ARM Architecture Reference Manual

Security Extensions Supplement

ARM DDI 0309F

ARM Architecture Reference Manual

Release Information

The following changes have been made to this document.

Change History

Date	Issue	Change
July 2004	A	Internal release only
July 2004	B	First release
December 2004	C	Second release
April 2005	E	Updated to incorporate corrections to errata
November 2005	F	Updated to incorporate corrections to errata. Non-confidential.

Proprietary Notice

4. No licence, express, implied or otherwise, is granted to LICENSEE, under the provisions of Clause 1, to use the ARM tradename, in connection with the use of the ARM Architecture Reference Manual or any products based thereon. Nothing in Clause 1 shall be construed as authority for you to make any representations on behalf of ARM in respect of the ARM Architecture Reference Manual or any products based thereon.

110 Fulbourn Road Cambridge, England CB1 9NJ

Contents
ARM Architecture Reference Manual Security Extensions Supplement

Preface

This preface introduces the *ARM Architecture Reference Manual, Security Extensions supplement*. It contains the following sections:

About this manual

The purpose of this manual is to describe the Security Extensions to the ARM® architecture. It is a supplement to the *ARM Architecture Reference Manual* (ARM DDI 0100), version F or later, and is intended to be used with it.

It is assumed that the reader is familiar with the ARM Programmers' Model (described in *ARM Architecture Reference Manual Part A*, chapter 2), and the memory system architecture support (described in *ARM Architecture Reference Manual Part B*). The System Coprocessor, Virtual Memory System Architecture, and Cache chapters are particularly relevant.

Intended audience

This book is written for all developers designing:

- ARM processors with Security Extensions
- hardware using ARM processors with Security Extensions
- software for systems using ARM processors with Security Extensions.

Using this manual

This manual is organized into the following chapters:

Chapter 1 ***Introduction***

Read this chapter for an introduction to the purpose of the ARM Architecture Security Extensions, and an overview of how this purpose is achieved.

It also describes the relationship of Security Extensions with the memory system, memory management, exception handling and system debug.

Chapter 2 ***CPU Architecture***

Read this chapter for details of the changes to the Programmer's Model and instruction set.

Chapter 3 ***Memory and System Architecture***

Read this chapter for details of the features added to the ARM Memory and System architecture as part of Security Extensions.

Chapter 4 ***Debug Architecture***

Read this chapter for details of the features added to the ARM Debug architecture as part of Security Extensions.

Conventions

This manual employs typographic and other conventions intended to improve its ease of use.

General typographic conventions

`typewriter` — Is used for assembler syntax descriptions, pseudo-code descriptions of instructions, and source code examples. In the cases of assembler syntax descriptions and pseudo-code descriptions, see the additional conventions below.

The `typewriter` font is also used in the main text for instruction mnemonics and for references to other items appearing in assembler syntax descriptions, pseudo-code descriptions of instructions and source code examples.

italic — Highlights important notes, introduces special terminology, and denotes internal cross-references and citations.

bold — Is used for emphasis in descriptive lists and elsewhere, where appropriate.

SMALL CAPITALS — Are used for a few terms which have specific technical meanings. Their meanings can be found in the *Glossary*.

Pseudo-code descriptions of instructions

A form of pseudo-code is used to provide precise descriptions of what instructions do. This pseudo-code is written in a `typewriter` font, and uses the following conventions for clarity and brevity:

- Indentation is used to indicate structure. For example, the range of statements that a `for` statement loops over, goes from the `for` statement to the next statement at the same or lower indentation level as the `for` statement (both ends exclusive).
- Comments are bracketed by `/*` and `*/`, as in the C language.
- English text is occasionally used outside comments to describe functionality that is hard to describe otherwise.
- All keywords and special functions used in the pseudo-code are described in the *Glossary*.
- Assignment and equality tests are distinguished by using = for an assignment and == for an equality test, as in the C language.

Assembler syntax descriptions

This manual contains numerous syntax descriptions for assembler instructions and for components of assembler instructions. These are shown in a `typewriter` font, and are as follows:

`< >` — 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 simply requires 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 ARM 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.

`{ }` 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.

`|` This indicates an alternative character string. For example, `LDM|STM` is either `LDM` or `STM`.

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.

`*` When used in a combination like `<immed_8> * 4`, this describes an immediate value which must be a specified multiple of a value taken from a numeric range. In this instance, the numeric range is 0 to 255 (the set of values that can be represented as an 8-bit immediate) and the specified multiple is 4, so the value described must be a multiple of 4 in the range 4*0 = 0 to 4*255 = 1020.

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. The { and } characters need to be encoded in a few places as part of a variable item. When this happens, the long description of the variable item indicates how they must be used.

System and debug coprocessors

This document refers to registers in coprocessors 14 (debug) and 15 (system configuration and control) using the following notation:

CP14r<n> refers to the <n>th CP14 register

CP15r<n> refers to the <n>th CP15 register.

Further reading

This section lists publications that provide additional information on the ARM family of processors.

ARM periodically provides updates and corrections to its documentation. See `http://www.arm.com` for current errata sheets and addenda, and the ARM Frequently Asked Questions.

ARM publications

This book is a supplement to, and must be read in conjunction with, the *ARM Architecture Reference Manual* (ARM DDI 0100), version F or later.

Feedback

ARM Limited welcomes feedback on its documentation.

Feedback on this book

If you notice any errors or omissions in this book, send email to `errata@arm` giving:

- the document title
- the document number
- the page number(s) to which your comments apply
- a concise explanation of the problem.

General suggestions for additions and improvements are also welcome.

Chapter 1
Introduction

This chapter introduces the ARM® Architecture Security Extensions.

Note

The ARM Architecture Security Extension, its associated implementations and supporting software are commonly referred to as TrustZone Technology.

This chapter contains the following sections:

1.1 About the ARM Architecture Security Extensions

ARM Architecture Security Extensions is an enhancement to the ARM architecture. It integrates hardware security features to facilitate the development of secure applications.

Security Extensions is based on a small number of fundamental principles:

- It defines a class of core state, that can be switched between Secure and Non-secure state. Most code runs in Non-secure state. Only trusted code runs in Secure state.
- It defines some memory as Secure memory. Secure memory can only be accessed by the core when the core is in Secure state.
- Entry into Secure state is strictly controlled.
- Exit from Secure state can only occur at programmed points.
- Debug is strictly controlled.
- Reset enters Secure state.

Exceptions are generally handled in a similar way to other ARM architectures. However, support is provided for some exceptions to be handled only by code running in Secure state.

1.2 Security state, Monitor mode, and the NS-bit

Security Extensions defines a processor mode, Monitor mode. Monitor mode is designed to provide a bridge between code running in Non-secure state and code running in Secure state.

For all modes except Monitor mode, the security state (Secure or Non-secure) is controlled by the *Non-Secure bit*, the NS-bit. The NS-bit is a bit in a register, the Secure Configuration Register (SCR). See *Secure Configuration Register* on page 3-11 for details.

In Monitor mode, the security state is always Secure, regardless of the state of the NS-bit.

Figure 1-1 shows the normal routes of transfer of control between different modes and security states. The dashed transfer marked `MCR` is a possible but not normally recommended route.

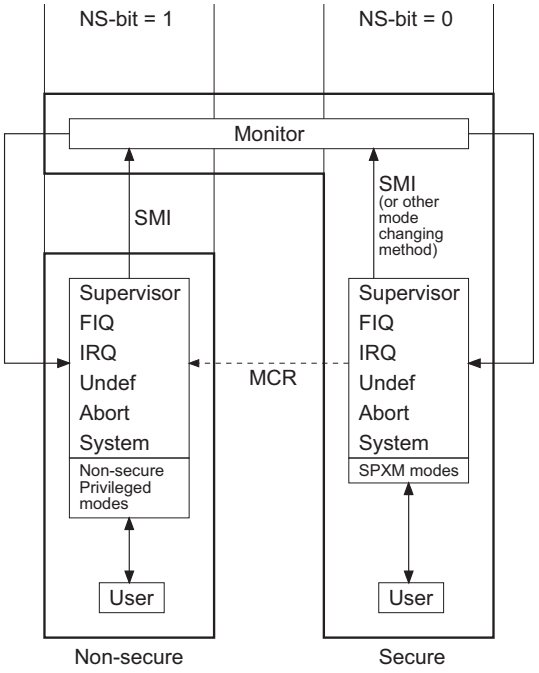


Figure 1-1 Security state, Monitor Mode, and the NS-bit

SPXM modes means *Secure Privileged modes, Excluding Monitor mode*.

Note

It is important to distinguish Monitor mode (a processor mode) from Monitor debug-mode (a debug mode).

1.3 NS attributes

This document refers to several attributes that specify whether particular resources are Secure or Non-secure. In all cases, the value of the attribute has the following meanings:

0 Secure

1 Non-secure.

Primary NS attributes are architecturally defined state maintained in various system locations. These are shown in Table 1-1.

Table 1-1 Primary NS attribute definitions

NS attribute	Description	Location
NS-bit	Determines NS-state, except in Monitor mode. In Monitor mode, controls whether the core accesses Secure or Non-secure resources.	CP15 r1 SCR[0]
NS-dma	Used by L1 DMA requests.	CP15 r1 NSAC[18]
NS-desc	Security descriptor read by a page table walk.	Page tables
NSTID	Security identifier for TLB entry. Indicates whether Secure or Non-secure page tables used to create the entry.	TLB entries
NS-tag	Security tag for cache line.	Cache lines
NS-itcm	Access control for Instruction (or unified) TCM.	CP15 r9 ITCM-NSAC[0]
NS-dtcm	Access control for Data TCM.	CP15 r9 DTCM-NSAC[0]
NS-prot	Security identifier for external memory and peripherals.	*Bus Interface Unit* (BIU)

Derived NS attributes are defined by combinations of the primary attributes. These are shown in Table 1-2.

Table 1-2 Derived NS attribute definitions

NS attribute	Definition	Derivation
NS-state	Current execution state of the core	Secure if (NS-bit==0) or (mode==Monitor)
NS-req	Security state of the memory access request	NS-state for core requests (unless explicitly stated differently), otherwise NS-dma
NS-attr	Security attribute of the memory location	if (MMU==OFF) then NS-req else if (NS-state==0) then NS-desc else 1

1.3.1 Notes

- Some CP15 registers are banked, with both Secure and Non-secure versions.
  - The NS-bit controls reads and writes to banked CP15 registers, for example for `MRC` and `MCR` instructions.
  - NS-req and NS-dma form part of a request to determine which banked CP15 registers are used, for example during address translation.
- NS-state can be stored or calculated from NS-bit and the mode.
- NS-state and NS-bit can be different only in Monitor mode.
- NS-state is visible for debug purposes. It is bit[18] in the CP14 r1 *Debug Status and Control register* (DSCR).
- NS-desc is defined as stored in all page tables, both Secure and Non-secure. However, it is IGNORED for Non-secure page tables (because its value cannot be trusted).
- NS-desc may be cached in a TLB entry along with the rest of the descriptor after it has been fetched from a page table walk.
- NS-attr is required to access the appropriate memory, Secure or Non-secure, at any level of abstraction.
- NSTID only applies to systems with a TLB.
- NS-tag only applies to systems with caches.
- NS-dma only applies to systems with one or more TCMs.

1.3.2 Secure and Non-secure worlds

NS-state describes the running state of the system. The resources available, both core and system, are described as:

- the *Non-secure World* when NS-state==1
- the *Secure World* when NS-state==0.

1.4 Exception handling

On reset, Supervisor mode is entered with NS-state==0 and unrestricted access to the memory map.

Compatibility is a key feature of Security Extensions, allowing VMSAv6 compliant operating system ports to run without change.

The changes to exception handling are as follows:

- There is a new Monitor mode exception.
- There are options to take certain aborts normally, or to switch to Monitor mode to handle them. The options are configured by SCR[3:1] in CP15r1.

  This applies to IRQ, FIQ, precise external aborts, and imprecise external aborts.
- There are options to prevent modification of the CPSR_A and CPSR_F bits by software when NS-state==1. This is configurable by SCR[5:4] in CP15r1.
- For register definitions see *Register 1, control registers* on page 3-9. For details of exception entry for all modes see *Exception model* on page 2-7. Only Secure Privileged modes can modify the secure control register (SCR) associated with these features.

1.5 Switching between Secure and Non-secure contexts

Software running predominantly in Monitor mode is responsible for switching safely between operations in Non-secure and Secure states. This software is called the *secure monitor code*.

It is the responsibility of the secure monitor code to stack and recover all register context associated with the ARM programmers' model during an NS-state transition.

Monitor mode has two banked general-purpose registers and a banked SPSR:

- R13_mon, the Stack Pointer (SP)
- R14_mon, the Link Register (LR)
- SPSR_mon, the Saved Program Status Register.

These registers can only be read or written in Monitor mode. In addition, the SRS instruction can only be used with R13_mon when NS-state==0.

Note

Hardware is not responsible for guaranteeing safe context switching. It assumes that the code running in Secure Privileged modes can be trusted.

1.6 Memory accesses

This section describes the secure memory access model for data, instructions, and address translations. In ARMv6, an ARM memory access can be initiated from two sources:

Core Instruction Read or Data Read/Write, cache writebacks, and page table walks.

L1 DMA Background task to fill TCMs from external memory.

For each request, the access can be for External Memory or L1 Memory (Caches or TCMs).

1.6.1 Security of memory accesses

NS-attr indicates the security of memory accesses. Values of NS-attr depend on two primary NS attributes and whether the MMU is enabled, as defined in Table 1-3.

Table 1-3 NS-attr values

NS-req	Secure M-bit [a]	NS-desc	NS-attr
Non-secure	any	any	Non-secure
Secure	MMU On	Non-secure	Non-secure
Secure	MMU On	Secure	Secure
Secure	MMU Off	any	Secure

a. Non-secure M-bit has no effect on NS-attr.

Note

For page table walks, NS-attr is defined to be NS-req of the transaction which caused the TLB miss and associated page table walk.

1.6.2 Memory management

Memory management involves:

- Address Translation (stored as page tables in external memory) from virtual to physical address space.
- Memory Attributes, for example sharing, caching and protection.

Security Extensions adds an attribute to the VMSAv6 page tables. The attribute is called NS-desc, the PTE-NS bit. For every memory access, an address translation is required when the MMU is on. This requires one or more additional memory accesses from either:

External page table walk

A page table walk involves one or two memory accesses.

TLB A *Translation Look-aside Buffer* (TLB). This provides a cache of address translations.

There is no architectural requirement to implement a TLB. However, where a TLB exists, each entry must be associated with an NSTID to identify whether it was created by a Secure or Non-secure page table. It must also have information on the translated address:

NS-desc Raw value from Page Table. This cannot be trusted if NS-req==1.

If NS-desc is stored in TLB entries, it must be dynamically modified when it is read during a Non-secure address translation.

1.6.3 Caches

Every cache line must include the NS-tag attribute. This can be considered as an additional address tag that must match NS-attr for a cache hit.

1.6.4 TCMs and L1 DMA

For ARMv6, both instruction and data TCM regions can be assigned to either value of NS-state. The architected DMA model that supports TCMs can be reserved exclusively for NS-state==0, or made available to the Non-secure world. See section *Tightly coupled memory* on page 3-43 and *L1 DMA* on page 3-46 for more details.

The security access registers associated with these features can only be modified in Secure Privileged modes.

1.6.5 External memory accesses

All memory must be partitioned into Secure and Non-secure regions. The partitioning can be static or dynamic. It can be done at different levels of the memory hierarchy, with all resources beyond the chosen point allocated to the prescribed region. All memory accesses must be checked against NS-attr. NS-attr is exported to the rest of the system on external memory transactions as NS-prot. System infrastructures must support NS-prot to ensure correctness in, for example, external memory, peripherals, system caches, and page tables.

1.7 Debug

ARM processors support two types of debug:

Invasive Debug of a halted system or running system, using watchpoints or breakpoints.

Non-invasive Data observation of running system, for example, trace, performance monitoring, or PC sampling.

Debug and security have diametrically opposed objectives:

Security Aims to protect security critical data from being corrupted or leaked.

Debug Aims to reveal as much information as possible, using shortcuts not allowed in normal operation.

Security Extensions debug is strictly controlled to avoid compromising security:

- The architecture allows debug in all Non-secure states.
- A system can be configured to enable or disable both types of debug in Secure states. This can be done independently for User and Privileged modes.

System design must configure debug access appropriately for the security needs of the application.

1.7.1 Invasive debug

Invasive debug supports:

- Halting debug-mode. The core can be stopped for examination, then restarted.
- Monitor debug-mode. The foreground task can be stopped and started, but background tasks are not halted.

Invasive debug is controlled by the existing **DBGEN** input and two debug-enables for Security Extensions:

DBGEN input Debug Enable. Required in ARMv6.

SPIDEN input Secure Privileged Invasive Debug Enable.

SUIDEN bit Secure User Invasive Debug Enable. This bit is in the SDE register in CP15r1. See *Secure Debug Enable register* on page 3-15 for details.

Table 1-4 shows a summary of invasive debug control.

Table 1-4 Invasive debug control

DBGEN [a]	SPIDEN	SUIDEN	Secure Privileged modes	Secure User mode	All Non-secure modes
0	X	X	Disabled	Disabled	Disabled
1	0	0	Disabled	Disabled	Enabled
1	0	1	Disabled	Enabled	Enabled
1	1	X [b]	Enabled	Enabled	Enabled

a. This input is called **DBGEN**, not **IDEN**, for compatibility with previous ARM cores.
b. **SPIDEN** = 1 overrides SUIDEN = 0. This enables debug in Secure User mode.

1.7.2 Non-invasive debug

Non-invasive debug allows data observation, but does not allow halting, or any alteration to state. This allows for the following:

- Tracing, for example, ETM.
- Performance monitoring.
- External PC sampling.

Non-invasive debug is controlled by three debug-enables:

NIDEN input Non-Invasive Debug Enable. Optional input, recommended.

SPNIDEN input Secure Privileged Non-Invasive Debug Enable.

SUNIDEN bit Secure User Non-Invasive Debug Enable. This bit is in the SDE register in CP15r1. See *Secure Debug Enable register* on page 3-15 for details.

Table 1-5 shows a summary of non-invasive debug control.

Table 1-5 Non-invasive debug control

NIDEN [a]	SPNIDEN	SUNIDEN	Secure Privileged modes	Secure User mode	All Non-secure modes
0	X	X	Disabled	Disabled	Disabled
1	0	0	Disabled	Disabled	Enabled
1	0	1	Disabled	Enabled	Enabled
1	1	X [b]	Enabled	Enabled	Enabled

a. Optional input.
b. **SPNIDEN** = 1 overrides SUNIDEN = 0. This enables debug in Secure User mode.

Chapter 2
CPU Architecture

This chapter describes the changes to the CPU architecture introduced with Security Extensions. It contains the following sections:

2.1 Processor modes

All the modes in earlier ARM® architectures exist in both Secure and Non-secure states (NS-state==0 and NS-state==1). There is an additional mode, Monitor mode.

Table 2-1 shows the modes.

Table 2-1 Processor modes

Processor	Mode	Description
User	usr	Normal program execution mode
FIQ	fiq	Supports high-speed data transfer or channel process
IRQ	irq	Used for general-purpose interrupt handling
Supervisor	svc	Protected mode for the operating system
Abort	abt	Implements virtual memory or memory protection
Undefined	und	Supports software emulation of hardware coprocessors
System	sys	Runs privileged operating system tasks
Monitor	mon	Runs the secure monitor kernel

Monitor mode is a Privileged mode. In Monitor mode, the processor is always in Secure state (NS-state = 0).

Monitor mode is entered when a software monitor interrupt (`SMI`) instruction is executed, or any of the following exceptions is trapped in the secure monitor:

- FIQ
- IRQ
- external abort.

In Secure Privileged modes, it is also possible to change directly to Monitor mode by modifying the mode bits in the CPSR, for details see *PSR mode bits* on page 2-5.

Table 2-2 shows the state and mode structure.

Table 2-2 Processor states and modes

Modes	Privilege	Processor state	
		NS-bit = 1	NS-bit = 0
User	User	Non-secure state	Secure state
FIQ	Privileged	Non-secure state	Secure state
IRQ	Privileged	Non-secure state	Secure state
Supervisor	Privileged	Non-secure state	Secure state
Abort	Privileged	Non-secure state	Secure state
Undefined	Privileged	Non-secure state	Secure state
System	Privileged	Non-secure state	Secure state
Monitor	Privileged	Secure state	Secure state

2.2 Registers

Security Extensions introduces two banked general-purpose registers and a banked SPSR. These three registers are highlighted in Table 2-3.

Table 2-3 Register organization

User	System	Supervisor	Abort	Undefined	Interrupt	Fast interrupt	Monitor
R0	R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8_fiq	R8
R9	R9	R9	R9	R9	R9	R9_fiq	R9
R10	R10	R10	R10	R10	R10	R10_fiq	R10
R11	R11	R11	R11	R11	R11	R11_fiq	R11
R12	R12	R12	R12	R12	R12	R12_fiq	R12
R13	R13	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq	R13_mon
R14	R14	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq	R14_mon
PC	PC	PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq	SPSR_mon

2.3 Program Status Registers

The format of the CPSR and SPSRs is unchanged.

2.3.1 PSR mode bits

The encoding of the mode bits is shown in Table 2-4. Only the Monitor mode is specific to Security Extensions.

Table 2-4 Mode encoding

M[4:0]	Mode	Accessible registers
0b10000	User	PC, R14-R0, CPSR
0b10001	FIQ	PC, R14_fiq-R8_fiq, R7-R0, CPSR, SPSR_fiq
0b10010	IRQ	PC, R14_irq, R13_irq, R12-R0, CPSR, SPSR_irq
0b10011	Supervisor	PC, R14_svc, R13_svc, R12-R0, CPSR, SPSR_svc
0b10111	Abort	PC, R14_abt, R13_abt, R12-R0, CPSR, SPSR_abt
0b11011	Undefined	PC, R14_und, R13_und, R12-R0, CPSR, SPSR_und
0b11111	System	PC, R14-R0, CPSR
0b10110	Monitor	PC, R14_mon, R13_mon, R12-R0, CPSR, SPSR_mon

Updating PSR mode bits

Programs running in SPXM modes can switch to Monitor mode by:

- using a `CPS` instruction
- using an `MSR` instruction
- using an `SMI`, `RFE`, or `LDM^` instruction
- using a flag-setting data processing instruction (for example `MOVS`, `ADDS`, or `SUBS`) with Rd = R15.

If the core is in Secure or Non-secure User mode, an attempt to enter Monitor mode by changing the CPSR directly using an `MSR` instruction is IGNORED. `RFE` is UNPREDICTABLE in User mode.

In Non-secure Privileged modes, an attempt to enter Monitor mode by changing the CPSR Mode into Monitor Mode using a `CPS`, `MSR`, or `RFE` instruction is UNPREDICTABLE. The Mode field in the SPSR can be changed to Monitor Mode using an `MSR` instruction. However, any instruction executed from a Non-secure Privileged state, which copies the SPSR to CPSR when the SPSR mode field is set to Monitor mode is UNPREDICTABLE.

Note

The UNPREDICTABLE behavior cannot introduce a security violation, and as a result cannot enter Monitor mode or any secure state.

A program running in a Privileged mode, either in Secure state or Non-secure state, can switch to Monitor mode by executing an SMI instruction. See *SMI* on page 2-20 for details.

A program running in User mode, either in Secure state or Non-secure state, can only switch to Monitor mode by first switching into a Privileged mode using a SWI instruction.

2.3.2 PSR F and A bits (ARMv6 only)

The F bit in the CPSR cannot be changed in Non-secure state if the FW bit in the SCR (bit[4]) is 0.

The A bit in the CPSR cannot be changed in Non-secure state if the AW bit in the SCR (bit[5]) is 0.

See *Secure Configuration Register* on page 3-11 for details.

Note

The F bit and the A bit in the SPSR can be changed in Non-secure state even if the corresponding bits in the SCR are 0. However, they are not copied into the CPSR if the corresponding bits in the SCR are 0.

2.3.3 PSR E bit (ARMv6 only)

The E bit in the CPSR is updated with the secure EE bit (CP15 r1 Control[25]) whenever the core switches from Non-secure to Secure state.

When the core stays in the current state on an exception, The E bit in the CPSR is updated with the EE bit of the current state.

2.4 Exception model

Security Extensions introduces an additional type of exception, the Software Monitor exception. This exception is generated by the `SMI` instruction.

IRQ, FIQ, or external abort exceptions can be configured to enter either Secure state or Non-secure state. This is controlled by SCR bits[3:1]. See *Secure Configuration Register* on page 3-11 for details.

The base address of the exception vector tables varies according to the mode the processor is in when the exception occurs. There are two banked registers for the base addresses, and a unique register for Monitor mode. All three registers are in CP15. These are shown in Table 2-5.

Table 2-5 Exception vector registers

Register name		Modes
Non_Secure_Base_Address	Vector Base Address ($VBAR_{NS}$)	All Non-secure states
Secure_Base_Address	Vector Base Address ($VBAR_{S}$)	All Secure states except Monitor mode
Monitor_Base_Address	Monitor Vector Base Address (MVBAR)	Monitor mode

See *CP15 register 12, miscellaneous registers* on page 3-27 for details.

High vectors can be enabled independently for the Secure and Non-secure states using the banked V-bit in CP15r1 (bit[13]). If high vectors are enabled, the associated Non_Secure_Base_Address and/or the Secure_Base_Address register are treated as being `0xFFFF0000`, regardless of the value of these registers.

The sequence of operations when an exception occurs are generally the same as in implementations without Security Extensions. The operation of each exception state, including any differences and additional steps, is listed in the following subsections.

2.4.1 Reset

When Reset is de-asserted:

```
SCR[0]     = 0                  /* NS-bit = 0, Secure state */
R14_svc    = UNPREDICTABLE
SPSR_svc   = UNPREDICTABLE
CPSR[4:0]  = 0b10011            /* Enter supervisor mode */
if Thumb-2 then
    CPSR[5]   = Secure TE-bit   /* Store value of Secure CP15r1 Control[30] */
                                /* Execute in ARM/Thumb */
else
    CPSR[5]   = 0         /* Execute in ARM state */
CPSR[6]   = 1             /* Disable fast interrupts */
CPSR[7]   = 1             /* Disable interrupts */
CPSR[8]   = 1             /* Disable imprecise aborts (from v6) */
CPSR[9]   = Secure EE bit /* Store value of Secure CP15r1 Control[25] (from v6) */
CPSR[24]  = 0             /* Clear J bit (from v5TEJ) */
if high vectors configured then
        PC = 0xFFFF0000
else
        PC = 0x00000000
```

Operation is the same whether Reset is entered from Secure or Non-secure state.

2.4.2 Undefined instruction

On an Undefined instruction in Non-secure state:

```
                                /* NS-bit UNCHANGED */
R14_und   = address of next instruction after the Undefined instruction
SPSR_und  = CPSR
CPSR[4:0] = 0b11011             /* Enter Undefined mode */
if Thumb-2 then
    CPSR[5]   = TE-bit          /* Store value of CP15r1 Control[30] */
                                /* Execute in ARM/Thumb */
else
    CPSR[5] = 0                 /* Execute in ARM state */
CPSR[7]   = 1                   /* Disable interrupts */
CPSR[9]   = EE bit              /* Store value of CP15r1 Control[25] (from v6) */
CPSR[24]  = 0                   /* Clear J bit (from v5TEJ) */
if high vectors configured then
        PC = 0xFFFF0004
else
        PC = Non_Secure_Base_Address + 0x00000004
```

The behavior in Secure state is identical to that in Non-secure state, except that:

- Secure_Base_Address is used instead of Non_Secure_Base_Address.
- If the exception occurs from Monitor mode, the value of NS-bit is changed to 0, that is Secure. See *Exceptions occurring in Monitor mode* on page 2-18 for more details.

2.4.3 Software interrupt

On a SWI in Non-secure state:

```
                              /* NS-bit UNCHANGED */
R14_svc   = address of next instruction after the SWI instruction
SPSR_svc  = CPSR
CPSR[4:0] = 0b10011           /* Enter supervisor mode */
if Thumb-2 then
    CPSR[5]   = TE-bit        /* Store value of CP15r1 Control[30] */
                              /* Execute in ARM/Thumb */
else
    CPSR[5]   = 0             /* Execute in ARM state */
CPSR[7]   = 1                 /* Disable interrupts */
CPSR[9]   = EE bit            /* Store value of CP15r1 Control[25] (from v6) */
CPSR[24]  = 0                 /* Clear J bit (from v5TEJ) */
if high vectors configured then
        PC = 0xFFFF0008
else
        PC = Non_Secure_Base_Address + 0x00000008
```

The behavior in Secure state is identical to that in Non-secure state, except that:

- Secure_Base_Address is used instead of Non_Secure_Base_Address.
- If the exception occurs from Monitor mode, the value of NS-bit is changed to 0, that is Secure. See *Exceptions occurring in Monitor mode* on page 2-18 for more details.

2.4.4 External prefetch abort

On an external prefetch abort in Non-secure state:

```
                                    /* NS-bit UNCHANGED */
if SCR[3] == 1 then                 /* external prefetch trapped to Monitor mode */
    R14_mon    = address of next instruction + 4
    SPSR_mon   = CPSR
    CPSR[4:0] = 0b10110             /* Enter Monitor mode */
    if Thumb-2 then
        CPSR[5]   = Secure TE-bit   /* Store value of Secure CP15r1 Control[30] */
                                    /* Execute in ARM/Thumb */
    else
        CPSR[5] = 0             /* Execute in ARM state */
    CPSR[6]   = 1               /* Disable fast interrupts */
    CPSR[7]   = 1               /* Disable interrupts */
    CPSR[8]   = 1               /* Disable imprecise aborts (from v6) */
    CPSR[9]   = Secure EE bit   /* Store value of Secure CP15r1 Control[25] (from v6) */
    CPSR[24]  = 0               /* Clear J bit (from v5TEJ) */
    PC = Monitor_Base_Address + 0x0000000C
else                            /* external prefetch trapped to Abort mode */
    R14_abt    = address of next instruction + 4
    SPSR_abt   = CPSR
    CPSR[4:0] = 0b10111             /* Enter Abort mode */
    if Thumb-2 then
        CPSR[5]   = TE-bit          /* Store value of CP15r1 Control[30] */
                                    /* Execute in ARM/Thumb */
    else
        CPSR[5] = 0           /* Execute in ARM state */
    CPSR[7]   = 1             /* Disable interrupts */
    if SCR[5] == 1 then       /* bit AW */
        CPSR[8] = 1           /* Disable imprecise aborts (from v6) */
                              /* else CPSR[8] = UNCHANGED */
    CPSR[9]   = EE bit        /* Store value of CP15r1 Control[25] (from v6) */
    CPSR[24]  = 0             /* Clear J bit (from v5TEJ) */
    if high vectors configured then
        PC = 0xFFFF000C
    else
        PC = Non_Secure_Base_Address + 0x0000000C
```

The behavior in Secure state is identical to that in Non-secure state, except that:

- Secure_Base_Address is used instead of Non_Secure_Base_Address
- CPSR[8] (Abit) is updated regardless of bit[5] of the SCR.
- If the exception occurs from Monitor mode, the value of NS-bit is changed to 0, that is Secure. See *Exceptions occurring in Monitor mode* on page 2-18 for more details.

2.4.5 Internal prefetch abort

On an internal prefetch abort in Non-secure state:

```
                              /* NS-bit UNCHANGED */
R14_abt    = address of next instruction + 4
SPSR_abt   = CPSR
CPSR[4:0] = 0b10111           /* Enter Abort mode */
if Thumb-2 then
    CPSR[5]   = TE-bit        /* Store value of CP15r1 Control[30] */
                              /* Execute in ARM/Thumb */
else
    CPSR[5] = 0               /* Execute in ARM state */
CPSR[7]   = 1                 /* Disable interrupts */
if SCR[5] == 1 then           /* bit AW */
    CPSR[8] = 1               /* Disable imprecise aborts (from v6) */
                              /* else CPSR[8] = UNCHANGED */
CPSR[9]   = EE bit            /* Store value of CP15r1 Control[25] (from v6) */
CPSR[24]  = 0                 /* Clear J bit (from v5TEJ) */
if high vectors configured then
    PC = 0xFFFF000C
else
    PC = Non_Secure_Base_Address + 0x0000000C
```

The behavior in Secure state is identical to that in Non-secure state, except that:

- Secure_Base_Address is used instead of Non_Secure_Base_Address
- CPSR[8] (Abit) is updated regardless of bit[5] of the SCR.
- If the exception occurs from Monitor mode, the value of NS-bit is changed to 0, that is Secure. See *Exceptions occurring in Monitor mode* on page 2-18 for more details.

2.4.6 External data abort

On an external precise data abort, or an external imprecise abort with CPSR[8] = 0, in Non-secure state:

```
                                /* NS-bit UNCHANGED */
if SCR[3] == 1 then             /* external aborts trapped to Monitor mode */
    R14_mon    = address of next instruction + 8
    SPSR_mon   = CPSR
    CPSR[4:0] = 0b10110         /* Enter Monitor mode */
    if Thumb-2 then
        CPSR[5]   = Secure TE-bit   /* Store value of Secure CP15r1 Control[30] */
                                    /* Execute in ARM/Thumb */
    else
        CPSR[5] = 0           /* Execute in ARM state */
    CPSR[6]   = 1             /* Disable fast interrupts */
    CPSR[7]   = 1             /* Disable interrupts */
    CPSR[8]   = 1             /* Disable imprecise aborts (from v6) */
    CPSR[9]   = Secure EE bit /* Store value of Secure CP15r1 Control[25] (from v6) */
    CPSR[24]  = 0             /* Clear J bit (from v5TEJ) */
    PC = Monitor_Base_Address + 0x00000010
else                          /* external aborts trapped to current state Abort mode */
    R14_abt    = address of next instruction + 8
    SPSR_abt   = CPSR
    CPSR[4:0]  = 0b10111        /* Enter Abort mode */
    if Thumb-2 then
        CPSR[5]   = TE-bit      /* Store value of CP15r1 Control[30] */
                                /* Execute in ARM/Thumb */
    else
        CPSR[5] = 0             /* Execute in ARM state */
    CPSR[7]   = 1               /* Disable interrupts */
    if SCR[5] == 1 then         /* bit AW */
        CPSR[8] = 1             /* Disable imprecise aborts (from v6) */
                                /* else CPSR[8] = UNCHANGED */
    CPSR[9]   = EE bit          /* Store value of CP15r1 Control[25] (from v6) */
    CPSR[24]  = 0               /* Clear J bit (from v5TEJ) */
    if high vectors configured then
        PC = 0xFFFF0010
    else
        PC = Non_Secure_Base_Address + 0x00000010
```

The behavior in Secure state is identical to that in Non-secure state, except that:

- Secure_Base_Address is used instead of Non_Secure_Base_Address
- CPSR[8] (Abit) is updated regardless of bit[5] of the SCR.
- If the exception occurs from Monitor mode, the value of NS-bit is changed to 0, that is Secure. See *Exceptions occurring in Monitor mode* on page 2-18 for more details.

2.4.7 Internal data abort

On an internal data abort in Non-secure state:

```
                              /* NS-bit UNCHANGED */
R14_abt    = address of next instruction + 8
SPSR_abt   = CPSR
CPSR[4:0]  = 0b10111          /* Enter Abort mode */
if Thumb-2 then
    CPSR[5]   = TE-bit        /* Store value of CP15r1 Control[30] */
                              /* Execute in ARM/Thumb */
else
    CPSR[5] = 0               /* Execute in ARM state */
CPSR[7]   = 1                 /* Disable interrupts */
if SCR[5] == 1 then           /* bit AW */
    CPSR[8] = 1               /* Disable imprecise aborts (from v6) */
                              /* else CPSR[8] = UNCHANGED */
CPSR[9]   = EE bit            /* Store value of CP15r1 Control[25] (from v6) */
CPSR[24]  = 0                 /* Clear J bit (from v5TEJ) */
if high vectors configured then
    PC = 0xFFFF0010
else
    PC = Non_Secure_Base_Address + 0x00000010
```

The behavior in Secure state is identical to that in Non-secure state, except that:

- Secure_Base_Address is used instead of Non_Secure_Base_Address
- CPSR[8] (Abit) is updated regardless of bit[5] of the SCR.
- If the exception occurs from Monitor mode, the value of NS-bit is changed to 0, that is Secure. See *Exceptions occurring in Monitor mode* on page 2-18 for more details.

2.4.8 Interrupt request

On an interrupt request (IRQ), with the I bit, CPSR[7] = 0, in Non-secure state:

```
                                /* NS-bit UNCHANGED */
if SCR[1] == 1 then             /* IRQ trapped to Monitor mode */
    R14_mon   = address of next instruction + 4
    SPSR_mon  = CPSR
    CPSR[4:0] = 0b10110         /* Enter Monitor mode */
    if Thumb-2 then
        CPSR[5]   = Secure TE-bit   /* Store value of Secure CP15r1 Control[30] */
                                    /* Execute in ARM/Thumb */
    else
        CPSR[5] = 0           /* Execute in ARM state */
    CPSR[6]   = 1             /* Disable fast interrupts */
    CPSR[7]   = 1             /* Disable interrupts */
    CPSR[8]   = 1             /* Disable imprecise aborts (from v6) */
    CPSR[9]   = Secure EE bit /* Store value of Secure CP15r1 Control[25] (from v6) */
    CPSR[24]  = 0             /* Clear J bit (from v5TEJ) */
    PC = Monitor_Base_Address + 0x00000018
else                          /* IRQ trapped to current state IRQ mode */
    R14_irq   = address of next instruction + 4
    SPSR_irq  = CPSR
    CPSR[4:0] = 0b10010           /* Enter IRQ mode */
    if Thumb-2 then
        CPSR[5]   = TE-bit        /* Store value of CP15r1 Control[30] */
                                  /* Execute in ARM/Thumb */
    else
        CPSR[5] = 0           /* Execute in ARM state */
    CPSR[7]   = 1             /* Disable interrupts */
    if SCR[5] == 1 then       /* bit AW */
        CPSR[8]   = 1         /* Disable imprecise aborts (from v6) */
                              /* else CPSR[8] = UNCHANGED */
    CPSR[9]   = EE bit        /* Store value of CP15r1 Control[25] (from v6) */
    CPSR[24]  = 0             /* Clear J bit (from v5TEJ) */
    if VE == 0 then           /* Core with VIC port only */
        if high vectors configured then
            PC = 0xFFFF0018
        else
            PC = Non_Secure_Base_Address + 0x00000018
    else
        PC = IMPLEMENTATION DEFINED
```

The behavior in Secure state is identical to that in Non-secure state, except that:

- Secure_Base_Address is used instead of Non_Secure_Base_Address
- CPSR[8] (Abit) is updated regardless of bit[5] of the SCR.
- If the exception occurs from Monitor mode, the value of NS-bit is changed to 0, that is Secure. See *Exceptions occurring in Monitor mode* on page 2-18 for more details.

2.4.9 Fast interrupt request

On a fast interrupt request (FIQ), with the F bit, CPSR[6] = 0, in Non-secure state:

```
                                    /* NS-bit UNCHANGED */
if SCR[2] == 1 then                 /* FIQ trapped to Monitor mode */
    R14_mon   = address of next instruction + 4
    SPSR_mon  = CPSR
    CPSR[4:0] = 0b10110             /* Enter Monitor mode */
    if Thumb-2 then
        CPSR[5]   = Secure TE-bit   /* Store value of Secure CP15r1 Control[30] */
                                    /* Execute in ARM/Thumb */
    else
        CPSR[5] = 0           /* Execute in ARM state */
    CPSR[6]   = 1             /* Disable fast interrupts */
    CPSR[7]   = 1             /* Disable interrupts */
    CPSR[8]   = 1             /* Disable imprecise aborts (from v6) */
    CPSR[9]   = Secure EE bit /* Store value of Secure CP15r1 Control[25] (from v6) */
    CPSR[24]  = 0             /* Clear J bit (from v5TEJ) */
    PC = Monitor_Base_Address + 0x0000001C
else                                /* FIQ trapped to FIQ mode */
                                    /* SCR[4] (FW) must be set */
                                    /* to avoid infinite loop until FIQ asserted */
    R14_fiq    = address of next instruction + 4
    SPSR_fiq   = CPSR
    CPSR[4:0] = 0b10001             /* Enter FIQ mode */
    if Thumb-2 then
        CPSR[5]   = TE-bit          /* Store value of CP15r1 Control[30] */
                                    /* Execute in ARM/Thumb */
    else
        CPSR[5] = 0           /* Execute in ARM state */
    if SCR[4] == 1 then       /* bit FW */
        CPSR[6] = 1           /* Disable fast interrupts */
                              /* else CPSR[6] = UNCHANGED */
    CPSR[7]   = 1             /* Disable interrupts */
    if SCR[5] == 1 then       /* bit AW */
        CPSR[8] = 1           /* Disable imprecise aborts (from v6) */
                              /* else CPSR[8] = UNCHANGED */
    CPSR[9]   = EE bit        /* Store value of CP15r1 Control[25] (from v6) */
    CPSR[24]  = 0             /* Clear J bit (from v5TEJ) */
    if VE == 0 then           /* Core with VIC port only */
        if high vectors configured then
            PC = 0xFFFF001C
        else
            PC = Non_Secure_Base_Address + 0x0000001C
    else
        PC = IMPLEMENTATION DEFINED
```

The behavior in Secure state is identical to that in Non-secure state, except that:

- Secure_Base_Address is used instead of Non_Secure_Base_Address
- CPSR[6] (F bit) and CPSR[8] (Abit) are updated regardless of bits[5:4] of the SCR.

- If the exception occurs from Monitor mode, the value of NS-bit is changed to 0, that is Secure. See *Exceptions occurring in Monitor mode* on page 2-18 for more details.

2.4.10 Software monitor exception

On an SMI in Secure or Non-secure state:

```
if (MonitorMode)
    SCR[0] = 0                      /* NS-bit set to Secure */
else
                                    /* NS-bit UNCHANGED */
if (UserMode) then                  /* Undefined instruction */
```

(see *Undefined instruction* on page 2-9)

```
else
    R14_mon   = address of next instruction after SMI
    SPSR_mon  = CPSR
    CPSR[4:0] = 0b10110             /* Enter Monitor mode */
    if Thumb-2 then
        CPSR[5]   = Secure TE-bit   /* Store value of Secure CP15r1 Control[30] */
                                    /* Execute in ARM/Thumb */
    else
        CPSR[5] = 0           /* Execute in ARM state */
    CPSR[6]   = 1             /* Disable fast interrupts */
    CPSR[7]   = 1             /* Disable interrupts */
    CPSR[8]   = 1             /* Disable imprecise aborts (from v6) */
    CPSR[9]   = Secure EE bit /* Store value of CP15r1 Control[25] (from v6) */
    CPSR[24]  = 0             /* Clear J bit (from v5TEJ) */
    PC = Monitor_Base_Address + 0x00000008
                              /* SMI vectored to the SWI vector */
```

Note

If the exception occurs from Monitor mode, the value of NS-bit is changed to 0, that is Secure. See *Exceptions occurring in Monitor mode* for more details.

2.4.11 Exception priorities

Exception priorities are the same as in other ARM architectures. The SMI exception has the same priority as `BKPT`, Undefined instruction, and SWI exceptions. That is, it has the lowest priority.

2.4.12 Exceptions occurring in Monitor mode

Except reset, the software model does not expect any other exception to occur in secure Monitor mode. Any exception occurring in Monitor mode causes the NS-bit in the Secure Configuration register (SCR[0]) to reset, forcing secure state entry for all exceptions. IRQ, FIQ, and external aborts will remain in Monitor mode if the relevant SCR[3:1] bit is set, otherwise the exception is taken in IRQ, FIQ, or Abort mode respectively.

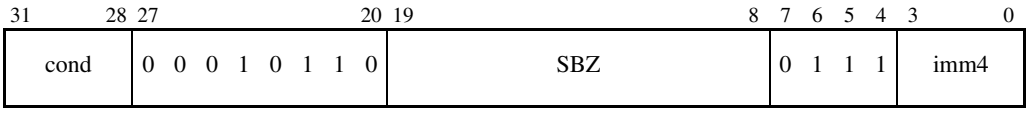
2.5 ARM instruction set

Security Extensions introduces one new ARM instruction, SMI, as defined in *SMI* on page 2-20. In addition, several other instructions are modified. Specifically, all instructions which directly manipulate the CSPR are modified, that is, MSR, RFE, and CPS. In addition, all instructions which cause a copy of the SPSR to the CSPR are also modified. These changes mean that any attempts to change into Monitor mode from a Non-secure Privileged mode are UNPREDICTABLE.

The list of modified instructions is as follows:

- CPS
- LDM (3)
- MSR
- RFE
- SRS
- Flag setting data operations (includes MOV) which write to the PC, for example, ADDS PC,Rn,Rm.

2.5.1 SMI



The SMI (Software Monitor) instruction causes an SMI exception.

Syntax

```
SMI{<cond>} <imm4>
```

where:

<cond> Is the condition under which the instruction is executed. The conditions are defined in the *ARM Architecture Reference Manual, Part A*. If <cond> is omitted, the AL (always) condition is used.

<imm4> Is a 4-bit immediate value. This is ignored by the ARM processor. It can be used by the SMI exception handler (monitor code) to determine what service is being requested, but this is not recommended (see *Notes*).

Exceptions

Software Monitor Interrupt, Undefined instruction.

Operation

See *Software monitor exception* on page 2-18.

Usage

Use the SMI instruction to request a secure monitor kernel service.

Notes

User mode The SMI instruction can only be executed in a Privileged mode. Attempting to execute an SMI instruction in User mode causes an Undefined Instruction exception.

Use of the Immediate value

The SMI instruction from the non-secure world causes entry into the secure world. This means that the code sequence for the secure world to read the instruction and determine the imm4 value is very complicated, particularly in systems using virtual memory. It is recommended that any arguments used to communicate which service is being requested are passed using registers, rather than using immediate values.

2.6 Security Extensions and VFP support

None of the registers in VFP are banked in the Security Extensions.

Chapter 3
Memory and System Architecture

This chapter describes the changes to the memory and system architecture introduced with Security Extensions. It contains the following sections:

3.1 System control coprocessor

This section summarizes the CP15 register space, and the read/write permissions of CP15 registers that are affected by Security Extensions. See the *ARM Architecture Reference Manual* for information about other CP15 registers, for example, common CP15r0 and implementation-defined CP15r15.

Access to CP15 registers is only allowed in Privileged modes, with the following exceptions:

- the PrefetchFlush and data memory barrier (DMB, DWB/DSB) operations
- L1 DMA (ARMv6 only) when configured for user access
- Smartcache operations (ARMv6 only).

The access permissions of IMPLEMENTATION DEFINED registers are IMPLEMENTATION DEFINED, but must not introduce any scope for security violations into the processor.

All other accesses are UNDEFINED in User mode, in both Secure and Non-secure state (when NS-state = 0 and 1 respectively). Where privileged accesses are further restricted due to Security Extensions, this is noted in the relevant sections.

3.1.1 Terminology for CP15 register selection

The following terminology is used to distinguish the application and the selection of CP15 registers:

CP15 access Reading or writing a CP15 register using an `MRC`, `MCR`, or `MCRR` instruction.

Selected by NS-bit. This allows Monitor mode to access both Secure and Non-secure CP15 registers.

CP15 usage Using a CP15 register internally, for example using the M-bit for MMU on/off control during a memory access.Selected by NS-req. This maintains security of the original request for memory access.

See *NS attributes* on page 1-4 for definitions of the NS attributes.

3.1.2 CP15 register space summary

Table 3-1 shows the primary allocation of the CP15 registers.

Table 3-1 Primary CP15 register mapping

Register	Generic use	Specific uses	For details see:
0	ID codes (read-only)	Processor ID, Cache, TCM, TLB type	a
1	Control bits	System Configuration Bits, Secure Configuration Register, Non-secure Access Control Register	b
2	Memory protection and control	Page table control	a
3	Memory protection and control	Domain access control	a
4	RESERVED	-	a
5	Memory protection and control	Fault Status	a
6	Memory protection and control	Fault Address	a
7	Cache and write buffer	Cache/write buffer control, VA to PA translation	a
8	Memory protection and control	TLB functions	a
9	Cache and TCM control	Cache lockdown, TCM region bits	a
10	Memory protection and control	TLB lockdown	a
11	DMA control	Internal DMA control	a
12	Miscellaneous	Vector Base Address, Monitor Vector Base Address, Interrupt Status	a
13	Process ID	FCSE ID, Process ID	a
14	RESERVED	-	a
15	IMPLEMENTATION DEFINED	IMPLEMENTATION DEFINED	a

a. *ARM Architecture Reference Manual*
b. *Register 1, control registers* on page 3-9.

3.1.3 Banked CP15 registers

Some CP15 registers are banked. Banked registers have two copies, one Secure and one Non-secure. The NS-bit selects the Secure or Non-secure register. Table 3-2 shows which registers are banked, and what accesses are permitted.

Table 3-2 Banked CP15 registers

CP15	Banked register	Permitted accesses [a]
Register 1	Control	Read/write in privileged modes [b]
	Auxiliary control [c]	Read/write in privileged modes
Register 2	TTB0 (Translation Table Base 0)	Read/write in privileged modes
	TTB1 (Translation Table Base 1) [d]	Read/write in privileged modes
	TTBC (Translation Table Base Control) [d]	Read/write in privileged modes
Register 3	DACR (Domain Access Control Register)	Read/write in privileged modes
Register 5	DFSR (Data Fault Status Register)	Read/write in privileged modes
	IFSR (Instruction Fault Status Register)	Read/write in privileged modes
Register 6	DFAR (Data Fault Address Register)	Read/write in privileged modes
	IFAR (Instruction Fault Address Register)	Read/write in privileged modes
Register 7	Cache dirty status [d]	Read only in privileged modes
	Physical address	Read/write in privileged modes
Register 9	TCMSR (TCM Selection Register)	Read/write in privileged modes
Register 10		
Register 12	VBAR (Vector Base Address Register)	Read/write in privileged modes
Register 13	FCSE PID register	Read/write in privileged modes
	Context ID register (includes ASID)	Read/write in privileged modes

a. Any attempt to execute an access that is not permitted results in an Undefined Instruction exception.
b. Some bits are common to the Secure and the Non-secure register, see *Control register* on page 3-9.
c. IMPLEMENTATION DEFINED.
d. ARMv6 only.

3.1.4 Restricted access CP15 registers

Some CP15 registers are only present in the Secure world. Read/write access permissions are:

- Secure CP15 registers cannot be modified in Non-secure state. This applies to the Secure version of banked registers, and to registers that only have a Secure version.
- NSAC can be read in Non-secure Privileged modes, but not in Non-secure User mode. This enables permissions to be read, for common CP15 registers that have configurable access.
- Apart from NSAC, Secure CP15 registers cannot be read in any Non-secure state.

Table 3-3 Secure CP15 registers

CP15	Secure Register	Permitted accesses [a]
Register 1	NSAC (Non-Secure Access Control)	Read/write in Secure Privileged modes Read only in Non-secure Privileged modes
	SCR (Secure Configuration Register)	Read/write in Secure Privileged modes
	SDE (Secure Debug Enable)	Read/write in Secure Privileged modes
Register 6	WFAR (Watchpoint Fault Address Register) [b]	Read/write in Secure Privileged modes
Register 9	DTCM-NSAC (Data TCM Non-Secure Access Control)	Read/write in Secure Privileged modes
	ITCM-NSAC (Instruction or unified TCM Non-Secure Access Control)	Read/write in Secure Privileged modes
Register 12	MVBAR (Monitor Vector Base Address Register)	Read/write in Secure Privileged modes

a. Any attempt to execute an access that is not permitted results in an Undefined Instruction exception.
b. Use deprecated.

Note

NSAC bits[15:14] are unused and SBZ. Access to CP15 and CP14 is not affected by NSAC.

Do not confuse the SCR (Secure Configuration Register) with the Secure version of the Control Register. SCR is Secure only.

3.1.5 Configurable access CP15 registers

Access to some CP15 registers is configurable. They can be configured to be accessible from Secure state only, or from both Secure and Non-secure state. This is controlled by Access Control bits in the CP15r1 NSAC register, the CP15r9 ITCM-NSAC, and CP15r9 DTCM-NSAC registers.

Table 3-4 shows the configurable access CP15 registers. For all these registers:

- if the corresponding access control bit is 0, access is restricted to Secure Privileged modes
- if the corresponding access control bit is 1, access is allowed from all Privileged modes.

For registers with restricted access:

- if the entire register is controlled by a single NSAC bit, any access (read or write) is UNDEFINED
- if parts of the register are controlled by different access control bits, those parts are RAZ/WI.

Table 3-4 Configurable access CP15 registers

CP15	Common Register	Controlled by
Register 1	Co-Processor Access Control	NSAC[13:0]
Register 9	Data TCM Region	DTCM-NSAC[0] [a]
	Instruction or unified TCM Region	ITCM-NSAC[0] [b]
	DCLR (Data Cache Lockdown Register)	NSAC[16]
	ICLR (Instruction Cache Lockdown Register)	NSAC[16]
Register 10	TLB Lockdown	NSAC[17]
Register 11	DMA Control	NSAC[18] [c]

a. NS-dtcm.
b. NS-itcm.
c. NS-dma.

3.1.6 CP15SDISABLE input

An input is provided to disable write access to some of the Secure registers. This input is called **CP15SDISABLE**. The interaction between **CP15SDISABLE** and any IMPLEMENTATION DEFINED registers is IMPLEMENTATION DEFINED.

Table 3-5 and Table 3-6 on page 3-8 show the registers and operations affected.

The input must be cleared (**CP15SDISABLE** = 0) on reset by the external system. This allows the Reset code to set up the Security Extension configuration. When the input is set (**CP15SDISABLE** = 1), any attempt to write to the Secure register results in an Undefined instruction exception. The **CP15SDISABLE** input does not affect reading Secure registers, or reading or writing Non-Secure registers.It is IMPLEMENTATION DEFINED how the input is changed and when changes to this input are reflected in the core. However, changes must be reflected as quickly as possible. The change must occur before completion of a PrefetchFlush CP15 operation, issued after the change, is visible to the core with respect to instruction execution boundaries. Software must perform a PrefetchFlush operation meeting the above conditions to ensure all subsequent instructions are affected by the change to **CP15SDISABLE**.The assertion of **CP15SDISABLE** enables key secure privileged features to be locked in a known good state, providing an additional level of overall system security. ARM® expects control of this input to reside in the system, in a system block dedicated to security.

Table 3-5 Secure registers affected by CP15SDISABLE

CP15 Register	Register name	Affected operation
1	Control register	`MCR p15, 0, Rd, c1, c0, 0`
2	Translation Table Base register 0	`MCR p15, 0, Rd, c2, c0, 0`
2	Translation Table Control register	`MCR p15, 0, Rd, c2, c0, 2`
3	Domain Access Control register	`MCR p15, 0, Rd, c3, c0, 0`
12	Vector Base register	`MCR p15, 0, Rd, c12, c0, 0`
12	Monitor Base register	`MCR p15, 0, Rd, c12, c0, 1`
13	FCSE ID register	`MCR p15, 0, Rd, c13, c0, 0`

Table 3-6 TCM related registers affected by CP15SDIABLE

CP15 r9 Register name	Affected operation
Data TCM Region register [a]	`MCR p15, 0, Rd, c9, c1, 0`
Instruction or unified TCM Region register [a]	`MCR p15, 0, Rd, c9, c1, 1`
Data TCM Non-Secure Access Control register (DTCM-NSAC)	`MCR p15, 0, Rd, c9, c1, 2`
Instruction or unified TCM Non-Secure Access Control register (ITCM-NSAC)	`MCR p15, 0, Rd, c9, c1, 3`

a. Region registers where the associated TCM-NSAC access bit is configured to restrict access to Secure world.

3.2 Register 1, control registers

CP15 register 1 contains configuration control bits for the ARM processor. It contains six registers selected by the `opcode_1`, `opcode_2`, and `CRm` fields.

Some registers are banked, because the configuration of the Non-secure and the Secure world can be different.

Table 3-7 shows the arrangement of CP15 register 1.

Table 3-7 CP15 register 1

Opcode_1	Opcode_2	CRm	Common register	Non-secure registers	Secure registers
0b0000	0b000	C0		Control register	Control register
0b0000	0b001	C0		Auxiliary control register (v6 only)	Auxiliary control register (v6 only)
0b0000	0b010	C0	Coprocessor access control register (v6 only)		
0b0000	0b000	C1			Secure configuration register
0b0000	0b001	C1			Secure debug enable register
0b0000	0b010	C1			Non-secure access control register

3.2.1 Control register

The Control register is banked in the two states. However, some bits define a global configuration of the system and are therefore shared between the Secure and Non-secure worlds.

The B, FI, L4, and RR bits are not banked and can only be modified in Secure Privileged modes. They are read-only in Non-secure Privileged modes.

Note

Use of the B and L4 bits is deprecated.

3.2.2 Auxiliary Control register

The Auxiliary Control register is banked in the two states. However, the contents are IMPLEMENTATION DEFINED.

Some bits might define a global configuration of the system. In this case they are shared between the Secure and Non-secure worlds.

3.2.3 Secure Configuration Register

The *Secure Configuration Register* (SCR) defines the configuration of the current state. It specifies the state of the core (Secure or Non-secure), and what mode the core branches to if an IRQ, FIQ or external abort occurs. The register also defines whether or not the F and A bits in the CPSR can be modified when NS-state==1.

The SCR is accessible in Secure Privileged modes only. An attempt to access this register in any other mode results in an Undefined Instruction exception.

Changing from Secure to Non-secure state

It is recommended that the SCR is modified only in Monitor mode. Monitor mode is responsible for switching between states.

To return to Non-secure state, set the NS-bit in the SCR and then execute a `MOVS` or `SUBS` instruction. All implementations must ensure that any prefetched instructions following `MOVS` or `SUBS` (or equivalent) instructions cannot be executed with the secure access permissions.

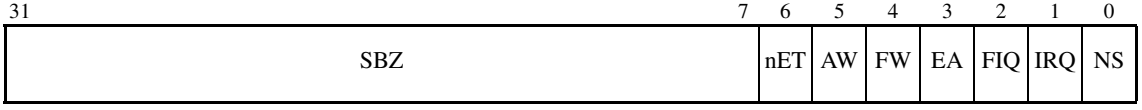
Caution

To avoid security holes, it is strongly recommended that:

- you do not use an `MSR` instruction to change from Secure to Non-secure state
- you do not alter the NS-bit in any mode except Monitor mode.

The usual mechanism for changing from Secure to Non-secure state is an exception return.

SCR-bit allocation



The reset value of the SCR is `0x00000000`.

Use the following instructions to alter or read the SCR:

```
MCR CP15, 0, <Rd>, C1, C1, 0 ; moves contents of <Rd> into SCR
MRC CP15, 0, <Rd>, C1, C1, 0 ; moves contents of SCR into <Rd>
```

NS-bit

The NS-bit (bit[0]), together with the mode bits in the CPSR, defines whether the core is in Secure state or Non-secure state. This is shown in Table 3-8.

Table 3-8 Security state

NS-bit value	Monitor mode	All modes except Monitor mode
0	Secure state	Secure state
1	Secure state	Non-secure state

The value of the NS-bit also affects the accessibility of the banked CP15 registers in Monitor mode, see *Access to registers in Monitor mode* on page 3-19.

EA, FIQ, and IRQ bits

These bits determine whether External Abort, FIQ, and IRQ exceptions are handled in Abort mode or Monitor mode:

EA = 0 Branch to Abort mode on an External Abort exception. This is the default.

EA = 1 Branch to Monitor mode on an External Abort exception.

FIQ = 0 Branch to FIQ mode on an FIQ exception. This is the default.

FIQ = 1 Branch to Monitor mode on an FIQ exception.

IRQ = 0 Branch to IRQ mode on an IRQ exception. This is the default.

IRQ = 1 Branch to Monitor mode on an IRQ exception.

Whenever the core changes security state, the monitor code can change the value of these bits. This means that the behavior of IRQ, FIQ and External Abort exceptions can be different in each state.

Note

In addition to determining the mode, these bits affect which exception base address is used. See *Exception model* on page 2-7 for details.

nET-bit

The nET-bit controls an option to disable early termination. This mechanism can be used to disable data dependent timing optimisations from multiplies and data operations. The details are IMPLEMENTATION DEFINED. The feature is designed to provide system support against information leakage, leakage which could be exploited by timing correlation types of attack.

nET==0 Enable early termination. Execution time of data operations can depend on the data values.

nET==1 Disable early termination. The number of cycles for data operations is forced to be independent of the data values.

FW-bit

The FW-bit controls whether the Non-secure world can modify the F-bit in the CPSR:

FW = 0 CPSR F-bit cannot be modified in Non-secure world. This is the default.

FW = 1 CPSR F-bit can be modified in Non-secure world.

Note

Interrupts driven by secure peripherals are called Secure Interrupts. If the FW-bit is 0 and bit[2] of SCR (trap FIQ to Monitor mode) is 1, FIQ exceptions can be used as Secure Interrupts. These enter Secure world in a deterministic way.

Table 3-9 shows the options for the FW and FIQ bits, and their functions.

Table 3-9 FIQ configuration

FW (SCR[4])	FIQ (SCR[2])	Function
1	0	FIQs handled locally.
1	1	Use with care. Allows Non-secure world to deny service.
0	0	Do not use in Non-secure world. This is the Reset state.
0	1	FIQ can be configured as a Secure interrupt.

This feature is complementary to the *Non-Maskable Fast Interrupt* (NMFI) architecture feature. For a description of NMFI behavior, and the implications of NMFI when used with SCR[4,2], see the NMFI section in the chapter describing the ARM Programmers' Model in the *ARM Architecture Reference Manual, Thumb-2 supplement*.

AW-bit

The AW-bit controls whether the Non-secure world can modify the A-bit in the CPSR:

AW = 0 CPSR A-bit cannot be modified in Non-secure world. This is the default.

AW = 1 CPSR A-bit can be modified in Non-secure world.

Note

If the AW-bit is 0 and bit[3] of SCR (trap external aborts to Monitor mode) is 1, all security aborts from peripherals can be treated in a safe manner in Monitor mode.

Table 3-10 shows the options for the AW and EA bits, and their functions.

Table 3-10 External abort configuration

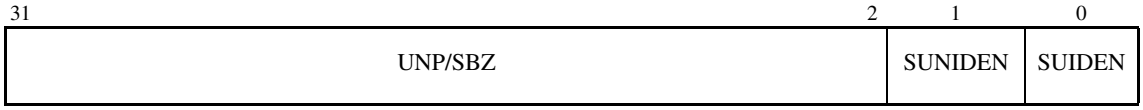
AW (SCR[5])	EA (SCR[3])	Function
1	0	Aborts handled locally.
1	1	All external data aborts trapped to Monitor mode, but enables Non-secure world to hide imprecise external data aborts from the Monitor.
0	0	Do not use in Non-secure world. This is the Reset state.
0	1	All external aborts, both precise and imprecise, reliably trapped to Monitor mode.

3.2.4 Secure Debug Enable register

The *Secure Debug Enable* register (SDE) enables or disables both Invasive and Non-invasive debug for Secure User mode. It is accessible in Secure Privileged modes only. See *External debug interface* on page 4-14 for further information about the usage of the SUNIDEN and SUIDEN bits.

Secure Privileged mode debug is controlled by hardware only. See *External debug interface* on page 4-14 for details.

SDE-bit allocation



The reset value of the SDE is UNDEFINED.

The meaning of the SUNIDEN and SUIDEN bits is as follows:

SUIDEN Invasive Secure User Debug Enable bit:

- **0** Invasive debug is not permitted in secure user mode.
- **1** Invasive debug is permitted in secure user mode.

SUNIDEN Non-Invasive Secure User Debug Enable bit:

- **0** Non-Invasive debug is not permitted in secure user mode.
- **1** Non-Invasive debug is permitted in secure user mode.

Use the following instructions to alter or read the SDE:

```
MCR CP15, 0, <Rd>, C1, C1, 1; moves contents of <Rd> into SDE
MRC CP15, 0, <Rd>, C1, C1, 1; moves contents of SDE into <Rd>
```

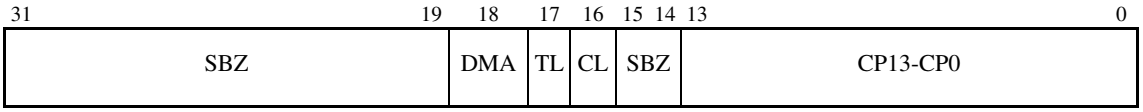
3.2.5 Non-Secure Access Control register

The *Non-Secure Access Control* register (NSAC) defines the Non-secure access permissions to the following:

- coprocessors
- cache lockdown registers
- TLB lockdown registers
- internal DMA (ARMv6).

The NSAC is Read/Write in Secure Privileged modes and Read-Only in Non-secure Privileged modes. Attempting to access this register in User mode (Secure or Non-secure), or writing it in Non-secure Privileged modes, results in an Undefined Instruction exception.

NSAC-bit allocation



The reset value of the NSAC is UNDEFINED. Bits[31:19,15:14] are RESERVED.

Use the following instructions to alter or read the NSAC:

```
MCR CP15, 0, <Rd>, C1, C1, 2 ; moves contents of <Rd> into NSAC
MRC CP15, 0, <Rd>, C1, C1, 2 ; moves contents of NSAC into <Rd>
```

Coprocessor access control bits

Bits[13:0] control whether access to the corresponding coprocessor is permitted in Non-secure world.

For bit *N*:

0 Specifies that Secure access only is permitted to coprocessor *N*. Any attempt to access coprocessor *N* in Non-secure world results in an Undefined Instruction exception.

1 Specifies that Secure or Non-secure access is permitted to coprocessor *N*.

Note

When a coprocessor is defined as Secure, the Non-secure world cannot write the corresponding bits in Coprocessor Access Control Register, and reads them as 0b00, access denied.

If these bits permit access, then the CP15 r1 (CPaccess) is checked for User/Privileged access.

Setting Bits [10] and [11] to different values will have UNPREDICTABLE effects if VFP is implemented.

For the bits which correspond to coprocessors which are not implemented, it is IMPLEMENTATION DEFINED whether the bits behave as RAZ/WI, or can be written by Secure Privileged modes.

CL-bit

The CL-bit controls whether cache lockdown is permitted in Non-secure world:

0 Specifies that cache lockdown entries cannot be allocated in Non-secure world. See *Cache operations* on page 3-32 for details.

1 Specifies that cache lockdown entries can be allocated in Secure or Non-secure world.

TL-bit

The TL-bit controls whether an instruction in Non-secure world can lock page tables in TLB lockdown entries:

0 Specifies that TLB lockdown entries cannot be allocated in Non-secure world.

1 Specifies that TLB lockdown entries can be allocated in Secure or Non-secure world.

Note

Operations *invalidate single entry* or *invalidate on ASID match* can match a TLB lockdown entry. *Invalidate all* operation only applies on unlocked entries. See the *ARM Architecture Reference Manual* for more information on TLB lockdown entries.

DMA-bit

The DMA-bit controls whether access to DMA registers is permitted in Non-secure world:

0 Specifies that DMA registers cannot be used in Non-secure world.

1 Specifies that DMA registers can be accessed in Secure or Non-secure world.

Non-secure page tables are used for address translation when the DMA-bit==1.

3.2.6 Process ID registers

Register 13 holds the Context ID register and the FCSE PID register. They are both banked in Secure and Non-secure world, as shown in Table 3-11. See the *ARM Architecture Reference Manual* for register format.

Table 3-11 Process ID registers

Opcode_1	Opcode_2	Non-secure register	Secure register
0b0000	0b000	Non-secure FCSE PID register	Secure FCSE PID register
0b0000	0b001	Non-secure Context ID register	Secure Context ID register

Note

The use of the FCSE PID register is deprecated.

3.2.7 Register 15

Register 15 is IMPLEMENTATION DEFINED. The functions and access to this register can be restricted to protect secure data according to the system requirement.

IMPLEMENTATION DEFINED registers might be used for non-architectural testing, profiling, or micro-architecture specific support.

3.3 Access to registers in Monitor mode

When the processor is in Monitor mode, the core is in Secure state regardless of the value of the NS-bit in the SCR. In Monitor mode, the NS-bit is used to define whether the Secure CP15 registers or Non-secure CP15 registers are accessed for reading or writing. That is:

NS = 0 Shared, restricted, and the secure banked registers accessed by CP15 register read/write commands.

CP15 operations use the NS-state attribute to determine all resources used, that is, all CP15 based operations performed in Secure state.

NS = 1 Shared and Non-secure banked registers accessed by CP15 register read/write commands.

CP15 operations use the NS-state attribute to determine all resources used, that is, all CP15 based operations performed in Secure state.

3.4 Memory management unit

The TLB entries are marked with an ID specific to Security Extensions, the *Non-Secure Table ID* (NSTID). The NSTID determines whether the entry is associated with a Secure or a Non-secure page table.

When the CPU generates a memory access, the *Memory Management Unit* (MMU) performs a lookup for a mapping for the requested virtual address, current ASID (or global mapping), and NSTID corresponding to the current NS-state in the TLB.

If no matching TLB entry is found then a translation page table walk might be performed using the Secure or Non-secure TTB registers, according to the state of the core. The NSTID of any new entry is updated with the NS-req attribute.

If a matching TLB entry is found then the information it contains is used as described in the Virtual Memory System Architecture chapter in the *ARM Architecture Reference Manual*, including the additional security specific NS-desc attribute.

If no matching TLB entry is found, then one of the following occurs, depending on CP15 r2[5:4] (TTBC):

- a translation page table walk
- an abort.

The corresponding VMSA information is stored in any allocated entry.

3.4.1 L1 descriptors

A Security Extensions specific bit in L1 descriptors, the PTE-NS bit, specifies whether the translated Physical Address targets Secure or Non-secure memory. This bit is used in Secure world only. It is ignored in Non-secure world. Table 3-12 shows where the NS-bit is implemented.

Table 3-12 ARMv6 first level descriptor formats with subpages disabled

	31 24	23 20	19	18	17	16	15 14	12	11 10	9	8 5	4	3	2	1	0
a	Ignored														0	0
b	Coarse page table base address									P	Domain	SBZ	NS	SBZ	0	1
c	Section base address		NS	0	nG	S	APX	TEX	AP	P	Domain	XN	C	B	1	0
d	Supersection base address	SBZ	NS	1	nG	S	APX	TEX	AP	P	Domain	XN	C	B	1	0
e	RESERVED														1	1

a. Translation fault.
b. Course page table.
c. Section (1MB).
d. Supersection (16MB).
e. RESERVED.

Note

The encoding of the PTE-NS bit is the same in ARMv6 first level descriptor formats with subpages enabled, and in backward-compatible first-level descriptor format (ARMv5).

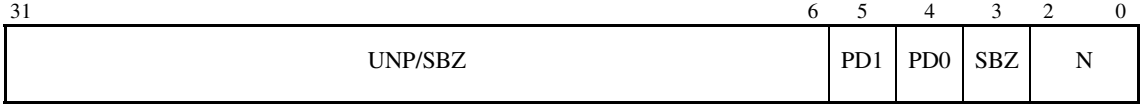
3.4.2 CP15 register 2, Translation Table Base

CP15 register 2 contains the *Translation Table Base* (TTB) registers for the Secure and the Non-secure page tables. Table 3-13 shows these registers.

Table 3-13 TTB registers

Opcode_1	Opcode_2	Non-secure register	Secure register
0b0000	0b000	$TTB0_{NS}$	TTB0
0b0000	0b001	$TTB1_{NS}$ (ARMv6 only)	TTB1 (ARMv6 only)
0b0000	0b010	$TTBC_{NS}$ (Control) (ARMv6 only)	TTBC (Control) (ARMv6 only)

TTBC



Reset value = `0x00000000`.

PD1 and PD0 are Security Extensions specific bits that specify whether page table walks are enabled or disabled.

N Defined in ARMv6.

PD0 = 0 A page table walk is performed on a TLB miss when TTB0 is used. Privilege is Secure or Non-secure according to the current world.

PD0 = 1 If a TLB miss occurs on a TTB0 access, a page table walk is not performed, and a Section Translation Fault is returned.

PD1 = 0 A page table walk is performed on a TLB miss when TTB1 is used. Privilege is Secure or Non-secure according to the current world.

PD1 = 1 If a TLB miss occurs on a TTB1 access, a page table walk is not performed, and a Section Translation Fault is returned.

Note

Setting PD0 == 0 or PD1 == 0 can result in a recursive entry into the abort handler, so effectively deadlocking the system, if the mapping for the abort vectors is not guaranteed to be resident in the TLB (for example, by the use of Lockdown within the TLB).

3.4.3 CP15 register 3, Domain Access Control registers

CP15 register 3 contains the *Domain Access Control Registers* (DACRs). There are separate registers for Secure and Non-secure worlds. Table 3-14 shows these registers.

Table 3-14 DACRs

Opcode_1	Opcode_2	Non-secure register	Secure register
0b0000	0b000	$DACR_{NS}$	DACR

The format is the same in both worlds, as defined in the *ARM Architecture Reference Manual*.

3.4.4 CP15 register 5, Fault Status Registers

CP15 register 5 contains the *Data Fault Status Register* (DFSR) and *Instruction Fault Status Register* (IFSR). These are banked in Secure and Non-secure worlds. Table 3-15 shows these registers.

Table 3-15 DFS and IFS registers

Opcode_1	Opcode_2	Non-secure register	Secure register
0b0000	0b000	$DFSR_{NS}$	DFSR
0b0000	0b001	$IFSR_{NS}$	IFSR

The format is the same in both worlds, as defined in the *ARM Architecture Reference Manual*.

FSR[12], added in conjunction with Security Extensions, is reserved for supporting additional status information on external aborts:

FSR[12]==0 indicates a decode error returned

FSR[12]==1 indicates a slave error returned.

The recommended uses are as follows:

- use decode errors for non-existent memory
- use decode errors for attempted accesses to Secure memory when NS-state==1
- use slave errors when the resource exists, but cannot be successfully read or written by the current transaction.
- use FSR[12] = 0 for all other errors.

See Table 3-16 for recommended encodings, shown in priority order.

Table 3-16 Recommended Security Extensions FSR encodings

FSR [12]	FSR [10]	FSR [3:0]	VMSA	PMSA	Notes
0	0	0b0001	Alignment fault	Alignment fault	-
0	0	0b0000	-	Background fault	-
0	0	0b0100	ICache Maintenance operation fault	-	-
0	0	0b1100	L1 translation external abort	-	DECERR
1	0	0b1100	L1 translation external abort	-	SLVERR
0	0	0b1110	L2 translation external abort	-	DECERR
1	0	0b1110	L2 translation external abort	-	SLVERR
0	0	0b0101	Section Translation fault	-	-
0	0	0b0111	Page Translation fault	-	-
0	0	0b1001	Section Domain fault	-	-
0	0	0b1011	Page Domain fault	-	-
0	0	0b1101	Section Permission fault	Permission fault	-
0	0	0b1111	Page Permission fault	-	-
0	0	0b1000	Precise external abort	Precise external abort	DECERR
1	0	0b1000	Precise external abort	Precise external abort	SLVERR
0	1	0b1010	IMPLEMENTATION DEFINED	IMPLEMENTATION DEFINED	(Coprocessor abort)
0	1	0b0110	Imprecise external abort	Imprecise external abort	DECERR
1	1	0b0110	Imprecise external abort	Imprecise external abort	SLVERR
0	0	0b0110	Precise parity error	Precise parity error	-
0	1	0b1000	Imprecise parity error	Imprecise parity error	-
0	0	0b0010	Debug event	Debug event	-

Note

When the SCR EA bit is set, a Monitor entry due to an external abort writes the Secure DFSR or IFSR.

3.4.5 CP15 register 6, Fault Address registers

CP15 register 6 contains the Fault Address registers. Table 3-17 shows these registers.

Separate *Data* and *Instruction Fault Address Registers*, DFAR and IFAR, are supported. These are banked in the Secure and Non-secure worlds.

Table 3-17 DFA registers

Opcode_1	Opcode_2	Non-secure register	Secure register
0b0000	0b000	$DFAR_{NS}$	DFAR
0b0000	0b001		WFAR (use deprecated)
0b0000	0b010	$IFAR_{NS}$	IFAR

Use of the WFAR in CP15 is deprecated. The function is moved to CP14. See *CP14 register 6, Watchpoint Fault Address Register* on page 4-8 for details.

Note

When the SCR EA bit is set, a Monitor entry due to an external abort writes the Secure DFAR or IFAR.

3.4.6 CP15 register 8, TLB operations

CP15 register 8 provides the TLB operations. The TLB operations are write-only. See the *ARM Architecture Reference Manual* for details.

Note

Invalidate operations have no effect if the NSTID for the entry does not correspond to the current core security state.

3.4.7 CP15 register 10, TLB lockdown registers

CP15 register 10 contains the TLB lockdown register. See the *ARM Architecture Reference Manual* for details of the register format.

If bit 17 (TL) of the NSAC is set, the TLB lockdown entries can be used in Non-secure world. Otherwise, lockdown entries are reserved for use in Secure world only, and a Non-secure access to the TLB lockdown register results in an Undefined Instruction exception.

The Non-secure world can program the lockdown register before the Secure world disables access to the lockdown registers. In this case, the entries that the Non-secure world had locked are still locked. To avoid this, when the Secure world restricts access to lockdown registers to Secure world only, it must flush previous lockdown entries.

3.4.8 CP15 register 12, miscellaneous registers

CP15 register 12 contains the *Vector Base Address Registers* (VBAR) and *Interrupt Status Register* (ISR). See Table 3-18 for details.

Table 3-18 CP15 register 12 registers

Opcode_1	Opcode_2	CRm	Common register	Non-secure register	Secure register
0b0000	0b000	C0		$VBAR_{NS}$	$VBAR_S$
0b0000	0b001	C0			Monitor VBAR (MVBAR)
0b0000	0b000	C1	ISR		

Secure and Non-secure Vector Base Address Registers

The *Vector Base Address Register* (VBAR) is banked in the Secure and Non-secure worlds.

When an exception branches to a Secure Privileged mode, the core branches to:

`Secure_Vector_Base_Address + Exception_Vector_Address.`

When an exception branches to a Non-secure Privileged mode, the core branches to:

`Non_Secure_Vector_Base_Address + Exception_Vector_Address.`

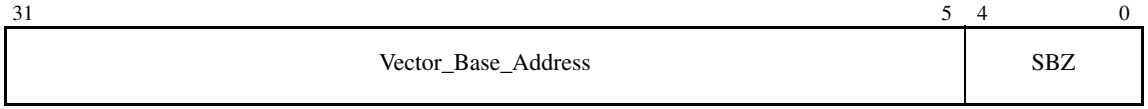
If high vectors are enabled for the target security state, `Vector_Base_Address` is treated as being `0xFFFF0000`, regardless of the value of the VBAR.

Both VBARs can only be accessed in Privileged modes. An attempt to access a VBAR in User mode results in an Undefined Instruction exception. The NS-bit defines whether $VBAR_S$ or $VBAR_{NS}$ is accessed.

The secure Vector Base Address Register ($VBAR_S$) resets to 0, the non-secure version ($VBAR_{NS}$) is UNDEFINED. Base addresses must be programmed as part of the boot sequence. The 5 lowest bits are defined by the exception offset (Exception_Vector_Address).

Use the following instructions to read or write the VBARs:

```
MCR CP15, 0, <Rd>, C12, C0, 0 ; moves contents of <Rd> into VBAR
MRC CP15, 0, <Rd>, C12, C0, 0 ; moves contents of VBAR into <Rd>
```



Reset value = `0x00000000` ($VBAR_S$ only, $VBAR_{NS}$ is UNDEFINED).

Monitor Vector Base Address Register

The *Monitor Vector Base Address Register* (MVBAR) exists only in the Secure world.

When an exception branches to Monitor mode, the core branches to:

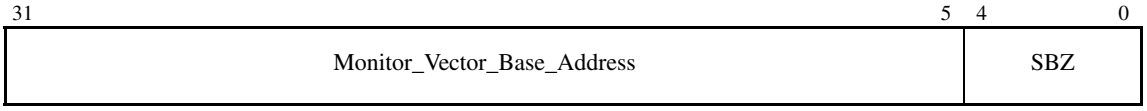
```
Monitor_Base_Address + Exception_Vector_Address.
```

The MVBAR can only be accessed in Secure Privileged modes. An attempt to access the MVBAR in User mode or any Non-secure state results in an Undefined Instruction exception.

The Monitor Vector Base Address Register (MVBAR) is UNDEFINED on reset. The base address must be programmed as part of the boot sequence. The 5 lowest bits are defined by the exception offset (Exception_Vector_Address).

Use the following instructions to read or write the MVBAR:

```
MCR CP15, 0, <Rd>, C12, C0, 1; moves contents of <Rd> into MVBAR
MRC CP15, 0, <Rd>, C12, C0, 1; moves contents of MVBAR into <Rd>
```



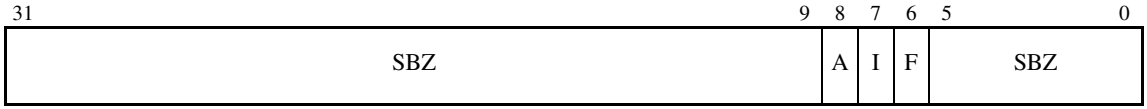
Reset value = UNDEFINED.

Interrupt status Register

The *Interrupt Status Register* (ISR) shows whether there is a pending IRQ, FIQ or External Abort. The three status bits are mapped in the same way as in the CPSR. This allows the same masks to be used to get the value.

Use the following instructions to read the ISR:

```
MRC CP15, 0, <Rd>, C12, C1, 0 ; moves contents of ISR into <Rd>
```



F = 0 No pending FIQ.

F = 1 Pending FIQ.

I = 0 No pending IRQ.

I = 1 Pending IRQ.

A = 0 No pending external abort.

A = 1 Pending external abort.

Note

The F and I bits directly reflect the state of the **FIQ** and **IRQ** inputs respectively. For external aborts, the A bit is set when an external abort occurs and is automatically cleared when the abort is taken.

3.5 L1 caches

This section contains the following subsections:

- *Virtual L1 caches*
- *Physical L1 caches*
- *CP15 register 7, cache management functions* on page 3-31
- *CP15 register 7, Virtual Address to Physical Address translation* on page 3-35.

3.5.1 Virtual L1 caches

If the virtual to physical mapping in Secure and Non-secure worlds are different, virtually tagged L1 caches need to be invalidated on each switch between Secure and Non-secure worlds. A data or unified cache must be cleaned first if necessary.

Otherwise, the behavior of virtual L1 caches is unchanged.

3.5.2 Physical L1 caches

The entries in the L1 cache are marked with a Secure line tag. This tag defines the entries as containing Secure or Non-secure data. A Secure entry contains Secure data. A Non-secure entry contains Non-secure data.

This tag can be considered as an additional physical address bit when cache lookups are performed.

The Security Extensions L1 cache behaves as follows:

- The entries in the cache need not be cleaned or invalidated by software for different Secure and Non-secure virtual to physical mappings.
- When the core is in Non-secure world, cache lookups are performed on lines tagged as Non-secure (NS-tag=1). The NS-desc attribute in the corresponding L1 page table entry is ignored.
- When the core is in Secure world, cache lookups are performed on lines marked as Non-secure if NS-attr==1.
- When the core is in Secure world, cache lookups are performed on lines marked as Secure if NS-attr=0.
- Because NS-desc is allocated in L1 page table entries, the granularity of Secure and Non-secure regions within a virtual address map is limited to 1MB. The physical mapping can still have a granularity of 4KB.
- Apart from Invalidate All, every Clean, Invalidate, and Clean and Invalidate operation applies to Non-secure entries when executed from the Non-secure world. Invalidate All applies to all entries for Icache if CL =1. See *Cache operations, Non-secure world* on page 3-33 for details.
- Clean, Invalidate, and Clean and Invalidate operations apply to both Secure and Non-secure entries when executed from the Secure world. See *CP15 register 7, cache management functions* on page 3-31 for details.

- For caches with more than 4KB per way, VA[13:12] must be the same for all aliases used in both Secure and Non-secure worlds.

In other respects, for example cache eviction rules, the behavior of Security Extensions L1 caches is the same as ARMv6.

3.5.3 CP15 register 7, cache management functions

Table 3-19 shows the CP15 register 7 registers.

CP15 register 7 provides the cache and write buffer operations. In addition to these operations, virtual to physical address features are provided, see *CP15 register 7, Virtual Address to Physical Address translation* on page 3-35.

The existing ARMv6 support is all common to Secure and Non-secure worlds, except the cache dirty status register.

Table 3-19 CP15 register 7 registers

Opcode_1	Opcode_2	CRm	Common register	Non-secure register	Secure register
0b0000	0b000	C0	Wait for interrupt	-	-
0b0000	0b000	C4	See *CP15 register 7, Virtual Address to Physical Address translation* on page 3-35	-	-
0b0000	0b001	C5	Invalidate entire instruction cache	-	-
0b0000	0b001	C5	Invalidate instruction cache line using MVA	-	-
0b0000	0b010	C5	Invalidate instruction cache line using index	-	-
0b0000	0b100	C5	Flush prefetch buffer	-	-
0b0000	0b110	C5	Flush entire branch target cache	-	-
0b0000	0b111	C5	Flush branch target cache entry	-	-
0b0000	0b000	C6	Invalidate entire data cache	-	-
0b0000	0b001	C6	Invalidate data cache line using MVA	-	-
0b0000	0b010	C6	Invalidate data cache line using index	-	-
0b0000	0b000	C7	Invalidate both caches	-	-

Table 3-19 CP15 register 7 registers (continued)

Opcode_1	Opcode_2	CRm	Common register	Non-secure register	Secure register
0b0000	Any	C8	See *CP15 register 7, Virtual Address to Physical Address translation* on page 3-35	-	-
0b0000	0b000	C10	Clean entire data cache	-	-
0b0000	0b001	C10	Clean data cache line using MVA	-	-
0b0000	0b010	C10	Clean data cache line using index	-	-
0b0000	0b100	C10	DataSynchronizationBarrier	-	-
0b0000	0b101	C10	DataMemoryBarrier	-	-
0b0000	0b110	C10	-	Cache dirty status register	Cache dirty status register
0b0000	0b100	C12	Block transfer status register	-	-
0b0000	0b101	C12	Stop prefetch range	-	-
0b0000	0b001	C13	Prefetch instruction cache line	-	-
0b0000	0b000	C14	Clean and Invalidate entire data cache	-	-
0b0000	0b001	C14	Clean and Invalidate data cache line using MVA	-	-
0b0000	0b010	C14	Clean and Invalidate data cache line using index	-	-

Cache operations

The operations are affected by the NS-tag attribute in the cache entries. The modifications are described in the following sections:

- *Cache operations, Non-secure world* on page 3-33
- *Cache operations, Secure world* on page 3-34
- *Cache dirty status register* on page 3-34.

Cache operations, Non-secure world

In the Non-secure world, the Clean, Invalidate, and Clean and Invalidate operations only affect Non-secure lines whatever the operations are (by set/way, by MVA or for all cache lines). Any attempt to access Secure lines is ignored (for example, an attempt to invalidate a Secure line by set/way).

- For the following operations, only Non-secure lines are affected:
  - Clean All, Clean by set/way, Clean by MVA
  - Clean and Invalidate All, Clean and Invalidate by set/way, Clean and Invalidate by MVA

  Attempts to Clean or Clean and Invalidate Secure lines are IGNORED.
- For Invalidate by set/way or by MVA, only Non-secure lines are affected. Attempts to Invalidate Secure lines are IGNORED.
- For Dcache, Invalidate All operations cause an Undefined Instruction exception. This prevents any entries marked as Secure being invalidated. Otherwise, the Non-secure world could corrupt Secure dirty data.
- For Icache, Invalidate All operations cause an Undefined Instruction exception if lockdown entries are reserved for the Secure world (CL = 0). Otherwise this operation affects all entries, Secure and Non-secure.

Caution

Lockdown, Clean, Invalidate, and Clean and Invalidate operations apply whether the entry is locked or not.

Cache operations, Secure world

In the Secure world, the Clean, Invalidate, and Clean and Invalidate operations can affect both Secure and Non-secure lines:

- For Clean All, Invalidate All, and Clean and Invalidate All, operations, all lines are affected.

  Caution

  The programmer must ensure that removal of dirty or locked data, whether Secure or Non-secure, does not cause a problem.

- For Clean by index, Invalidate by index, and Clean and Invalidate by index, the selected line is affected regardless of the NS-tag.
- For Clean by MVA, Invalidate by MVA, and Clean and Invalidate by MVA:
  - if the virtual address is marked as Non-secure in the page table (NS-attr==1), only Non-secure entries are affected
  - if the virtual address is marked as Secure in the page table (NS-attr==0), only Secure entries are affected.

Caution

Clean, Invalidate, and Clean and Invalidate operations apply whether the entry is locked or not.

Cache dirty status register

The Cache Dirty Status Register (CDSR) is banked. This is because Non-secure Clean, Invalidate, and Clean and Invalidate operations only clean or invalidate Non-secure lines. When Non-secure lines are all clean, the Non-secure CDSR indicates no dirty lines, even if there are dirty Secure lines. The behavior of the CDSRs is as follows:

- Non-secure Clean All, Invalidate All, and Clean and Invalidate All operations clear the Non-secure CDSR.
- Secure Clean All, Invalidate All, and Clean and Invalidate All operations clear both the Secure CDSR and the Non-secure CDSR.
- The Non-secure CDSR is set on stores to a Non-secure line (that is, when the core is in Non-secure world, or is in Secure world and targets a Non-secure line) that write a dirty bit in the cache.
- The Secure CDSR is set on any stores that write a dirty bit in the cache.

3.5.4 CP15 register 7, Virtual Address to Physical Address translation

In addition to the existing cache and write buffer operations registers, CP15 register 7 contains *Virtual Address* (VA) to *Physical Address* (PA) operation registers. Table 3-20 shows these registers.

Table 3-20 CP15 register 7 VA to PA translation registers

Opcode_1	Opcode_2	CRm	Common register	Non-secure register	Secure register
0b0000	0b000	C4	-	PA register	PA register
0b0000	0b000	C8	Current world, Privileged read	-	-
0b0000	0b001	C8	Current world, Privileged write	-	-
0b0000	0b010	C8	Current world, User read	-	-
0b0000	0b011	C8	Current world, User write	-	-
0b0000	0b100	C8	-	-	Other world, Privileged read
0b0000	0b101	C8	-	-	Other world, Privileged write
0b0000	0b110	C8	-	-	Other world, User read
0b0000	0b111	C8	-	-	Other world, User write

VA to PA translation in the current world

A write to the VA to PA translation register translates the VA provided by the general-purpose register <Rd> and stores the corresponding PA in the PA register. The operation is performed with the current virtual mapping, Secure or Non-secure.

If the MMU is disabled for the current world, the MMU returns the address and attributes in the same way as described for a disabled MMU in the ARM Architecture Reference Manual.

The VA to PA translation can only be performed in current Privileged modes. All VA to PA translation operations (CRm=8) are Write-Only operations. The PA result register can be read or written.

Examples of VA to PA operations:

```
MCR CP15, 0, <Rd>, C7, C8, 3 ; VA(Rn) to PA with User Write permission
MRC CP15, 0, <Rd>, C7, C4, 0 ; PA to Rn
```

Note

The VA transferred is the true VA, not the MVA. It is subject to the VA to MVA conversion of the FCSE mechanism.

VA to PA translation in the other world

In the Secure world, a VA to PA translation in the other (Non-secure) world performs the translation as if the current world was Non-secure. It uses the Non-secure translation resources (for example, TTBR-NS) and Non-secure page table to translate the specified VA to a PA. The result of the operation is captured in the Secure PA register, not the Non-secure PA register, even though the result is a Non-secure translation.

If the MMU is disabled for the other world, the MMU returns the address and attributes in the same way as the MMU disabled behavior described in the ARM Architecture Reference Manual.

Other world always means Non-secure world, because the Non-secure world cannot access the Secure world. If the Non-secure world attempts to run a VA to PA translation on the other world, an Undefined Instruction exception is generated.

The VA to PA translation can only be performed in current privileged modes. All VA to PA translation operations (CRm=8) are Write-Only operations. The PA result register can be read or written.

For these operations, NS-req=1.

Example of VA to PA operation:

```
MCR CP15, 0, <Rd>, C7, C8, 4 ; VA(Rn) to PA with NS Privileged Read permission
MRC CP15, 0, <Rd>, C7, C4, 0 ; PA to Rn
```

Note

The VA transferred is the true VA, not the MVA. It is subject to the VA to MVA conversion of the FCSE mechanism.

VA to PA Translation when the MMU is Disabled

VA to PA operations occur even when the MMU on the relevant world is disabled. They report the flat address mapping and the MMU-disabled value of the attributes and permissions for the data side accesses. These include any MMU-disabled re-mapping specified by the TEX-remap facilities. The SuperSection bit is 0 when the MMU is disabled.

PA register

The *PA register* (PAR) of the current world receives the PA during any VA to PA translation.

The PAR format depends on the value of bit[0]. Bit[0] specifies whether the operation has completed successfully or not.

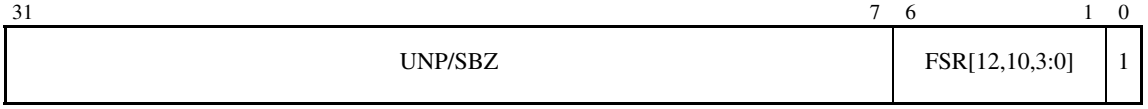
The VA to PA translation only generates an abort if the translation fails because an external abort occurred on a page table walk request. In this case, the PAR is updated. The DFSR and DFAR of the world that the abort is handled in are also updated. If the EA bit in the SCR is set, this abort is trapped to Monitor mode.

For all other cases where the VA to PA translation fails, only the PAR is updated with the FSR encoding and bit[0] set. The DFSR and DFAR remain unchanged and no abort is generated.

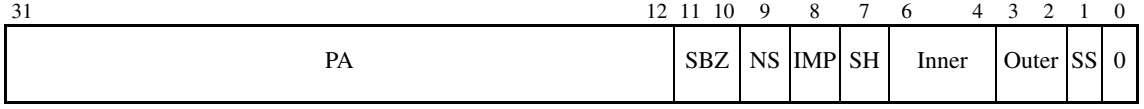
Caution

If the EA bit is set (CP15r1 SCR[3]==1), the operation is being performed with NS-state = 1 (Non-secure) and an external abort occurs during a page table walk associated with an address translation, PAR_{NS} (not PAR_S) and $DFSR_S$/$DFAR_S$ are updated with the external abort information.

If the translation has aborted, bits[6:1] give the encoding of the source of the abort:



If the translation has completed successfully, the PAR has the following encoding:



To read or write the PAR, use the following instructions:

```
MCR CP15, 0, <Rd>, C7, C4, 0
MRC CP15, 0, <Rd>, C7, C4, 0
```

The PAR write instruction is provided to allow the PAR to be context switched.

The register provides the translated PA, and the following information from the page table:

NS Non-Secure.

IMP Implementation Defined.

SH Shareable attribute. In some cases this differs from the S bit in the page tables.

SS SuperSection. Used to encode supersections:

- **0b0** Page was not a supersection, that is, PAR[31:12] are PA[31:12], regardless of the page size.

0b1 Page was part of a supersection, that is, PAR[31:12] contains: {PA[31:24], PA[39:32], 0b0000}.

Inner[2:0], inner attributes

0b111 Write back, no write allocate.

0b110 Write through.

0b101 Write back, write allocate.

0b011 Device.

0b001 Strongly ordered.

0b000 Non-cacheable.

Other Inner encodings are RESERVED.

Outer[1:0], outer attributes

0b11 Write back, no allocate on write.

0b10 Write through, no allocate on write.

0b01 Write back, allocate on write.

0b00 Non-cacheable.

Implementations that do not support all attributes can report the behavior for those memory types that the cache does support.

3.5.5 CP15 register 9, cache lockdown and cache behavior override registers

CP15 register 9 contains the *Cache Lockdown Registers* (CLRs) and *Cache Behavior Override Register* (CBOR). Table 3-21 shows these registers.

Table 3-21 CP15 register 9, cache lockdown and cache behavior override registers

Opcode_1	Opcode_2	CRm	Common register
0b0000	0b000	C0	DCLR (Format C)
0b0000	0b001	C0	ICLR (Format C)
0b0000	0b000	C8	CBOR [a]

a. Some bits are only accessible in Secure state, see *Cache behavior override register* on page 3-41.

Cache lockdown registers

The *Data CLR* (DCLR) and *Instruction CLR* (ICLR) are common to the Secure and the Non-secure worlds.

If the NSAC bit[16] (CL) is set, the Non-secure world can lock cache ways. Otherwise attempting to access the CLRs in Non-secure state causes an Undefined Instruction exception.

The Non-secure world can program the CLRs and then let the Secure world disable access to the CLRs. In this case, the entries that the Non-secure world locked are still locked. To avoid this, when the Secure world restricts access to the CLRs to the Secure world, it must clean and invalidate any previously locked (NS-tag=1) entries.

Cache behavior override register

It is sometimes necessary to ensure that the cache content is not modified when it is debugged, or when an interruptible cache operation is being processed.

For example, Clean All , and Clean and Invalidate All operations in Non-secure world might not prevent fast interrupts to the Secure side if the FW-bit in the SCR is clear. In this case, the Secure world can read or write Non-secure locations in the cache. This can potentially cause the cache to contain valid or dirty Non-secure entries once the Non-secure Clean All , and Clean and Invalidate All operation has completed. To avoid such problems, the Secure world must not be allowed to allocate Non-secure entries into the cache (it must disable linefill) and must treat all writes to Non-secure regions that hit in the cache as being write-though.

Three bits for each world are provided to prevent cache refill and force write-through operations while keeping the caches themselves enabled. The Non-Secure bits apply to Non-secure regions, and can be accessed in the Secure or in the Non-secure world. The Secure bits apply to Secure regions, and can only be accessed in Secure world .

The format of the *Cache Behavior Override Register* (CBOR) is as follows:

31 … 6	5	4	3	2	1	0
SBZ	S_WT	S_IL	S_DL	NS_WT	NS_IL	NS_DL

Reset value: `0x00000000`.

To read or write the CBOR, use the following instructions:

```
MCR CP15, 0, <Rd>, C9, C8, 0
MRC CP15, 0, <Rd>, C9, C8, 0
```

Bits[5:3] are only accessible in Secure Privileged modes. In Non-secure Privileged modes, write has no effect and reads read 0.

Any attempt to access the CBOR in User mode (Secure or Non-secure) results in an Undefined Instruction exception.

The meanings of the bits in the CBOR are as follows:

S_WT	**0**	do not force Write-Through for regions marked Secure (NS-attr=0) and Write-Back (normal operation)
	1	force Write-Through for regions marked Secure (NS-attr=0) and Write-Back
S_IL	**0**	enable instruction cache linefill for regions marked Secure (normal operation)
	1	disable instruction cache linefill for regions marked Secure
S_DL	**0**	enable data cache linefill for regions marked Secure (normal operation)
	1	disable data cache linefill for regions marked Secure

NS_WT	**0**	do not force Write-Through for regions marked Non-secure (NS-attr=1) and Write-Back (normal operation)
	1	force Write-Through for regions marked Non-secure (NS-attr=1) and Write-Back
NS_IL	**0**	enable instruction cache linefill for regions marked Non-secure (normal operation)
	1	disable instruction cache linefill for regions marked Non-secure
NS_DL	**0**	enable data cache linefill for regions marked Non-secure (normal operation)
	1	disable data cache linefill for regions marked Non-secure.

3.6 Tightly coupled memory

This section applies only to ARMv6 systems.

ARMv6 defines a *Tightly Coupled Memory* (TCM) Status Register with up to four TCM Instruction Regions, and up to four TCM data regions.

Each TCM region has its own Region register, defining:

- the base address of the region
- the size of the region
- whether the region is enabled or disabled
- whether the region behaves as *smartcache* or local RAM.

To ensure the security of the data in the TCM:

- The entries in the TCM regions do not need to be cleaned or invalidated by software for different Secure and Non-secure virtual addresses.
- TCM regions containing Secure data are under control of the Secure kernel only.
- TCM regions controlled by Privileged modes in both worlds are visible:
  - in Non-secure world
  - in Secure world if the corresponding L1 TLB descriptor is marked as Non-secure (NS-attr==1).

3.6.1 CP15 register 9: TCM regions, TCM selection, and region control registers

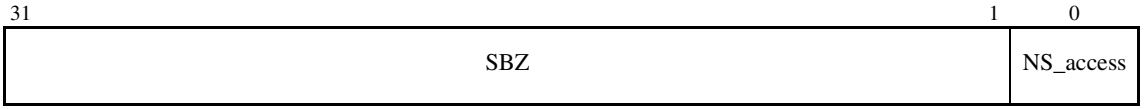
For each region, CP15 register 9 contains a TCM Region Register (TCMRR) and a TCM Non-Secure Access Control Register (TCM-NSAC). The *TCM Selection Register* (TCMSR) indicates the number of the TCM region that the TCMRR and the TCM-NSAC apply to.

Table 3-22 CP15 register 9: TCM regions, TCM selection, and region control registers

Opcode_1	Opcode_2	CRm	Common register	Non-secure register	Secure register
0b0000	0b000	C1	Data TCMRR		
0b0000	0b001	C1	Instruction or Unified TCMRR		
0b0000	0b010	C1			Data TCM-NSAC
0b0000	0b011	C1			Instruction or Unified TCM-NSAC
0b0000	0b000	C2		TCMSR	TCMSR

Instruction or Unified TCM-NSAC

The Instruction or unified TCM-NSAC ({I}TCM-NSAC) defines the accessibility of the associated TCMRR, selected by the TCMSR. It also defines whether instructions stored in the TCM region are Secure or Non-secure.



Reset value: `0x00000000`.

If `NS_access` = 0 (TCM with Secure data), a Non-secure access to the TCMRR causes an Undefined Instruction exception.

To access the instruction TCM-NSAC, use one of the following instructions:

```
MCR CP15, 0, Rd, C9, C1, 3
MRC CP15, 0, Rd, C9, C1, 3
```

NS_access = 0 The instruction TCMRR is accessible in Secure Privileged modes only. The data stored in the TCM is Secure. The TCM is only visible in the Secure world, and only if the page table is correctly marked as Secure (NS-attr==0).

NS_access = 1 The instruction TCMRR is accessible in all Privileged modes. The data stored in the TCM is Non-secure. The TCM is visible in the Non-secure world, and also in the Secure world if the page table is correctly marked as Non-secure (NS-attr==1).

Table 3-23 shows when the TCM is visible, and what data is visible.

Table 3-24 on page 3-45 shows when control of the TCM is available.

Table 3-23 TCM region visibility

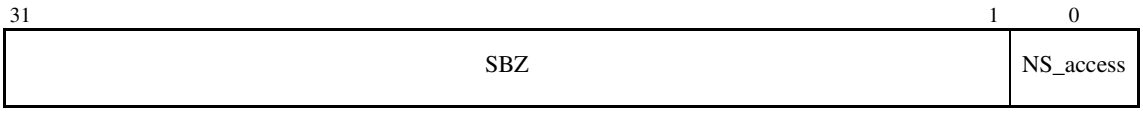
Core state	NS_access	NS-desc	Region visibility	Data visible
Secure	0	0	Visible	Secure
Secure	0	1	Not visible	-
Secure	1	0	Not visible	-
Secure	1	1	Visible	Non-secure
Non-secure	0	x	Not visible	-
Non-secure	1	x	Visible	Non-secure

Table 3-24 TCM region control

Core state	NS_access	Control
Secure	x	In Privileged modes only
Non-secure	0	No access
Non-secure	1	In Privileged modes only

Data TCM-NSAC

The Data TCM-NSAC (DTCM-NSAC) defines the accessibility of the associated TCMRR of the TCM region selected by the TCMSR, and defines whether data stored in the TCM region is Secure or Non-secure. The DTCM-NSAC can be accessed in Secure Privileged modes only. Attempting to access this register in any other mode causes an Undefined Instruction exception.



Reset value: `0x00000000`.

If `NS_access` = 0 (TCM with Secure data), a Non-secure access to the TCM region register causes an Undefined Instruction exception.

To access the DTCM-NSAC, use one of the following instructions:

```
MCR CP15, 1, Rd, C9, C1, 2
MRC CP15, 1, Rd, C9, C1, 2
```

NS_access = 0 The data TCMRR is accessible in Secure Privileged modes only. The data stored in the TCM is Secure. The TCM is only visible in the Secure world, and only if the page table is correctly marked as Secure (NS-attr==0).

NS_access = 1 The data TCMRR is accessible in all Privileged modes. The data stored in the TCM is Non-secure. The TCM is visible in the Non-secure world, and also in the Secure world if the page table is correctly marked as Secure (NS-attr==1).

3.7 L1 DMA

This section applies only to ARMv6 systems.

The ARMv6 L1 DMA mechanism can be used with both the Secure and Non-secure worlds. In the Non-secure world, read and write access to these registers depends on bit[18] in the NSAC (NS-dma). Any attempt to access these registers in Non-secure world when NS-dma==0 causes an Undefined Instruction exception. See *Non-Secure Access Control register* on page 3-16 for additional information.

NS-dma defines the security status for a DMA access:

- NS-dma determines which set of page tables to use for address translation, and is compared with NSTID for any associated TLB hits.
- When NS-dma==1, all memory and TCM accesses are restricted to the Non-secure world.
- When NS-dma==0:
  - — the secure page tables are used (NSTID = 0),
  - — the TCM and memory transfer addresses use their respective value of NS-attr to control the access to the memory.
- NS-prot supports external accesses.

The DMA architecture supports generation of interrupts on error or on completion. Separate interrupts for signaling the Secure or Non-secure world dependent on the value of NS-dma are supported. In addition, external aborts are routed to a unique interrupt, DMAEXTERRIRQ. This allows control of when the abort is handled by the core.

Note

DMAEXTERRIRQ is non-maskable. The IE-bit in the DMA control register is IGNORED.

The L1 DMA is controlled using CP15 r11. This register is common to the Secure and Non-secure worlds.

Chapter 4
Debug Architecture

This chapter describes features added to the ARM® Debug architecture as part of Security Extensions. It contains the following sections:

4.1 Overview of Security Extensions debug

ARM processors with Security Extensions implement two types of debug support:

Invasive debug All debug features that allow modification of the processor state.

Non-invasive debug All debug features that allow data and program flow observation, especially trace support.

Security Extensions debug allows you to disable invasive debug and non-invasive debug independently in either:

- all Secure modes
- only in Secure Privileged modes.

This is controlled by the existing *Debug Enable* (**DBGEN**) input signal, together with three Security Extensions specific input signals, *Non-Invasive Debug Enable* (**NIDEN**), *Secure Privileged Invasive Debug Enable* (**SPIDEN**) and *Secure Privileged Non-Invasive Debug Enable* (**SPNIDEN**). See *External debug interface* on page 4-14 for details.

4.2 CP14 register 0, Debug ID Register

The *Debug ID Register* (DIDR) provides information about the Debug Architecture Version.

Table 4-1 DIDR values

Bits	Core view	External view	Value	Meaning
19:16	R	R	0x2	Debug Architecture Version v6.1
15:12	R	R	0x1	Security Extension features implemented

4.3 CP14 register 1, Debug Status and Control Register

The *Debug Status and Control Register* (DSCR) contains status and configuration information about the state of the debug system.

Security Extensions defines five additional bits, and alters the behavior of four others. Table 4-2 provides an overview. The following sections provide details.

Table 4-2 New and altered bits in CP14 register 1

Bits	New or altered	Core view	External view	Reset value	Description
19	New	R	R	0	Imprecise Data Aborts ignored
18	New	R	R	0	Non-secure Status bit
17	New	R	R	n/a	Not *Secure Privilege Non-Invasive Debug Enable* (SPNIDEN)
16	New	R	R	n/a	Not *Secure Privilege Invasive Debug Enable* (SPIDEN)
11	Altered	R	RW	0	Interrupts disable
8	New	R	RC	0	Sticky Undefined bit
7	Altered	R	RC	0	Sticky Imprecise Data Abort
6	Altered	R	RC	0	Sticky Precise Data Abort
5:2	Altered	RW	R	-	Method of Debug Entry

4.3.1 Imprecise Data Aborts Ignored

This bit indicates whether a *DataSynchronizationBarrier* (DSB) operation has occurred in Debug state:

0 No DSB operation has occurred since entering Debug state.

1 A DSB operation has occurred since entering Debug state.

Imprecise data aborts are not taken when this bit is set. See *Imprecise data aborts in detail* on page 4-23 for further information.

4.3.2 Non-secure status bit

This bit indicates whether the processor is in Secure state or Non-secure state:

0 The processor is in Secure state. That is, either the CP15 NS-bit = 0 or the processor is in Monitor mode.

1 The processor is in Non-secure state. That is, the CP15 NS-bit = 1 and the processor is not in Monitor mode.

See *NS-bit* on page 3-12 for further information.

4.3.3 SPNIDEN

DSCR[17] reflects the inverse of the value on the **SPNIDEN** input:

0 The **SPNIDEN** input is HIGH.

1 The **SPNIDEN** input is LOW.

See *SPNIDEN* on page 4-15 for further information.

4.3.4 SPIDEN

DSCR[16] reflects the inverse of the value on the **SPIDEN** input:

0 The **SPIDEN** input is HIGH.

1 The **SPIDEN** input is LOW.

See *SPIDEN* on page 4-14 for further information.

4.3.5 Interrupts disable

This bit controls whether IRQ and FIQ input signals are permitted when Debug is enabled:

0 interrupts enabled

1 interrupts disabled.

This bit has no effect if DSCR[15:14] = 0b00 (Debug disabled) or if **DBGEN** is LOW.

4.3.6 Sticky Undefined bit

This bit indicates whether an Undefined Instruction exception has occurred, in Debug state, since an external read to the DSCR:

0 No Undefined Instruction exception occurred since the last time this bit was cleared.

1 An Undefined Instruction exception occurred since the last time this bit was cleared.

Reads by an external debugger to the DSCR clear the Sticky Undefined bit.

Note

This bit is only set by Undefined Instruction exceptions that occur while the processor is in Debug state.

4.3.7 Sticky Imprecise Data abort

This bit is now set when the processor detects an Imprecise Data Abort returned by the memory system, only if the processor is in Debug state.

In earlier versions of the ARM architecture, it is set when the processor takes an Imprecise Data Abort exception, whether or not the processor is in Debug state.

This bit is cleared after reads.

4.3.8 Sticky Precise Data abort

This bit is now set by Precise Data Aborts, only if the processor is in Debug state.

In earlier versions of the ARM architecture, it is set by Precise Data Abort exception, whether or not the processor is in Debug state.

4.3.9 Method of Debug Entry

Table 4-3 shows the meanings of the method of debug entry values.

Table 4-3 Meaning of method of debug entry values

Value	Description
0b0000	A Debug state Entry Request command occurred.
0b0001	Breakpoint occurred.
0b0010	Watchpoint occurred.
0b0011	`BKPT` instruction occurred.
0b0100	External Debug Request signal activation occurred.
0b0101	Vector catch occurred.
0b0110	RESERVED. (Formerly D-side abort occurred.)
0b0111	RESERVED. (Formerly I-side abort occurred.)
0b1xxx	RESERVED.

4.4 CP14 register 6, Watchpoint Fault Address Register

The use of the *Watchpoint Fault Address Register* (WFAR) mapped in CP15 register 6 is UNPREDICTABLE in Security Extensions. Use the *Watchpoint Fault Address Register* (WFAR) in CP14 instead. Table 4-4 shows the location of the WFAR in CP14.

Table 4-4 WFAR

Opcode_1	Opcode_2	CRn	CRm	Secure register
0b0000	0b000	C0	C6	WFAR

Use the following instructions to read or write the WFAR:

```
MCR CP14, 0, <Rd>, C0, C6, 0 ; moves contents of <Rd> into WFAR

MRC CP14, 0, <Rd>, C0, C6, 0 ; moves contents of WFAR into <Rd>
```

The WFAR is accessible as follows:

External view No access.

Core view The WFAR is accessible in Privileged modes or in Debug state only.

4.5 CP14 register 7, Vector Catch Register

Table 4-5 shows the *Vector Catch Register* (VCR).

The table uses the following abbreviations:

VBAR Vector Base Address Register (Secure).

$VBAR_{NS}$ Vector Base Address Register (Non-secure).

MVBAR Monitor Vector Base Address Register.

If a bit in the VCR is set, when the corresponding vector is pre-fetched and the instruction is committed for execution, either a Debug exception or a Debug State entry can be generated.

Note

Under this model, any kind of pre-fetch of an exception vector can trigger a vector trap, not just those due to exception entries.

Catches due to bits[15:0] are only triggered in Secure world. Catches due to bits [31:25] are only triggered in Non-secure world.

Bits[7:6,4:0] are mandatory. Bits[31,30,28:25,15,14,12,10] are optional for Security Extension compliant cores.

Bits[28,27,12,4,3] do not generate a Debug exception in Privileged modes in Monitor debug-mode. This avoids the processor entering an unrecoverable state.

Note

The VCR can only be accessed in Privileged modes or in Debug state.

Table 4-5 VCR

Bits	Read/write attributes	Reset value	Vector trap enable	World	Normal address	High vector address
0	RW	0	Reset	-	0x00000000	0xFFFF0000
1	RW	0	Undefined	Secure	VBAR+0x00000004	0xFFFF0004
2	RW	0	SWI	Secure	VBAR+0x00000008	0xFFFF0008
3	RW	0	Prefetch abort	Secure	VBAR+0x0000000C	0xFFFF000C

Table 4-5 VCR (continued)

Bits	Read/write attributes	Reset value	Vector trap enable	World	Normal address	High vector address
4	RW	0	Data abort	Secure	VBAR+0x00000010	0xFFFF0010
5	DNM/RAZ	-	RESERVED	-	-	-
6	RW	0	IRQ VE = 0	Secure	VBAR+0x00000018 [a]	0xFFFF0018 [a]
7	RW	0	FIQ VE = 0	Secure	VBAR+0x0000001C [b]	0xFFFF001C [b]
9:8	DNM/RAZ	-	RESERVED	-	-	-
10	RW	0	SMI	Secure	MVBAR+0x00000008	MVBAR+0x00000008
11	DNM/RAZ	-	RESERVED	-		
12	RW	0	Data abort	Secure	MVBAR+0x00000010	MVBAR+0x00000010
13	DNM/RAZ	-	RESERVED	-	-	-
14	RW	0	IRQ	Secure	MVBAR+0x00000018	MVBAR+0x00000018
15	RW	0	FIQ	Secure	MVBAR+0x0000001C	MVBAR+0x0000001C
24:16	DNM/RAZ	-	RESERVED	-	-	-
25	RW	0	Undefined	Non-secure	$VBAR_{NS}$+0x00000004	0xFFFF0004
26	RW	0	SWI	Non-secure	$VBAR_{NS}$+0x00000008	0xFFFF0008
27	RW	0	Prefetch abort	Non-secure	$VBAR_{NS}$+0x0000000C	0xFFFF000C
28	RW	0	Data abort	Non-secure	$VBAR_{NS}$+0x00000010	0xFFFF0010

Table 4-5 VCR (continued)

Bits	Read/write attributes	Reset value	Vector trap enable		World	Normal address	High vector address
29	DNM/RAZ	-	RESERVED		-	-	-
30	RW	0	IRQ	VE = 0	Non-secure	$VBAR_{NS}$+0x00000018 [c]	0xFFFF0018 [c]
31	RW	0	FIQ	VE = 0	Non-secure	$VBAR_{NS}$+0x0000001C [d]	0xFFFF001C [d]

a. If VE = 1, most recent Secure IRQ address
b. If VE = 1, most recent Secure FIQ address
c. If VE = 1, most recent Non-secure IRQ address
d. If VE = 1, most recent Non-secure FIQ address

4.6 CP14 registers 80-95: Breakpoint Control Registers

Security Extensions extends the *Breakpoint Control Register*s (BCRs) with two bits. Table 4-6 shows these bits. For more details about existing bits in the BCRs, see the *ARM Architecture Reference Manual*.

Bits BCR[15:14] control whether breakpoints are conditional on the processor state (Secure or Non-secure).

Note

The BCRs can only be accessed in Privileged modes or in Debug state.

Table 4-6 BCR

Bits	Read/write attributes	Reset value	Values	Meaning
15:14	RW	00	00	Breakpoint matches any virtual address
			01	Breakpoint matches a Non-secure virtual address
			10	Breakpoint matches a Secure virtual address
			11	RESERVED

4.7 CP14 registers 112-127: Watchpoint Control Registers

Security Extensions extends the *Watchpoint Control Registers* (WCRs) with two bits. Table 4-7 shows these bits. For more details about existing bits in the WCRs, see the *ARM Architecture Reference Manual*.

Bits WCR[15:14] control whether watchpoints are conditional on the processor state (Secure or Non-secure).

Note

The WCRs can only be accessed in Privileged modes.

Table 4-7 WCR

Bits	Read/write attributes	Reset value	Values	Meaning
15:14	RW	00	00	Watchpoint matches any virtual address
			01	Watchpoint matches a Non-secure virtual address
			10	Watchpoint matches a Secure virtual address
			11	RESERVED

4.8 External debug interface

Security Extensions adds three signals to the existing debug interface:

NIDEN Non-Invasive Debug Enable

SPIDEN Secure Privileged Invasive Debug Enable

SPNIDEN Secure Privileged Non-Invasive Debug Enable.

Note

To prevent these bits being controlled from JTAG, they are not included in the boundary scan chain.

Note

For legacy systems, these input inputs must be tied high to enable debug.

4.8.1 NIDEN

The **NIDEN** input enables or disables non-invasive debug:

- If **NIDEN** is high, non-invasive debug is enabled in all Non-secure modes, and in any Secure modes permitted by **SPNIDEN** and SUNIDEN.
- If **NIDEN** is low, non-invasive debug is not enabled in any mode. The state of **SPNIDEN** and SUNIDEN have no effect.

See *Secure Debug Enable register* on page 3-15 for details of the SUNIDEN bit.

4.8.2 SPIDEN

The **SPIDEN** input enables or disables invasive debug in the Secure world:

- If **SPIDEN** is high, invasive debug is permitted in all Secure modes.

  Note

  In this case, invasive debug is permitted in Secure User mode, regardless of the value of the SUIDEN bit.

- If **SPIDEN** is low, invasive debug is not permitted in Secure Privileged modes. Invasive debug is permitted in Secure User mode according to the value of the SUIDEN bit.

See *Secure Debug Enable register* on page 3-15 for details of the SUIDEN bit.

4.8.3 SPNIDEN

The **SPNIDEN** input enables or disables non-invasive debug in the Secure world:

- If **SPNIDEN** is high, non-invasive debug is permitted in all Secure modes.

  Note

  In this case, non-invasive debug is permitted in Secure User mode, regardless of the value of the SUNIDEN bit.

- If **SPNIDEN** is low, non-invasive debug is not permitted in Secure Privileged modes. Non-invasive debug is permitted in Secure User mode according to the value of the SUNIDEN bit.

See *Secure Debug Enable register* on page 3-15 for details of the SUNIDEN bit.

4.8.4 DBGEN, SPIDEN and SUIDEN

Table 4-8 shows the effects of the **DBGEN** and **SPIDEN** inputs, and the SUIDEN bit.

Table 4-8 DBGEN, SPIDEN and SUIDEN

DBGEN	DSCR[15:14]	SPIDEN	SUIDEN	Debug mode	Invasive debug permitted in modes
0	XX [a]	X	X	Disabled	None
1	00	0	0	Not selected [b]	Non-secure modes
1	00	0	1	Not selected [b]	Non-secure modes, Secure User
1	00	1	X	Not selected [b]	Secure and Non-secure
1	10	1	X	Monitor debug-mode	Secure and Non-secure
1	10	0	0	Monitor debug-mode	Non-secure modes
1	10	0	1	Monitor debug-mode	Non-secure, Secure User
1	X1	1	X	Halting debug-mode	Secure and Non-secure
1	X1	0	0	Halting debug-mode	Non-secure modes
1	X1	0	1	Halting debug-mode	Non-secure, Secure User

a. DCSR[15:14] reads as 0b00.
b. When **DBGEN**==1 and DSCR[15:14]==0b00, some debug events are permitted. See Table 4-10 on page 4-18 for details.

4.8.5 NIDEN, SPNIDEN and SUNIDEN

Table 4-9 shows the effects of the **NIDEN** and **SPNIDEN** inputs, and the SUNIDEN bit.

Table 4-9 NIDEN, SPNIDEN and SUNIDEN

NIDEN	SPNIDEN	SUNIDEN	Debug	Non-invasive debug permitted modes
0	X	X	Disabled	-
1	1	X	Enabled	All Secure and Non-secure modes
1	0	0	Enabled	All Non-secure modes only
1	0	1	Enabled	All Non-secure, Secure User only

4.8.6 Changing the Debug Enable signals

The behavior of the **NIDEN**, **DBGEN**, **SPIDEN**, and **SPNIDEN** control signals is mainly the responsibility of the external debug interface. However, these signals can be changed while the processor is running, or while the processor is in Debug state.

Software running on the processor can change the state of these signals as follows:

1. Execute an implementation specific sequence of instructions to change the signal value. For example, this might be an instruction to write a value to a control register in a system peripheral.
2. If step 1 involves any memory operation, issue a DSB.
3. Poll the debug registers for the processor view of the signal values. This is required because the system might not issue the signal change to the processor until several cycles after the DSB completes.
4. Issue a PrefetchFlush.

The software cannot perform debug or analysis operations that rely on the new value until this procedure has been completed. The same rules apply for instructions executed through the ITR while in Debug state.

The processor view of the **SPIDEN** and **SPNIDEN** signals can be polled through the DSCR.

The processor has no direct view of the **DBGEN** signal. However, if **DBGEN** is low, DSCR[15:14] reads as 0b00. Software can determine **DBGEN** by reading DSCR[15:14] after writing a non-zero value to it.

4.9 Debug event

If any debug event occurs while the core is in a mode where invasive debug is not permitted, it is ignored. This includes all the following debug events:

- breakpoint
- watchpoint
- vector catch
- external debug request signal activation
- debug state entry request command.

Note

If a BKPT instruction is executed while invasive debug is not permitted, the same sequence of actions is taken as if Monitor debug-mode were selected. This is not affected by the actual configuration of Debug mode (Halting or Monitor).

If invasive debug is enabled and permitted by **DBGEN**, **SPIDEN**, and SUIDEN, the following debug events cause an entry to Debug state:

- external debug request signal activation
- debug state entry request command.

This occurs regardless of the configuration of debug mode.

When a debug event occurs and is not ignored, the following registers are updated:

- IFSR
- DFSR
- FAR
- WFAR.

See *CP14 register 6, Watchpoint Fault Address Register* on page 4-8 for details of the WFAR.

4.9.1 Action on EBDGRQ and Debug state Entry Request command

The processor enters Debug state if either of the following occurs:

- the external debug request signal (**EDBGRQ**) is activated, and debug is not disabled at the External Debug interface
- a Debug state Entry Request (HALT) command is received, and debug is not disabled at the External Debug interface.

These events are IGNORED if debug is disabled at the External Debug interface.

Table 4-10 shows the behavior of the processor on debug events.

Table 4-10 Processor behavior on debug events

Debug mode	Action on BKPT	Action on EDBGRQ	Action on HALT	Action on other debug events
Debug disabled	Exception	Ignored	Ignored	Ignored
Not permitted	Exception	Ignored	Pended [a]	Ignored
Not selected	Exception	Debug state entry	Debug state entry	Ignored
Monitor debug-mode	Exception	Debug state entry	Debug state entry	Exception
Halting debug-mode	Debug state entry	Debug state entry	Debug state entry	Debug state entry

a. The cores latches the debug entry request.
If the next debug mode change is to Not Selected, Monitor, or Halting debug mode, the core enters Debug state.
If the next debug mode change is to Debug disabled, the request is dropped.

4.10 Debug state

Security Extensions requires some changes to behavior in Debug state. These are described in the following subsections.

4.10.1 Altering mode bits in Debug state

In Debug state, any instruction executed that updates the CPSR to a mode where invasive debug is not permitted have that update ignored. Similarly, if privileged debug is not permitted in the current security world, all privileged bits of the CPSR are read-only.

Table 4-11 shows which updates are permitted in Debug state.

Table 4-11 Permitted updates to the CPSR in Debug state

Mode	NS-state	SPIDEN	Update privileged CPSR bits [a]	Modify M[4:0] to Monitor mode
User	0	0	Update ignored	Update ignored
Privileged	0	0	Permitted	Permitted
Any	1	0	Permitted	Update ignored
Any	X	1	Permitted	Permitted

a. Other than changing M[4:0] to Monitor mode.

4.10.2 Altering the NS-bit

In Debug state, the NS-bit can be altered only:

- in Secure User mode if **SPIDEN** is asserted
- in a Secure Privileged mode, regardless of the state of **SPIDEN**.

This is the case even if invasive debug is permitted everywhere.

4.10.3 Privilege

The processor ignores any attempt to execute privileged instructions, other than CP14 debug instructions, if all the following conditions are true:

- the processor is in Debug state
- the processor is in Secure User mode
- Debug is permitted in Non-secure state and in Secure User mode (**SPIDEN** = 0).

When the processor ignores an instruction in these circumstances, it sets DSCR[8], the sticky undefined bit.

4.10.4 Coprocessor instructions

Instructions that access CP14 are always permitted in Debug state, regardless of debug permissions and the processor mode and security state.

For accesses to CP15 registers, the processor behaves as follows:

- If the debugger is permitted to write to the M[4:0] bits of the CPSR to change to a Privileged mode, then the debugger is permitted to access CP15 registers. There is no requirement to change to a Privileged mode first.
- Access to CP15 registers is limited to the same access granted to any Privileged mode.
- Any attempt to perform accesses that are not permitted is treated as an Undefined exception.
- Accesses to CP15 registers access the CP15 registers of the current World (Secure or Non-secure). If the debugger requires access to Secure CP15 registers, it must change to Secure state.

This means, for example, that:

- If debug is permitted everywhere, and the processor is stopped in any Secure mode (including Secure User mode), then the processor has access to the Secure banked CP15 registers.
- If the processor is stopped in any Non-secure mode (including Non-secure User mode), then the processor is restricted to the following only:
  - Any access to the Non-secure banked CP15 registers.
  - Access to Non-banked CP15 registers accessible in Non-secure state. Some bits are read only, in the same way as usual in Non-secure state.
- If debug is permitted only in Non-secure state, the following conditions apply:
  - the debugger cannot access Secure CP15 registers
  - the debugger cannot write those bits in non-banked registers that are read-only in Non-secure state.

4.10.5 Instructions that modify the CPSR

With the exception of instructions that modify the CPSR, the processor can execute any ARM state instruction in Debug state.

The only instruction that can modify the CPSR in Debug state is the MSR instruction. All other instructions that normally update the CPSR are UNPREDICTABLE in Debug state. This affects the following instructions, for example:

- BX
- BXJ
- CPS
- LDM (3)
- RFE
- SETEND
- data processing instructions that transfer the SPSR to the CPSR.

When not in Debug state, an MSR instruction that modifies the Execution State bits in the CPSR is UNPREDICTABLE. However, in Debug state an MSR instruction does update the Execution State bits in the CPSR. A direct modification of the Execution State bits in the CPSR by an MSR instruction must be followed by an instruction memory barrier sequence.

If an MRS instruction reads the CPSR after an MSR writes the Execution State bits, and before any instruction memory barrier sequence, the result is UNPREDICTABLE.

If the processor leaves Debug state after an MSR writes the Execution State bits, and before any instruction memory barrier sequence, the behavior of the processor is UNPREDICTABLE.

4.10.6 Exceptions

Exceptions are handled as follows when the processor is in Debug state:

Reset No change. The processor leaves Debug state.

Prefetch abort No change. This exception cannot occur because no instructions are prefetched in Debug state.

SWI SWI exceptions are ignored in Debug state.

SMI SMI exceptions are ignored in Debug state.

BKPT Breakpoints are ignored in Debug state.

Undefined When an Undefined Instruction exception occurs in Debug state, the core behaves as follows:

- PC, CPSR, SPSR_und, R14_und, and DSCR[5:2] are unaltered
- the processor remains in Debug state
- DSCR[8] (sticky undefined bit) is set. See *Sticky Undefined bit* on page 4-6 for details.

Precise data abort When a Precise Data Abort occurs in Debug state, the core behaves as follows:

- PC, CPSR, SPSR_abt, R14_abt, and DSCR[5:2] are unaltered
- the processor remains in Debug state
- DSCR[6] (sticky precise data abort bit) is set
- DFSR and FAR are set. It is IMPLEMENTATION DEFINED whether DFSR and FAR are updated when the processor is in Secure User mode, and debug is not permitted in Secure Privileged modes.

Imprecise data abort

When an imprecise Data Abort occurs in Debug state, the core behaves as follows:

- the setting of the CPSR A bit is ignored
- PC, CPSR, SPSR_abt, R14_abt, and DSCR[5:2] are unaltered
- the processor remains in Debug state
- DSCR[7] (sticky imprecise data abort bit) is set
- If DSCR[19] is set, the imprecise data abort is not taken and DFSR is not set.

4.10.7 Imprecise data aborts in detail

On entry to Debug state:

1. DSCR[19] is normally 0.
2. The debugger must issue a DSB operation to flush all pending memory operations to the system.
3. If any of these operations cause an imprecise data abort, the processor latches the abort and its type until the processor leaves Debug state. The aborts are not taken immediately.
4. The processor sets DSCR[19].

After the processor sets DSCR[19], any memory accesses from Debug state that cause imprecise data aborts set DSCR[7] (sticky imprecise data abort), but are otherwise discarded. The cause and type of the abort are not recorded. This means that any abort that is still latched from the initial DSB that was completed on entry to Debug state is not overwritten by any new abort.

After writes to memory by the debugger, it must issue a DSB operation to ensure that any imprecise data aborts have been detected.

Before exit from Debug state, a debugger must issue a DSB operation.

On exit from Debug state, DSCR[19] is cleared by the processor.

If an imprecise data abort occurred between entry to Debug state and the processor setting DSCR[19]:

- If the A-bit is 1, the abort is pended until the A-bit is cleared.
- If the A-bit is 0, the abort is taken by the processor on exit from Debug state.

4.10.8 Imprecise data aborts and watchpoints

The Watchpoint exception has a higher priority than an Imprecise Data Abort. If a data access causes both a watchpoint and an Imprecise Data Abort, the processor enters Debug state before taking the Imprecise Data Abort. The abort is latched because the processor is in Debug state.

This ensures correct behavior where debug is not permitted in Privileged modes.

4.11 Non-invasive Debug

Security Extensions requires some changes to non-invasive Debug. These are described in the following subsections.

4.11.1 Performance Monitoring Unit

When the core is in Debug state, or in a mode where non-invasive debug is not permitted:

- events are not counted by the Performance Monitoring Unit
- events are not visible to the Trace device
- the cycle count register, CCNT, continues to count.

4.11.2 PC sample register

When the core is in Debug state, or in a mode where non-invasive debug is not permitted, the PC sample register always reads `0xFFFFFFFF`.

4.11.3 Trace

When the core is in Debug state, or in a mode where non-invasive debug is not permitted, all instructions and data transfers are ignored by the Trace device.

Glossary

Interrupt Status Register (ISR)
The ISR shows whether there is pending IRQ, FIQ or External Abort.

Monitor mode
An ARM mode that is responsible for switching the core between the Secure and Non-secure state. Do not confuse Monitor mode with *Monitor debug-mode*.

Monitor Vector Base Address Register (MVBAR)
The MVBAR exists only in the Secure world. When an exception branches to Monitor mode, the core branches to:

`Monitor_Base_Address + Exception_Vector_Address`.

Non-Secure Access Control register (NSAC)
The NSAC defines the Non-secure access permissions to coprocessors, cache lockdown registers, TLB lockdown registers, and internal DMA.

Non-secure interrupt
An interrupt generated by a non-secure peripheral.

NS-bit Bit[0] in the SCR is the Non-Secure bit. Controls whether the processor is in Secure (0) or Non-secure (1) state, except that in Monitor mode the processor is in Secure state regardless of the value of the NS-bit.

NS-state
The processor is in Non-secure state if the NS-bit is 1, and the processor is in any mode *other than* Monitor mode. The core cannot access the Secure world.

Non-Secure Table ID (NSTID)
TLB entries are marked with an ID, the NSTID. The NSTID determines whether the entry corresponds to a Secure or a Non-secure entry.

Non-secure world
All the hardware, both core and system, that is accessible when the core is in Non-secure state.

PA Physical Address.

PA Register (PAR)
The PAR of the current world receives the PA during any VA to PA translation.

Secure Configuration Register (SCR)
The SCR is a CP15 register. It defines the configuration of the current state. It specifies the state of the core (Secure or Non-secure), and what mode the core branches to if an IRQ, FIQ or external abort occurs. It also defines whether or not the I and A bits in the CPSR can be modified in Non-secure world.

It is accessible in Secure Privileged modes only.

Secure Debug Enable Register (SDE)
The SDE enables or disables both Invasive and Non-invasive debug. It is accessible in Secure Privileged modes only.

Secure interrupt
An interrupt generated by a secure peripheral.

Secure state
The processor is in Secure state if the NS-bit is 0, *or* the processor is in Monitor mode. The core can access both the Secure and Non-secure worlds.

Secure world
All the hardware, both core and system, that is only accessible when the core is in Secure state.

SMI Software Monitor Interrupt instruction. If executed in a Privileged mode, this instruction causes a Software Monitor exception and enters Monitor mode. Otherwise, it causes an Undefined Instruction exception.

Software Monitor exception
A dedicated exception for the SMI instruction.

Thread ID
An identifier provided by a Non-secure OS to distinguish multiple threads of execution in the Secure world.

VA Virtual Address.

Vector Base Address Register (VBAR)
The VBAR is banked in the Secure and Non-secure worlds.When an exception branches to a Secure Privileged mode, the core branches to:

```
Secure_Vector_Base_Address + Exception_Vector_Address
```

When an exception branches to a Non-secure Privileged mode, the core branches to:

```
Non_Secure_Vector_Base_Address + Exception_Vector_Address
```

