CPP RCTX, Cache Prefetch Prediction Restriction by Context

The CPP RCTX characteristics are:

Purpose

Cache Prefetch Prediction Restriction by Context applies to all Cache Allocation Resources that predict cache allocations based on information gathered within the target execution context or contexts.

The actions of code in the target execution context or contexts appearing in program order before the instruction cannot exploitatively control cache prefetch predictions occurring after the instruction is complete and synchronized.

This instruction applies to all:

- Instruction caches.
- Data caches.
- TLB prefetching hardware used by the executing PE that applies to the supplied context or contexts.

This instruction is guaranteed to be complete following a DSB that covers both read and write behavior on the same PE as executed the original restriction instruction, and a subsequent context synchronization event is required to ensure that the effect of the completion of the instructions is synchronized to the current execution.

Note

This instruction does not require the invalidation of Cache Allocation Resources so long as the behavior described for completion of this instruction is met by the implementation.

On some implementations the instruction is likely to take a significant number of cycles to execute. This instruction is expected to be used very rarely, such as on the roll-over of an ASID or VMID, but should not be used on every context switch.

Configuration

This instruction is present only when FEAT_SPECRES is implemented. Otherwise, direct accesses to CPP RCTX are undefined.

Attributes

CPP RCTX is a 64-bit System instruction.

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
RES0	GVMID	VMID

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
RES0	NSE	NS	EL	RES0	GASID	ASID

Bits [63:49]

Reserved, res0.

GVMID, bit [48]

Execution of this instruction applies to all VMIDs or a specified VMID.

GVMID	Meaning
`0b0`	Applies to specified VMID for an EL0 or EL1 target execution context.
`0b1`	Applies to all VMIDs for an EL0 or EL1 target execution context.

For target execution contexts other than EL0 and EL1, this field is res0.

If the instruction is executed at EL0 or EL1, this field has an Effective value of 0.

If EL2 is not implemented or not enabled for the target Security state, this field is res0.

VMID, bits [47:32]

Only applies when bit[48] is 0 and the target execution context is either:

- EL1.
- EL0 when (HCR_EL2.E2H==0 or HCR_EL2.TGE==0).

Otherwise this field is res0.

When the instruction is executed at EL1, this field is treated as the current VMID.

When the instruction is executed at EL0 and (HCR_EL2.E2H==0 or HCR_EL2.TGE==0), this field is treated as the current VMID.

When the instruction is executed at EL0 and (HCR_EL2.E2H==1 and HCR_EL2.TGE==1), this field is ignored.

If EL2 is not implemented or not enabled for the target Security state, this field is res0.

If the implementation supports 16 bits of VMID, then the upper 8 bits of the VMID must be written to 0 by software when the context being affected only uses 8 bits.

Bits [31:28]

Reserved, res0.

NSE, bit [27]
When FEAT_RME is implemented:

Together with the NS field, selects the Security state.

For a description of the values derived by evaluating NS and NSE together, see CPP_RCTX.NS.

Otherwise:

Reserved, res0.

NS, bit [26]
When FEAT_RME is implemented:

Together with the NSE field, selects the Security state. Defined values are:

NSE	NS	Meaning
`0b0`	`0b0`	When Secure state is implemented, Secure. Otherwise reserved.
`0b0`	`0b1`	Non-secure.
`0b1`	`0b0`	Root.
`0b1`	`0b1`	Realm.

Some Effective values are determined by the current Security state:

- When executed in Secure state, the Effective value of NSE is 0.
- When executed in Non-secure state, the Effective value of {NSE, NS} is {0, 1}.

- When executed in Realm state, the Effective value of {NSE, NS} is {1, 1}.

This instruction is treated as a NOP when executed at EL3 and either:

- CPP_RCTX.{NSE, NS} selects a reserved value.
- CPP_RCTX.{NSE, NS} == {1, 0} and CPP_RCTX.EL has a value other than `0b11`.

Otherwise:

Security State. Defined values are:

NS	Meaning
`0b0`	Secure state.
`0b1`	Non-secure state.

When executed in Non-secure state, the Effective value of NS is 1.

EL, bits [25:24]

Exception Level. Indicates the Exception level of the target execution context.

EL	Meaning
`0b00`	EL0.
`0b01`	EL1.
`0b10`	EL2.
`0b11`	EL3.

If the instruction is executed at an Exception level lower than the specified level, this instruction is treated as a NOP.

Bits [23:17]

Reserved, res0.

GASID, bit [16]

Execution of this instruction applies to all ASIDs or a specified ASID.

GASID	Meaning
`0b0`	Applies to specified ASID for an EL0 target execution context.
`0b1`	Applies to all ASIDs for an EL0 target execution context.

For target execution contexts other than EL0, this field is res0.

If the instruction is executed at EL0, this field has an Effective value of 0.

ASID, bits [15:0]

Only applies for an EL0 target execution context and when bit[16] is 0.

Otherwise, this field is res0.

When the instruction is executed at EL0, this field is treated as the current ASID.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being affected only uses 8 bits.

Executing CPP RCTX

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

```
CPP RCTX, <Xt>
```

op0	op1	CRn	CRm	op2
0b01	0b011	0b0111	0b0011	0b111

```
if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11')
&& SCTLR_EL1.EnRCTX == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11'
&& IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3)
|| SCR_EL3.FGTEn == '1') && HFGITR_EL2.CPPRCTX ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11'
&& SCTLR_EL2.EnRCTX == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.RestrictPrediction(X[t, 64],
RestrictType_CachePrefetch);
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() &&
IsFeatureImplemented(FEAT_FGT) && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGITR_EL2.CPPRCTX == '1'
```

```
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.RestrictPrediction(X[t, 64],
RestrictType_CachePrefetch);
elsif PSTATE.EL == EL2 then
    AArch64.RestrictPrediction(X[t, 64],
RestrictType_CachePrefetch);
elsif PSTATE.EL == EL3 then
    AArch64.RestrictPrediction(X[t, 64],
RestrictType_CachePrefetch);
```