatedRegister();
```

CONSTRAINED UNPREDICTABLE behavior

If `PSTATE.EL == EL0`, then one of the following behaviors must occur:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

- The instruction is UNDEFINED.
- The instruction executes as NOP.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.120 MSR (Banked register)

Move general-purpose register to Banked or Special register

Move to Banked or Special register from general-purpose register moves the value of a general-purpose register to the Banked general-purpose register or Saved Program Status Registers (SPSRs) of the specified mode, or to ELR_hyp.

`MSR` (Banked register) is UNPREDICTABLE if executed in User mode.

When EL3 is using AArch64, if an MSR (Banked register) instruction that is executed in a Secure EL1 mode would access SPSR_mon, SP_mon, or LR_mon, it is trapped to EL3.

The effect of using an `MSR` (Banked register) instruction with a register argument that is not valid for the current mode is UNPREDICTABLE. For more information see Usage restrictions on the Banked register transfer instructions.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

31–28	27	26	25	23	22	21	20	19–16	15–12	11	10	9	8	7	6	4	3–0
!=1111 (cond)	0	0	0	0	R	1	0	M1	(1)(1)(1)(1) (Rd)	(0)	(0)	1 (B)	M	0	0	0	Rn

Encoding

```
MSR{<c>}{<q>}  <banked_reg>, <Rn>
```

Decode for this encoding

```
constant integer n = UInt(Rn);
constant boolean write_spsr = (R == '1');
if n == 15 then UNPREDICTABLE;
constant bits(5) SYSm = M:M1;
```

T1

15–13	12	11	10	9–6	5	4	3–0	15	14–12	11–8	7	6	5	4	3–0
1 1 1	1	0	0	1 1 1 0	0	R	Rn	1	0(0)0	M1	(0)	(0)	1	M	(0)(0)(0)(0)

Encoding

```
MSR{<c>}{<q>}  <banked_reg>, <Rn>
```

Decode for this encoding

```
constant integer n = UInt(Rn);
constant boolean write_spsr = (R == '1');
// Armv8-A removes UNPREDICTABLE for R13
if n == 15 then UNPREDICTABLE;
constant bits(5) SYSm = M:M1;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<banked_reg>

Is the name of the banked register to be transferred to or from, encoded in “R:M:M1”:

R	M	M1	<banked_reg>
0	0	0000	R8_usr
0	0	0001	R9_usr
0	0	0010	R10_usr
0	0	0011	R11_usr
0	0	0100	R12_usr
0	0	0101	SP_usr
0	0	0110	LR_usr
0	0	0111	UNPREDICTABLE
0	0	1000	R8_fiq
0	0	1001	R9_fiq
0	0	1010	R10_fiq
0	0	1011	R11_fiq
0	0	1100	R12_fiq
0	0	1101	SP_fiq
0	0	1110	LR_fiq
0	0	1111	UNPREDICTABLE
0	1	0000	LR_irq
0	1	0001	SP_irq
0	1	0010	LR_svc
0	1	0011	SP_svc
0	1	0100	LR_abt
0	1	0101	SP_abt
0	1	0110	LR_und
0	1	0111	SP_und
0	1	10xx	UNPREDICTABLE
0	1	1100	LR_mon
0	1	1101	SP_mon
0	1	1110	ELR_hyp
0	1	1111	SP_hyp
1	0	0xxx	UNPREDICTABLE

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

R	M	M1	<banked_reg>
1	0	10xx	UNPREDICTABLE
1	0	110x	UNPREDICTABLE
1	0	1110	SPSR_fiq
1	0	1111	UNPREDICTABLE
1	1	0000	SPSR_irq
1	1	0001	UNPREDICTABLE
1	1	0010	SPSR_svc
1	1	0011	UNPREDICTABLE
1	1	0100	SPSR_abt
1	1	0101	UNPREDICTABLE
1	1	0110	SPSR_und
1	1	0111	UNPREDICTABLE
1	1	10xx	UNPREDICTABLE
1	1	1100	SPSR_mon
1	1	1101	UNPREDICTABLE
1	1	1110	SPSR_hyp
1	1	1111	UNPREDICTABLE

<Rn>

Is the general-purpose source register, encoded in the “Rn” field.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    if PSTATE.EL == EL0 then
        constant Constraint c = ConstrainUnpredictable(Unpredictable_BankedRegister);
        case c of
            when Constraint_UNDEF
                UNDEFINED;
            when Constraint_NOP, Constraint_UNKNOWN
                ExecuteAsNOP();
    else
        constant bits(5) mode = PSTATE.M;
        if write_spsr then
            // Check for CONSTRAINED UNPREDICTABLE cases
            boolean valid;
            Constraint c = Constraint_NONE;
            (valid, c)   = SPSRaccessValid(SYSm, mode);

            if valid then
                case SYSm of
                    when '01110'  SPSR_fiq<31:0> = R[n];
                    when '10000'  SPSR_irq<31:0> = R[n];
                    when '10010'  SPSR_svc<31:0> = R[n];
                    when '10100'  SPSR_abt<31:0> = R[n];
                    when '10110'  SPSR_und<31:0> = R[n];
                    when '11100'
                        if !ELUsingAArch32(EL3) then AArch64.MonitorModeTrap();
                        SPSR_mon<31:0> = R[n];
                    when '11110'  SPSR_hyp<31:0> = R[n];
            else
                case c of
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
                when Constraint_UNDEF
                    UNDEFINED;
                when Constraint_NOP, Constraint_UNKNOWN
                    ExecuteAsNOP();
                when Constraint_ANYREG
                    WriteAnyAllocatedSPSR(R[n]);
        else
            // Check for CONSTRAINED UNPREDICTABLE cases
            boolean valid;
            Constraint c = Constraint_NONE;
            (valid, c)   = BankedRegisterAccessValid(SYSm, mode);

            if valid then
                case SYSm of
                    when '00xxx'                        // Access the User mode registers
                        constant integer m = UInt(SYSm<2:0>) + 8;
                        Rmode[m,M32_User] = R[n];
                    when '01xxx'                        // Access the FIQ mode registers
                        constant integer m = UInt(SYSm<2:0>) + 8;
                        Rmode[m,M32_FIQ] = R[n];
                    when '1000x'                        // Access the IRQ mode registers
                        // LR when SYSm<0> == 0, otherwise SP
                        constant integer m = 14 - UInt(SYSm<0>);
                        Rmode[m,M32_IRQ] = R[n];
                    when '1001x'                        // Access the Supervisor mode registers
                        // LR when SYSm<0> == 0, otherwise SP
                        constant integer m = 14 - UInt(SYSm<0>);
                        Rmode[m,M32_Svc] = R[n];
                    when '1010x'                        // Access the Abort mode registers
                        // LR when SYSm<0> == 0, otherwise SP
                        constant integer m = 14 - UInt(SYSm<0>);
                        Rmode[m,M32_Abort] = R[n];
                    when '1011x'                        // Access the Undefined mode registers
                        // LR when SYSm<0> == 0, otherwise SP
                        constant integer m = 14 - UInt(SYSm<0>);
                        Rmode[m,M32_Undef] = R[n];
                    when '1110x'                        // Access Monitor registers
                        if !ELUsingAArch32(EL3) then AArch64.MonitorModeTrap();
                        // LR when SYSm<0> == 0, otherwise SP
                        constant integer m = 14 - UInt(SYSm<0>);
                        Rmode[m,M32_Monitor] = R[n];
                    when '11110'                        // Access ELR_hyp register
                        ELR_hyp = R[n];
                    when '11111'                        // Access SP_hyp register
                        Rmode[13,M32_Hyp] = R[n];
            else
                case c of
                    when Constraint_UNDEF
                        UNDEFINED;
                    when Constraint_NOP, Constraint_UNKNOWN
                        ExecuteAsNOP();
                    when Constraint_ANYREG
                        WriteAnyAllocatedRegister(R[n]);
```

CONSTRAINED UNPREDICTABLE behavior

If `PSTATE.EL == EL0`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.121 MSR (immediate)

Move immediate value to Special register moves selected bits of an immediate value to the corresponding bits in the APSR, CPSR, or SPSR_<current_mode>.

Because of the Do-Not-Modify nature of its reserved bits, the immediate form of `MSR` is normally only useful at the Application level for writing to `APSR_nzcvq` (`CPSR_f`).

If an `MSR` (immediate) moves selected bits of an immediate value to the CPSR, the PE checks whether the value being written to PSTATE.M is legal. See Illegal changes to PSTATE.M.

An `MSR` (immediate) executed in User mode:

- Is CONSTRAINED UNPREDICTABLE if it attempts to update the SPSR.
- Otherwise, does not update any CPSR field that is accessible only at EL1 or higher,

An `MSR` (immediate) executed in System mode is CONSTRAINED UNPREDICTABLE if it attempts to update the SPSR.

The CPSR.E bit is writable from any mode using an `MSR` instruction. Arm deprecates using this to change its value.

31–28	27–26	25	24–23	22	21–20	19–16	15–12	11–0
!=1111 (cond)	0 0	1	1 0	R	1 0	mask	(1)(1)(1)(1)	imm12

Encoding

Applies when `(!(R == 0 && mask == 0000))`

```
MSR{<c>}{<q>}  <spec_reg>, #<imm>
```

Decode for this encoding

```
if mask == '0000' && R == '0' then SEE "Related encodings";
constant bits(32) imm32 = A32ExpandImm(imm12);
constant boolean write_spsr = (R == '1');
if mask == '0000' then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `mask == '0000' && R == '1'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Related encodings: Move Special Register and Hints (immediate).

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

<spec_reg>

Is one of:

- `APSR_<bits>`.
- `CPSR_<fields>`.
- `SPSR_<fields>`.

For CPSR and SPSR, <fields> is a sequence of one or more of the following:

c mask<0> = '1' to enable writing of bits<7:0> of the destination PSR.
x mask<1> = '1' to enable writing of bits<15:8> of the destination PSR.
s mask<2> = '1' to enable writing of bits<23:16> of the destination PSR.
f mask<3> = '1' to enable writing of bits<31:24> of the destination PSR.

For APSR, <bits> is one of `nzcvq`, `g`, or `nzcvqg`. These map to the following CPSR_<fields> values:

- `APSR_nzcvq` is the same as `CPSR_f` (mask== '1000').
- `APSR_g` is the same as `CPSR_s` (mask == '0100').
- `APSR_nzcvqg` is the same as `CPSR_fs` (mask == '1100').

Arm recommends the `APSR_<bits>` forms when only the N, Z, C, V, Q, and GE[3:0] bits are being written. For more information, see The Application Program Status Register, APSR.

<imm>

Is an immediate value. See Modified immediate constants in A32 instructions for the range of values.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    if write_spsr then
        if PSTATE.M IN {M32_User,M32_System} then
            UNPREDICTABLE;
        else
            SPSRWriteByInstr(imm32, mask);
    else
        // Attempts to change to an illegal mode will invoke the Illegal Execution state mechanism
        CPSRWriteByInstr(imm32, mask);
```

CONSTRAINED UNPREDICTABLE behavior

If `PSTATE.M IN {M32_User,M32_System} && write_spsr`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.122 MSR (register)

Move general-purpose register to Special register moves selected bits of a general-purpose register to the APSR, CPSR or SPSR_<current_mode>.

Because of the Do-Not-Modify nature of its reserved bits, a read-modify-write sequence is normally required when the `MSR` instruction is being used at Application level and its destination is not `APSR_nzcvq` (`CPSR_f`).

If an `MSR` (register) moves selected bits of an immediate value to the CPSR, the PE checks whether the value being written to PSTATE.M is legal. See Illegal changes to PSTATE.M.

An `MSR` (register) executed in User mode:

- Is UNPREDICTABLE if it attempts to update the SPSR.
- Otherwise, does not update any CPSR field that is accessible only at EL1 or higher.

An `MSR` (register) executed in System mode is UNPREDICTABLE if it attempts to update the SPSR.

The CPSR.E bit is writable from any mode using an `MSR` instruction. Arm deprecates using this to change its value.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

31–28	27 26	25–23	22	21	20	19–16	15–12	11 10	9	8	7	6–4	3–0
!=1111 (cond)	0 0	0 1 0	R	1	0	mask	(1)(1)(1)(1) (Rd)	(0)(0)	0 (B)	(0) (m)	0	0 0 0	Rn

Encoding

```
MSR{<c>}{<q>}  <spec_reg>, <Rn>
```

Decode for this encoding

```
constant integer n = UInt(Rn);
constant boolean write_spsr = (R == '1');
if mask == '0000' then UNPREDICTABLE;
if n == 15 then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `mask == '0000'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.

T1

15–13	12 11	10	9–6	5	4	3–0	15	14–12	11–8	7 6	5	4–0
1 1 1	1 0	0	1 1 1 0	0	R	Rn	1	0(0)0	mask	(0)(0)	0	(0)(0)(0)(0)(0)

Encoding

```
MSR{<c>}{<q>}  <spec_reg>, <Rn>
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for this encoding

```
constant integer n = UInt(Rn);
constant boolean write_spsr = (R == '1');
if mask == '0000' then UNPREDICTABLE;
// Armv8-A removes UNPREDICTABLE for R13
if n == 15 then UNPREDICTABLE;
```

CONSTRAINED UNPREDICTABLE behavior

If `mask == '0000'`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as NOP.

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<spec_reg>

Is one of:

- `APSR_<bits>`.
- `CPSR_<fields>`.
- `SPSR_<fields>`.

For CPSR and SPSR, <fields> is a sequence of one or more of the following:

c mask<0> = ‘1’ to enable writing of bits<7:0> of the destination PSR.
x mask<1> = ‘1’ to enable writing of bits<15:8> of the destination PSR.
s mask<2> = ‘1’ to enable writing of bits<23:16> of the destination PSR.
f mask<3> = ‘1’ to enable writing of bits<31:24> of the destination PSR.

For APSR, <bits> is one of `nzcvq`, `g`, or `nzcvqg`. These map to the following CPSR_<fields> values:

- `APSR_nzcvq` is the same as `CPSR_f` (mask== ‘1000’).
- `APSR_g` is the same as `CPSR_s` (mask == ‘0100’).
- `APSR_nzcvqg` is the same as `CPSR_fs` (mask == ‘1100’).

Arm recommends the `APSR_<bits>` forms when only the N, Z, C, V, Q, and GE[3:0] bits are being written. For more information, see The Application Program Status Register, APSR.

<Rn>

Is the general-purpose source register, encoded in the “Rn” field.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    if write_spsr then
        if PSTATE.M IN {M32_User,M32_System} then
            UNPREDICTABLE;
        else
            SPSRWriteByInstr(R[n], mask);
    else
        // Attempts to change to an illegal mode will invoke the Illegal Execution state mechanism
        CPSRWriteByInstr(R[n], mask);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

CONSTRAINED UNPREDICTABLE behavior

If `write_spsr && PSTATE.M IN {M32_User,M32_System}`, then one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction executes as `NOP`.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.123 MUL, MULS

Multiply multiplies two register values. The least significant 32 bits of the result are written to the destination register. These 32 bits do not depend on whether the source register values are considered to be signed values or unsigned values.

Optionally, it can update the condition flags based on the result. In the T32 instruction set, this option is limited to only a few forms of the instruction. Use of this option adversely affects performance on many implementations.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1 and T2).

A1

Encoding for the Flag setting variant

Applies when `(S == 1)`

```
MULS{<c>}{<q>}  <Rd>, <Rn>{, <Rm>}
```

Encoding for the Not flag setting variant

Applies when `(S == 0)`

```
MUL{<c>}{<q>}  <Rd>, <Rn>{, <Rm>}
```

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean setflags = (S == '1');
if d == 15 || n == 15 || m == 15 then UNPREDICTABLE;
```

T1

Encoding

```
MUL<c>{<q>}  <Rdm>, <Rn>{, <Rdm>} // (InITBlock())
```

```
MULS{<q>}  <Rdm>, <Rn>{, <Rdm>} // (Outside IT block)
```

Decode for this encoding

```
constant integer d = UInt(Rdm);
constant integer n = UInt(Rn);
constant integer m = UInt(Rdm);
constant boolean setflags = !InITBlock();
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T2

15 … 7	6 … 4	3 … 0	15 … 12	11 … 8	7 … 6	5 … 4	3 … 0
1 1 1 1 1 0 1 1 0	0 0 0 (op1)	Rn	1 1 1 1 (Ra)	Rd	0 0	0 0 (op2)	Rm

Encoding

```
MUL{<c>}{<q>}  <Rd>, <Rn>{, <Rm>}
```

```
MUL<c>.W  <Rd>, <Rn>{, <Rm>} // (Inside IT block, and <Rd>, <Rn>, <Rm> can be represented in T1)
```

Decode for this encoding

```
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean setflags = FALSE;
// Armv8-A removes UNPREDICTABLE for R13
if d == 15 || n == 15 || m == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

Is the general-purpose destination register, encoded in the “Rd” field.

<Rn>

Is the first general-purpose source register holding the multiplicand, encoded in the “Rn” field.

<Rm>

Is the second general-purpose source register holding the multiplier, encoded in the “Rm” field. If omitted, `<Rd>` is used.

<Rdm>

Is the second general-purpose source register holding the multiplier and the destination register, encoded in the “Rdm” field.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant integer operand1 = SInt(R[n]); // operand1 = UInt(R[n]) produces the same final results
    constant integer operand2 = SInt(R[m]); // operand2 = UInt(R[m]) produces the same final results
    constant integer result = operand1 * operand2;
    R[d] = result<31:0>;
    if setflags then
        PSTATE.N = result<31>;
        PSTATE.Z = IsZeroBit(result<31:0>);
        // PSTATE.C, PSTATE.V unchanged
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.124 MVN, MVNS (immediate)

Bitwise NOT (immediate) writes the bitwise inverse of an immediate value to the destination register.

If the destination register is not the PC, the MVNS variant of the instruction updates the condition flags based on the result.

The field descriptions for `<Rd>` identify the encodings where the PC is permitted as the destination register. ARM deprecates any use of these encodings. However, when the destination register is the PC:

- The MVN variant of the instruction is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- The MVNS variant of the instruction performs an exception return without the use of the stack. In this case:
  - The PE branches to the address written to the PC, and restores PSTATE from SPSR_<current_mode>.
  - The PE checks SPSR_<current_mode> for an illegal return event. See Illegal return events from AArch32 state.
  - The instruction is UNDEFINED in Hyp mode.
  - The instruction is CONSTRAINED UNPREDICTABLE in User mode and System mode.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding for the MVN variant

Applies when `(S == 0)`

```
MVN{<c>}{<q>}  <Rd>, #<const>
```

Encoding for the MVNS variant

Applies when `(S == 1)`

```
MVNS{<c>}{<q>}  <Rd>, #<const>
```

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant boolean setflags = (S == '1');
constant boolean a32 = TRUE;
constant bits(12) imm = imm12;
```

T1

Encoding for the MVN variant

Applies when `(S == 0)`

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
MVN{<c>}{<q>}  <Rd>, #<const>
```

Encoding for the MVNS variant

Applies when `(S == 1)`

```
MVNS{<c>}{<q>}  <Rd>, #<const>
```

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant boolean setflags = (S == '1');
constant boolean a32 = FALSE;
constant bits(12) imm = i:imm3:imm8;
// Armv8-A removes UNPREDICTABLE for R13
if d == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

For the "A1 MVN" and "A1 MVNS" variants: is the general-purpose destination register, encoded in the "Rd" field. Arm deprecates using the PC as the destination register, but if the PC is used:

- For the MVN variant, the instruction is a branch to the address calculated by the operation. This is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- For the MVNS variant, the instruction performs an exception return, that restores PSTATE from SPSR_<current_mode>.

For the "T1 MVN" and "T1 MVNS" variants: is the general-purpose destination register, encoded in the "Rd" field.

<const>

For the "A1 MVN" and "A1 MVNS" variants: an immediate value. See Modified immediate constants in A32 instructions for the range of values.

For the "T1 MVN" and "T1 MVNS" variants: an immediate value. See Modified immediate constants in T32 instructions for the range of values.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    bits(32) imm32;
    bit carry;
    (imm32, carry) = if a32 then A32ExpandImm_C(imm, PSTATE.C) else T32ExpandImm_C(imm, PSTATE.C);
    constant bits(32) result = NOT(imm32);
    if d == 15 then          // Can only occur for A32 encoding
        if setflags then
            ALUExceptionReturn(result);
        else
            ALUWritePC(result);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

```
else
    R[d] = result;
    if setflags then
        PSTATE.N = result<31>;
        PSTATE.Z = IsZeroBit(result);
        PSTATE.C = carry;
        // PSTATE.V unchanged
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.125 MVN, MVNS (register)

Bitwise NOT (register) writes the bitwise inverse of a register value to the destination register.

If the destination register is not the PC, the MVNS variant of the instruction updates the condition flags based on the result.

The field descriptions for `<Rd>` identify the encodings where the PC is permitted as the destination register. ARM deprecates any use of these encodings. However, when the destination register is the PC:

- The MVN variant of the instruction is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- The MVNS variant of the instruction performs an exception return without the use of the stack. In this case:
  - The PE branches to the address written to the PC, and restores PSTATE from SPSR_<current_mode>.
  - The PE checks SPSR_<current_mode> for an illegal return event. See Illegal return events from AArch32 state.
  - The instruction is UNDEFINED in Hyp mode.
  - The instruction is CONSTRAINED UNPREDICTABLE in User mode and System mode.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1 and T2).

A1

31–28	27–23	22–21	20	19–16	15–12	11–7	6–5	4	3–0
!=1111	0 0 0 1 1	1 1	S	(0)(0)(0)(0)	Rd	imm5	stype	0	Rm
cond		opc		Rn					

Encoding for the MVN, rotate right with extend variant

Applies when `(S == 0 && imm5 == 00000 && stype == 11)`

```
MVN{<c>}{<q>}  <Rd>, <Rm>, RRX
```

Encoding for the MVN, shift or rotate by value variant

Applies when `(S == 0 && !(imm5 == 00000 && stype == 11))`

```
MVN{<c>}{<q>}  <Rd>, <Rm> {, <shift> #<amount>}
```

Encoding for the MVNS, rotate right with extend variant

Applies when `(S == 1 && imm5 == 00000 && stype == 11)`

```
MVNS{<c>}{<q>}  <Rd>, <Rm>, RRX
```

Encoding for the MVNS, shift or rotate by value variant

Applies when `(S == 1 && !(imm5 == 00000 && stype == 11))`

```
MVNS{<c>}{<q>}  <Rd>, <Rm> {, <shift> #<amount>}
```

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant integer m = UInt(Rm);
constant boolean setflags = (S == '1');
SRType shift_t;
integer shift_n;
(shift_t, shift_n) = DecodeImmShift(stype, imm5);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T1

Encoding

```
MVN<c>{<q>}  <Rd>, <Rm> // (InITBlock())
```

```
MVNS{<q>}  <Rd>, <Rm> // (Outside IT block)
```

Decode for this encoding

```
constant integer d = UInt(Rd);
constant integer m = UInt(Rm);
constant boolean setflags = !InITBlock();
constant SRType shift_t = SRType_LSL;
constant integer shift_n = 0;
```

T2

Encoding for the MVN, rotate right with extend variant

Applies when `(S == 0 && imm3 == 000 && imm2 == 00 && stype == 11)`

```
MVN{<c>}{<q>}  <Rd>, <Rm>, RRX
```

Encoding for the MVN, shift or rotate by value variant

Applies when `(S == 0 && !(imm3 == 000 && imm2 == 00 && stype == 11))`

```
MVN{<c>}{<q>}  <Rd>, <Rm> {, <shift> #<amount>}
```

```
MVN<c>.W  <Rd>, <Rm> // (Inside IT block, and <Rd>, <Rm> can be represented in T1)
```

Encoding for the MVNS, rotate right with extend variant

Applies when `(S == 1 && imm3 == 000 && imm2 == 00 && stype == 11)`

```
MVNS{<c>}{<q>}  <Rd>, <Rm>, RRX
```

Encoding for the MVNS, shift or rotate by value variant

Applies when `(S == 1 && !(imm3 == 000 && imm2 == 00 && stype == 11))`

```
MVNS{<c>}{<q>}  <Rd>, <Rm> {, <shift> #<amount>}
```

```
MVNS.W  <Rd>, <Rm> // (Outside IT block, and <Rd>, <Rm> can be represented in T1)
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant integer m = UInt(Rm);
constant boolean setflags = (S == '1');
SRType shift_t;
integer shift_n;
(shift_t, shift_n) = DecodeImmShift(stype, imm3:imm2);
// Armv8-A removes UNPREDICTABLE for R13
if d == 15 || m == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

For the “A1 MVN, rotate right with extend”, “A1 MVN, shift or rotate by value”, “A1 MVNS, rotate right with extend”, and “A1 MVNS, shift or rotate by value” variants: is the general-purpose destination register, encoded in the “Rd” field. Arm deprecates using the PC as the destination register, but if the PC is used:

- For the MVN variant, the instruction is a branch to the address calculated by the operation. This is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- For the MVNS variant, the instruction performs an exception return, that restores PSTATE from SPSR_<current_mode>.

For the “T1”, “T2 MVN, rotate right with extend”, “T2 MVN, shift or rotate by value”, “T2 MVNS, rotate right with extend”, and “T2 MVNS, shift or rotate by value” variants: is the general-purpose destination register, encoded in the “Rd” field.

<Rm>

For the “A1 MVN, rotate right with extend”, “A1 MVN, shift or rotate by value”, “A1 MVNS, rotate right with extend”, and “A1 MVNS, shift or rotate by value” variants: is the general-purpose source register, encoded in the “Rm” field. The PC can be used, but this is deprecated.

For the “T1”, “T2 MVN, rotate right with extend”, “T2 MVN, shift or rotate by value”, “T2 MVNS, rotate right with extend”, and “T2 MVNS, shift or rotate by value” variants: is the general-purpose source register, encoded in the “Rm” field.

<shift>

Is the type of shift to be applied to the source register, encoded in “stype”:

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

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

<amount>

For the “A1 MVN, shift or rotate by value” and “A1 MVNS, shift or rotate by value” variants: is the shift amount, in the range 1 to 31 (when <shift> = LSL or ROR) or 1 to 32 (when <shift> = LSR or ASR), encoded in the “imm5” field as <amount> modulo 32.

For the “T2 MVN, shift or rotate by value” and “T2 MVNS, shift or rotate by value” variants: is the shift amount, in the range 1 to 31 (when <shift> = LSL or ROR) or 1 to 32 (when <shift> = LSR or ASR), encoded in the “imm3:imm2” field as <amount> modulo 32.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    bits(32) shifted;
    bit carry;
    (shifted, carry) = Shift_C(R[m], shift_t, shift_n, PSTATE.C);
    constant bits(32) result = NOT(shifted);
    if d == 15 then          // Can only occur for A32 encoding
        if setflags then
            ALUExceptionReturn(result);
        else
            ALUWritePC(result);
    else
        R[d] = result;
        if setflags then
            PSTATE.N = result<31>;
            PSTATE.Z = IsZeroBit(result);
            PSTATE.C = carry;
            // PSTATE.V unchanged
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.126 MVN, MVNS (register-shifted register)

Bitwise NOT (register-shifted register) writes the bitwise inverse of a register-shifted register value to the destination register. It can optionally update the condition flags based on the result.

31–28	27–23	22–21	20	19–16	15–12	11–8	7	6–5	4	3–0
!=1111	0 0 0 1 1	1 1	S	(0) (0) (0) (0)	Rd	Rs	0	stype	1	Rm
cond		opc		Rn						

Encoding for the Flag setting variant

Applies when `(S == 1)`

```
MVNS{<c>}{<q>}  <Rd>, <Rm>, <shift> <Rs>
```

Encoding for the Not flag setting variant

Applies when `(S == 0)`

```
MVN{<c>}{<q>}  <Rd>, <Rm>, <shift> <Rs>
```

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant integer m = UInt(Rm);
constant integer s = UInt(Rs);
constant boolean setflags = (S == '1');
constant SRType shift_t = DecodeRegShift(stype);
if d == 15 || m == 15 || s == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

Is the general-purpose destination register, encoded in the “Rd” field.

<Rm>

Is the general-purpose source register, encoded in the “Rm” field.

<shift>

Is the type of shift to be applied to the second source register, encoded in “stype”:

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

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

<Rs>

Is the general-purpose source register holding a shift amount in its bottom 8 bits, encoded in the “Rs” field.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant integer shift_n = UInt(R[s]<7:0>);
    bits(32) shifted;
    bit carry;
    (shifted, carry) = Shift_C(R[m], shift_t, shift_n, PSTATE.C);
    constant bits(32) result = NOT(shifted);
    R[d] = result;
    if setflags then
        PSTATE.N = result<31>;
        PSTATE.Z = IsZeroBit(result);
        PSTATE.C = carry;
        // PSTATE.V unchanged
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.127 NOP

No Operation does nothing. This instruction can be used for instruction alignment purposes.

Note

The timing effects of including a `NOP` instruction in a program are not guaranteed. It can increase execution time, leave it unchanged, or even reduce it. Therefore, `NOP` instructions are not suitable for timing loops.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1 and T2).

A1

Encoding

```
NOP{<c>}{<q>}
```

Decode for this encoding

Listing F5-55

```
// No additional decoding required
```

T1

Encoding

```
NOP{<c>}{<q>}
```

Decode for this encoding

Listing F5-56

```
// No additional decoding required
```

T2

Encoding

```
NOP{<c>}.W
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for this encoding

Listing F5-57

```
// No additional decoding required
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

Operation

Listing F5-58

```
if ConditionPassed() then
    EncodingSpecificOperations();
    // Do nothing
end
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.128 ORN, ORNS (immediate)

Bitwise OR NOT (immediate) performs a bitwise (inclusive) OR of a register value and the complement of an immediate value, and writes the result to the destination register. It can optionally update the condition flags based on the result.

15–11	10	9	8–5	4	3–0	15	14–12	11–8	7–0
1 1 1 1 0	i	0	0 0 1 1 (op1)	S	!=1111 (Rn)	0	imm3	Rd	imm8

Encoding for the Flag setting variant

Applies when `(S == 1)`

```
ORNS{<c>}{<q>}  {<Rd>, }<Rn>, #<const>
```

Encoding for the Not flag setting variant

Applies when `(S == 0)`

```
ORN{<c>}{<q>}  {<Rd>, }<Rn>, #<const>
```

Decode for all variants of this encoding

```
if Rn == '1111' then SEE "MVN (immediate)";
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant boolean setflags = (S == '1');
constant boolean a32 = FALSE;
constant bits(12) imm = i:imm3:imm8;
// Armv8-A removes UNPREDICTABLE for R13
if d == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

Is the general-purpose destination register, encoded in the “Rd” field. If omitted, this register is the same as `<Rn>`.

<Rn>

Is the general-purpose source register, encoded in the “Rn” field.

<const>

An immediate value. See Modified immediate constants in T32 instructions for the range of values.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    bits(32) imm32;
    bit carry;
    (imm32, carry) = if a32 then A32ExpandImm_C(imm, PSTATE.C) else T32ExpandImm_C(imm, PSTATE.C);
    constant bits(32) result = R[n] OR NOT(imm32);
    R[d] = result;
    if setflags then
        PSTATE.N = result<31>;
        PSTATE.Z = IsZeroBit(result);
        PSTATE.C = carry;
        // PSTATE.V unchanged
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.129 ORN, ORNS (register)

Bitwise OR NOT (register) performs a bitwise (inclusive) OR of a register value and the complement of an optionally-shifted register value, and writes the result to the destination register. It can optionally update the condition flags based on the result.

15–9	8–5	4	3–0	15	14–12	11–8	7–6	5–4	3–0
1 1 1 0 1 0 1	0 0 1 1 (op1)	S	!=1111 (Rn)	(0)	imm3	Rd	imm2	stype	Rm

Encoding for the ORN, rotate right with extend variant

Applies when `(S == 0 && imm3 == 000 && imm2 == 00 && stype == 11)`

```
ORN{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, RRX
```

Encoding for the ORN, shift or rotate by value variant

Applies when `(S == 0 && !(imm3 == 000 && imm2 == 00 && stype == 11))`

```
ORN{<c>}{<q>}  {<Rd>, }<Rn>, <Rm> {, <shift> #<amount>}
```

Encoding for the ORNS, rotate right with extend variant

Applies when `(S == 1 && imm3 == 000 && imm2 == 00 && stype == 11)`

```
ORNS{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, RRX
```

Encoding for the ORNS, shift or rotate by value variant

Applies when `(S == 1 && !(imm3 == 000 && imm2 == 00 && stype == 11))`

```
ORNS{<c>}{<q>}  {<Rd>, }<Rn>, <Rm> {, <shift> #<amount>}
```

Decode for all variants of this encoding

```
if Rn == '1111' then SEE "MVN (register)";
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean setflags = (S == '1');
SRType shift_t;
integer shift_n;
(shift_t, shift_n) = DecodeImmShift(stype, imm3:imm2);
// Armv8-A removes UNPREDICTABLE for R13
if d == 15 || m == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

Is the general-purpose destination register, encoded in the “Rd” field.

<Rn>

Is the first general-purpose source register, encoded in the “Rn” field.

<Rm>

Is the second general-purpose source register, encoded in the “Rm” field.

<shift>

Is the type of shift to be applied to the second source register, encoded in “stype”:

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

<amount>

Is the shift amount, in the range 1 to 31 (when <shift> = LSL or ROR) or 1 to 32 (when <shift> = LSR or ASR), encoded in the “imm3:imm2” field as <amount> modulo 32.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    bits(32) shifted;
    bit carry;
    (shifted, carry) = Shift_C(R[m], shift_t, shift_n, PSTATE.C);
    constant bits(32) result = R[n] OR NOT(shifted);
    R[d] = result;
    if setflags then
        PSTATE.N = result<31>;
        PSTATE.Z = IsZeroBit(result);
        PSTATE.C = carry;
        // PSTATE.V unchanged
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.130 ORR, ORRS (immediate)

Bitwise OR (immediate) performs a bitwise (inclusive) OR of a register value and an immediate value, and writes the result to the destination register.

If the destination register is not the PC, the ORRS variant of the instruction updates the condition flags based on the result.

The field descriptions for `<Rd>` identify the encodings where the PC is permitted as the destination register. ARM deprecates any use of these encodings. However, when the destination register is the PC:

- The ORR variant of the instruction is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- The ORRS variant of the instruction performs an exception return without the use of the stack. In this case:
  - The PE branches to the address written to the PC, and restores PSTATE from SPSR_<current_mode>.
  - The PE checks SPSR_<current_mode> for an illegal return event. See Illegal return events from AArch32 state.
  - The instruction is UNDEFINED in Hyp mode.
  - The instruction is CONSTRAINED UNPREDICTABLE in User mode and System mode.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding for the ORR variant

Applies when `(S == 0)`

```
ORR{<c>}{<q>}  {<Rd>, }<Rn>, #<const>
```

Encoding for the ORRS variant

Applies when `(S == 1)`

```
ORRS{<c>}{<q>}  {<Rd>, }<Rn>, #<const>
```

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant boolean setflags = (S == '1');
constant boolean a32 = TRUE;
constant bits(12) imm = imm12;
```

T1

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Encoding for the ORR variant

Applies when `(S == 0)`

```
ORR{<c>}{<q>}  {<Rd>, }<Rn>, #<const>
```

Encoding for the ORRS variant

Applies when `(S == 1)`

```
ORRS{<c>}{<q>}  {<Rd>, }<Rn>, #<const>
```

Decode for all variants of this encoding

```
if Rn == '1111' then SEE "MOV (immediate)";
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant boolean setflags = (S == '1');
constant boolean a32 = FALSE;
constant bits(12) imm = i:imm3:imm8;
// Armv8-A removes UNPREDICTABLE for R13
if d == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

For the “A1 ORR” and “A1 ORRS” variants: is the general-purpose destination register, encoded in the “Rd” field. If omitted, this register is the same as `<Rn>`. Arm deprecates using the PC as the destination register, but if the PC is used:

- For the ORR variant, the instruction is a branch to the address calculated by the operation. This is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- For the ORRS variant, the instruction performs an exception return, that restores PSTATE from SPSR_<current_mode>.

For the “T1 ORR” and “T1 ORRS” variants: is the general-purpose destination register, encoded in the “Rd” field. If omitted, this register is the same as `<Rn>`.

<Rn>

For the “A1 ORR” and “A1 ORRS” variants: is the general-purpose source register, encoded in the “Rn” field. The PC can be used, but this is deprecated.

For the “T1 ORR” and “T1 ORRS” variants: is the general-purpose source register, encoded in the “Rn” field.

<const>

For the “A1 ORR” and “A1 ORRS” variants: an immediate value. See Modified immediate constants in A32 instructions for the range of values.

For the “T1 ORR” and “T1 ORRS” variants: an immediate value. See Modified immediate constants in T32 instructions for the range of values.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    bits(32) imm32;
    bit carry;
    (imm32, carry) = if a32 then A32ExpandImm_C(imm, PSTATE.C) else T32ExpandImm_C(imm, PSTATE.C);
    constant bits(32) result = R[n] OR imm32;
    if d == 15 then          // Can only occur for A32 encoding
        if setflags then
            ALUExceptionReturn(result);
        else
            ALUWritePC(result);
    else
        R[d] = result;
        if setflags then
            PSTATE.N = result<31>;
            PSTATE.Z = IsZeroBit(result);
            PSTATE.C = carry;
            // PSTATE.V unchanged
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.131 ORR, ORRS (register)

Bitwise OR (register) performs a bitwise (inclusive) OR of a register value and an optionally-shifted register value, and writes the result to the destination register.

If the destination register is not the PC, the ORRS variant of the instruction updates the condition flags based on the result.

The field descriptions for `<Rd>` identify the encodings where the PC is permitted as the destination register. ARM deprecates any use of these encodings. However, when the destination register is the PC:

- The ORR variant of the instruction is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- The ORRS variant of the instruction performs an exception return without the use of the stack. In this case:
  - The PE branches to the address written to the PC, and restores PSTATE from SPSR_<current_mode>.
  - The PE checks SPSR_<current_mode> for an illegal return event. See Illegal return events from AArch32 state.
  - The instruction is UNDEFINED in Hyp mode.
  - The instruction is CONSTRAINED UNPREDICTABLE in User mode and System mode.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1 and T2).

A1

31–28	27–23	22–21	20	19–16	15–12	11–7	6–5	4	3–0
!=1111 (cond)	0 0 0 1 1	0 0 (opc)	S	Rn	Rd	imm5	stype	0	Rm

Encoding for the ORR, rotate right with extend variant

Applies when `(S == 0 && imm5 == 00000 && stype == 11)`

```
ORR{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, RRX
```

Encoding for the ORR, shift or rotate by value variant

Applies when `(S == 0 && !(imm5 == 00000 && stype == 11))`

```
ORR{<c>}{<q>}  {<Rd>, }<Rn>, <Rm> {, <shift> #<amount>}
```

Encoding for the ORRS, rotate right with extend variant

Applies when `(S == 1 && imm5 == 00000 && stype == 11)`

```
ORRS{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, RRX
```

Encoding for the ORRS, shift or rotate by value variant

Applies when `(S == 1 && !(imm5 == 00000 && stype == 11))`

```
ORRS{<c>}{<q>}  {<Rd>, }<Rn>, <Rm> {, <shift> #<amount>}
```

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean setflags = (S == '1');
SRType shift_t;
integer shift_n;
(shift_t, shift_n) = DecodeImmShift(stype, imm5);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

T1

15–10	9–6	5–3	2–0
0 1 0 0 0 0	1 1 0 0	Rm	Rdn
	op		

Encoding

```
ORR<c>{<q>}  {<Rdn>, }<Rdn>, <Rm> // (InITBlock())
```

```
ORRS{<q>}  {<Rdn>, }<Rdn>, <Rm> // (Outside IT block)
```

Decode for this encoding

```
constant integer d = UInt(Rdn);
constant integer n = UInt(Rdn);
constant integer m = UInt(Rm);
constant boolean setflags = !InITBlock();
constant SRType shift_t = SRType_LSL;
constant integer shift_n = 0;
```

T2

15–9	8–5	4	3–0	15	14–12	11–8	7–6	5–4	3–0
1 1 1 0 1 0 1	0 0 1 0	S	!=1111	(0)	imm3	Rd	imm2	stype	Rm
	op1		Rn						

Encoding for the ORR, rotate right with extend variant

Applies when `(S == 0 && imm3 == 000 && imm2 == 00 && stype == 11)`

```
ORR{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, RRX
```

Encoding for the ORR, shift or rotate by value variant

Applies when `(S == 0 && !(imm3 == 000 && imm2 == 00 && stype == 11))`

```
ORR{<c>}{<q>}  {<Rd>, }<Rn>, <Rm> {, <shift> #<amount>}
```

```
ORR<c>.W  {<Rd>, }<Rn>, <Rm> // (Inside IT block, and <Rd>, <Rn>, <Rm> can be represented in T1)
```

Encoding for the ORRS, rotate right with extend variant

Applies when `(S == 1 && imm3 == 000 && imm2 == 00 && stype == 11)`

```
ORRS{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, RRX
```

Encoding for the ORRS, shift or rotate by value variant

Applies when `(S == 1 && !(imm3 == 000 && imm2 == 00 && stype == 11))`

```
ORRS{<c>}{<q>}  {<Rd>, }<Rn>, <Rm> {, <shift> #<amount>}
```

```
ORRS.W  {<Rd>, }<Rn>, <Rm> // (Outside IT block, and <Rd>, <Rn>, <Rm> can be represented in T1)
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for all variants of this encoding

```
if Rn == '1111' then SEE "Related encodings";
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean setflags = (S == '1');
SRType shift_t;
integer shift_n;
(shift_t, shift_n) = DecodeImmShift(stype, imm3:imm2);
// Armv8-A removes UNPREDICTABLE for R13
if d == 15 || m == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Related encodings: Data-processing (shifted register)

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

For the “A1 ORR, rotate right with extend”, “A1 ORR, shift or rotate by value”, “A1 ORRS, rotate right with extend”, and “A1 ORRS, shift or rotate by value” variants: is the general-purpose destination register, encoded in the “Rd” field. If omitted, this register is the same as `<Rn>`. Arm deprecates using the PC as the destination register, but if the PC is used:

- For the ORR variant, the instruction is a branch to the address calculated by the operation. This is an interworking branch, see Pseudocode description of operations on the AArch32 general-purpose registers and the PC.
- For the ORRS variant, the instruction performs an exception return, that restores PSTATE from SPSR_<current_mode>.

For the “T2 ORR, rotate right with extend”, “T2 ORR, shift or rotate by value”, “T2 ORRS, rotate right with extend”, and “T2 ORRS, shift or rotate by value” variants: is the general-purpose destination register, encoded in the “Rd” field. If omitted, this register is the same as `<Rn>`.

<Rn>

For the “A1 ORR, rotate right with extend”, “A1 ORR, shift or rotate by value”, “A1 ORRS, rotate right with extend”, and “A1 ORRS, shift or rotate by value” variants: is the first general-purpose source register, encoded in the “Rn” field. The PC can be used, but this is deprecated.

For the “T2 ORR, rotate right with extend”, “T2 ORR, shift or rotate by value”, “T2 ORRS, rotate right with extend”, and “T2 ORRS, shift or rotate by value” variants: is the first general-purpose source register, encoded in the “Rn” field.

<Rm>

For the “A1 ORR, rotate right with extend”, “A1 ORR, shift or rotate by value”, “A1 ORRS, rotate right with extend”, and “A1 ORRS, shift or rotate by value” variants: is the second general-purpose source register, encoded in the “Rm” field. The PC can be used, but this is deprecated.

For the “T1”, “T2 ORR, rotate right with extend”, “T2 ORR, shift or rotate by value”, “T2 ORRS, rotate right with extend”, and “T2 ORRS, shift or rotate by value” variants: is the second general-purpose source register, encoded in the “Rm” field.

<shift>

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Is the type of shift to be applied to the second source register, encoded in “stype”:

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

<amount>

For the “A1 ORR, shift or rotate by value” and “A1 ORRS, shift or rotate by value” variants: is the shift amount, in the range 1 to 31 (when <shift> = LSL or ROR) or 1 to 32 (when <shift> = LSR or ASR), encoded in the “imm5” field as <amount> modulo 32.

For the “T2 ORR, shift or rotate by value” and “T2 ORRS, shift or rotate by value” variants: is the shift amount, in the range 1 to 31 (when <shift> = LSL or ROR) or 1 to 32 (when <shift> = LSR or ASR), encoded in the “imm3:imm2” field as <amount> modulo 32.

<Rdn>

Is the first general-purpose source register and the destination register, encoded in the “Rdn” field.

In T32 assembly:

- Outside an IT block, if `ORRS <Rd>, <Rn>, <Rd>` is written with `<Rd>` and `<Rn>` both in the range R0-R7, it is assembled using encoding T1 as though `ORRS <Rd>, <Rn>` had been written.
- Inside an IT block, if `ORR<c> <Rd>, <Rn>, <Rd>` is written with `<Rd>` and `<Rn>` both in the range R0-R7, it is assembled using encoding T1 as though `ORR<c> <Rd>, <Rn>` had been written.

To prevent either of these happening, use the `.W` qualifier.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    bits(32) shifted;
    bit carry;
    (shifted, carry) = Shift_C(R[m], shift_t, shift_n, PSTATE.C);
    constant bits(32) result = R[n] OR shifted;
    if d == 15 then          // Can only occur for A32 encoding
        if setflags then
            ALUExceptionReturn(result);
        else
            ALUWritePC(result);
    else
        R[d] = result;
        if setflags then
            PSTATE.N = result<31>;
            PSTATE.Z = IsZeroBit(result);
            PSTATE.C = carry;
            // PSTATE.V unchanged
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.132 ORR, ORRS (register-shifted register)

Bitwise OR (register-shifted register) performs a bitwise (inclusive) OR of a register value and a register-shifted register value, and writes the result to the destination register. It can optionally update the condition flags based on the result.

31–28	27–23	22–21	20	19–16	15–12	11–8	7	6–5	4	3–0
!=1111 (cond)	0 0 0 1 1	0 0 (opc)	S	Rn	Rd	Rs	0	stype	1	Rm

Encoding for the Flag setting variant

Applies when `(S == 1)`

```
ORRS{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, <shift> <Rs>
```

Encoding for the Not flag setting variant

Applies when `(S == 0)`

```
ORR{<c>}{<q>}  {<Rd>, }<Rn>, <Rm>, <shift> <Rs>
```

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant integer s = UInt(Rs);
constant boolean setflags = (S == '1');
constant SRType shift_t = DecodeRegShift(stype);
if d == 15 || n == 15 || m == 15 || s == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

Is the general-purpose destination register, encoded in the “Rd” field.

<Rn>

Is the first general-purpose source register, encoded in the “Rn” field.

<Rm>

Is the second general-purpose source register, encoded in the “Rm” field.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

<shift>

Is the type of shift to be applied to the second source register, encoded in “stype”:

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

<Rs>

Is the general-purpose source register holding a shift amount in its bottom 8 bits, encoded in the “Rs” field.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant integer shift_n = UInt(R[s]<7:0>);
    bits(32) shifted;
    bit carry;
    (shifted, carry) = Shift_C(R[m], shift_t, shift_n, PSTATE.C);
    constant bits(32) result = R[n] OR shifted;
    R[d] = result;
    if setflags then
        PSTATE.N = result<31>;
        PSTATE.Z = IsZeroBit(result);
        PSTATE.C = carry;
        // PSTATE.V unchanged
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.133 PKHBT, PKHTB

Pack Halfword combines one halfword of its first operand with the other halfword of its shifted second operand.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding for the PKHBT variant

Applies when `(tb == 0)`

```
PKHBT{<c>}{<q>}  {<Rd>, }<Rn>, <Rm> {, LSL #<imm>}
```

Encoding for the PKHTB variant

Applies when `(tb == 1)`

```
PKHTB{<c>}{<q>}  {<Rd>, }<Rn>, <Rm> {, ASR #<imm>}
```

Decode for all variants of this encoding

```
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean tbform = (tb == '1');
SRType shift_t;
integer shift_n;
(shift_t, shift_n) = DecodeImmShift(tb:'0', imm5);
if d == 15 || n == 15 || m == 15 then UNPREDICTABLE;
```

T1

Encoding for the PKHBT variant

Applies when `(tb == 0)`

```
PKHBT{<c>}{<q>}  {<Rd>, }<Rn>, <Rm> {, LSL #<imm>} // (tbform == FALSE)
```

Encoding for the PKHTB variant

Applies when `(tb == 1)`

```
PKHTB{<c>}{<q>}  {<Rd>, }<Rn>, <Rm> {, ASR #<imm>} // (tbform == TRUE)
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for all variants of this encoding

```
if S == '1' || T == '1' then UNDEFINED;
constant integer d = UInt(Rd);
constant integer n = UInt(Rn);
constant integer m = UInt(Rm);
constant boolean tbform = (tb == '1');
SRType shift_t;
integer shift_n;
(shift_t, shift_n) = DecodeImmShift(tb:'0', imm3:imm2);
// Armv8-A removes UNPREDICTABLE for R13
if d == 15 || n == 15 || m == 15 then UNPREDICTABLE;
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

See Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rd>

Is the general-purpose destination register, encoded in the “Rd” field.

<Rn>

Is the first general-purpose source register, encoded in the “Rn” field.

<Rm>

Is the second general-purpose source register, encoded in the “Rm” field.

<imm>

For the “A1 PKHBT” and “A1 PKHTB” variants: the shift to apply to the value read from `<Rm>`, encoded in the “imm5” field.

For `PKHBT`, it is one of:

omitted No shift, encoded as `0b00000`.
1-31 Left shift by specified number of bits, encoded as a binary number.

For `PKHTB`, it is one of:

omitted Instruction is a pseudo-instruction and is assembled as though `PKHBT{<c>}{<q>} <Rd>, <Rm>, <Rn>` had been written.
1-32 Arithmetic right shift by specified number of bits. A shift by 32 bits is encoded as `0b00000`. Other shift amounts are encoded as binary numbers.

Note

An assembler can permit `<imm>` = 0 to mean the same thing as omitting the shift, but this is not standard UAL and must not be used for disassembly.

For the “T1 PKHBT” and “T1 PKHTB” variants: the shift to apply to the value read from `<Rm>`, encoded in the “imm3:imm2” field.

For `PKHBT`, it is one of:

omitted No shift, encoded as `0b00000`.
1-31 Left shift by specified number of bits, encoded as a binary number.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

For PKHTB, it is one of:

omitted Instruction is a pseudo-instruction and is assembled as though PKHBT{<c>}{<q>} <Rd>, <Rm>, <Rn> had been written.

1-32 Arithmetic right shift by specified number of bits. A shift by 32 bits is encoded as 0b00000. Other shift amounts are encoded as binary numbers.

Note

An assembler can permit <imm> = 0 to mean the same thing as omitting the shift, but this is not standard UAL and must not be used for disassembly.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) operand2 = Shift(R[m], shift_t, shift_n, PSTATE.C);  // PSTATE.C ignored
    R[d]<15:0>  = if tbform then operand2<15:0> else R[n]<15:0>;
    R[d]<31:16> = if tbform then R[n]<31:16>    else operand2<31:16>;
```

Operational Information

This instruction is a data-independent-time instruction as described in About the DIT bit.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.134 PLD, PLDW (immediate)

Preload Data (immediate) signals the memory system that data memory accesses from a specified address are likely in the near future. The memory system can respond by taking actions that are expected to speed up the memory accesses when they do occur, such as preloading the cache line containing the specified address into the data cache.

The `PLD` instruction signals that the likely memory access is a read, and the `PLDW` instruction signals that it is a write.

The effect of a `PLD` or `PLDW` instruction is IMPLEMENTATION DEFINED. For more information, see Preloading caches.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1 and T2).

A1

Encoding for the Preload read variant

Applies when `(R == 1)`

```
PLD{<c>}{<q>}  [<Rn> {, #{+/-}<imm>}]
```

Encoding for the Preload write variant

Applies when `(R == 0)`

```
PLDW{<c>}{<q>}  [<Rn> {, #{+/-}<imm>}]
```

Decode for all variants of this encoding

```
if Rn == '1111' then SEE "PLD (literal)";
constant integer n = UInt(Rn);
constant bits(32) imm32 = ZeroExtend(imm12, 32);
constant boolean add = (U == '1');
constant boolean is_pldw = (R == '0');
```

T1

Encoding for the Preload read variant

Applies when `(W == 0)`

```
PLD{<c>}{<q>}  [<Rn> {, #{+}<imm>}]
```

Encoding for the Preload write variant

Applies when `(W == 1)`

```
PLDW{<c>}{<q>}  [<Rn> {, #{+}<imm>}]
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Decode for all variants of this encoding

```
if Rn == '1111' then SEE "PLD (literal)";
constant integer n = UInt(Rn);
constant bits(32) imm32 = ZeroExtend(imm12, 32);
constant boolean add = TRUE;
constant boolean is_pldw = (W == '1');
```

T2

15									6	5	4	3		0	15			12	11	10	9	8	7		0
1	1	1	1	1	0	0	0	0	0	W	1		!=1111		1	1	1	1	1	1	0	0		imm8	
										L			Rn				Rt		P		U				

Encoding for the Preload read variant

Applies when `(W == 0)`

```
PLD{<c>}{<q>}  [<Rn> {, #-<imm>}]
```

Encoding for the Preload write variant

Applies when `(W == 1)`

```
PLDW{<c>}{<q>}  [<Rn> {, #-<imm>}]
```

Decode for all variants of this encoding

```
if Rn == '1111' then SEE "PLD (literal)";
constant integer n = UInt(Rn);
constant bits(32) imm32 = ZeroExtend(imm8, 32);
constant boolean add = FALSE;
constant boolean is_pldw = (W == '1');
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Assembler Symbols

<c>

For the “A1 Preload read” and “A1 Preload write” variants: see Standard assembler syntax fields. Must be `AL` or omitted.

For the “T1 Preload read”, “T1 Preload write”, “T2 Preload read”, and “T2 Preload write” variants: see Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<Rn>

Is the general-purpose base register, encoded in the “Rn” field. If the PC is used, see PLD (literal).

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

+/-

Specifies the offset is added to or subtracted from the base register, defaulting to + if omitted and encoded in “U”:

U	+/-
0	-
1	+

<imm>

For the “A1 Preload read” and “A1 Preload write” variants: is the optional 12-bit unsigned immediate byte offset, in the range 0 to 4095, defaulting to 0 and encoded in the “imm12” field.

For the “T1 Preload read” and “T1 Preload write” variants: is an optional 12-bit unsigned immediate byte offset, in the range 0 to 4095, defaulting to 0 and encoded in the “imm12” field.

For the “T2 Preload read” and “T2 Preload write” variants: is an 8-bit unsigned immediate byte offset, in the range 0 to 255, defaulting to 0 if omitted, and encoded in the “imm8” field.

+

Specifies the offset is added to the base register.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) address = if add then (R[n] + imm32) else (R[n] - imm32);
    if is_pldw then
        Hint_PreloadDataForWrite(address);
    else
        Hint_PreloadData(address);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

F5.1.135 PLD (literal)

Preload Data (literal) signals the memory system that data memory accesses from a specified address are likely in the near future. The memory system can respond by taking actions that are expected to speed up the memory accesses when they do occur, such as preloading the cache line containing the specified address into the data cache.

The effect of a PLD instruction is IMPLEMENTATION DEFINED. For more information, see Preloading caches.

It has encodings from the following instruction sets: A32 (A1) and T32 (T1).

A1

Encoding

```
PLD{<c>}{<q>}  <label> // (Normal form)
```

```
PLD{<c>}{<q>}  [PC, #{+/-}<imm>] // (Alternative form)
```

Decode for this encoding

```
constant bits(32) imm32 = ZeroExtend(imm12, 32);
constant boolean add = (U == '1');
```

T1

Encoding

```
PLD{<c>}{<q>}  <label> // (Preferred syntax)
```

```
PLD{<c>}{<q>}  [PC, #{+/-}<imm>] // (Alternative syntax)
```

Decode for this encoding

```
constant bits(32) imm32 = ZeroExtend(imm12, 32);
constant boolean add = (U == '1');
```

For more information about the CONSTRAINED UNPREDICTABLE behavior of this instruction, see Architectural Constraints on UNPREDICTABLE behaviors.

Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential

Assembler Symbols

<c>

For the "A1" variant: see Standard assembler syntax fields. Must be `AL` or omitted.

For the "T1" variant: see Standard assembler syntax fields.

<q>

See Standard assembler syntax fields.

<label>

The label of the literal data item that is likely to be accessed in the near future. The assembler calculates the required value of the offset from the `Align(PC, 4)` value of the instruction to this label. The offset must be in the range –4095 to 4095.

If the offset is zero or positive, `imm32` is equal to the offset and `add == TRUE`.

If the offset is negative, `imm32` is equal to minus the offset and `add == FALSE`.

+/-

Specifies the offset is added to or subtracted from the base register, defaulting to + if omitted and encoded in "U":

U	+/-
0	-
1	+

<imm>

For the "A1" variant: is the 12-bit unsigned immediate byte offset, in the range 0 to 4095, encoded in the "imm12" field.

For the "T1" variant: is a 12-bit unsigned immediate byte offset, in the range 0 to 4095, encoded in the "imm12" field.

The alternative syntax permits the addition or subtraction of the offset and the immediate offset to be specified separately, including permitting a subtraction of 0 that cannot be specified using the normal syntax. For more information, see Use of labels in UAL instruction syntax.

Operation

```
if ConditionPassed() then
    EncodingSpecificOperations();
    constant bits(32) address = (if add then (Align(PC32,4) + imm32)
                                 else (Align(PC32,4) - imm32));
    Hint_PreloadData(address);
```


Copyright © 2013-2025 Arm Limited or its affiliates. All rights reserved.
Non-Confidential