## 2.12 Exceptions

Exceptions occur whenever the normal flow of a program has to be halted temporarily. For example, to service an interrupt from a peripheral. Before attempting to handle an exception, the processor preserves the current processor state so that the original program can resume when the handler routine has finished.

If two or more exceptions occur simultaneously, the exceptions are dealt with in the fixed order given in *Exception priorities* on page 2-57.

This section provides details of the processor exception handling:

- *Exception entry and exit summary* on page 2-37
- *Entering an ARM exception* on page 2-38
- *Leaving an ARM exception* on page 2-38.

Several enhancements are made in ARM architecture v6 to the exception model, mostly to improve interrupt latency, as follows:

- New instructions are added to give a choice of stack to use for storing the exception return state after exception entry, and to simplify changes of processor mode and the disabling and enabling of interrupts.
- The interrupt vector definitions on ARMv6 are changed to support the addition of hardware to prioritize the interrupt sources and to look up the start vector for the related interrupt handling routine.
- A low interrupt latency configuration is added in ARMv6. In terms of the instruction set architecture, it specifies that multi-access load/store instructions, ARM LDC, LDM, LDRD, STC, STM, and STRD, and Thumb LDMIA, POP, PUSH, and STMIA, can be interrupted and then restarted after the interrupt has been processed.
- Support for an imprecise Data Abort that behaves as an interrupt rather than as an abort, in that it occurs asynchronously relative to the instruction execution. Support involves the masking of a pending imprecise Data Abort at times when entry into Abort mode is deemed unrecoverable.

### 2.12.1 New instructions for exception handling

This section describes the instructions added to accelerate the handling of exceptions. Full details of these instructions are given in the *ARM Architecture Reference Manual*.

#### Store Return State (SRS)

This instruction stores R14_<current_mode> and SPSR_<current_mode> to sequential addresses, using the banked version of R13 for a specified mode to supply the base address, and to be written back to if base register Write-Back is specified. This enables an exception handler to store its return state on a stack other than the one automatically selected by its exception entry sequence.

The addressing mode used is a version of an ARM addressing mode, modified to assume a {R14,SPSR} register list rather than using a list specified by a bit mask in the instruction. For more information see the *ARM Architecture Reference Manual*. This enables the SRS instruction to access stacks in a manner compatible with the normal use of STM instructions for stack accesses.

When in Non-secure state, specifying Secure Monitor mode in <mode> parameter field causes the SRS to be an Undefined exception. The behavior prevents the Secure Monitor stack values being altered.

### Return From Exception (RFE)

This instruction loads the PC and CPSR from sequential addresses. This is used to return from an exception that has had its return state saved using the SRS instruction, see *Store Return State (SRS)* on page 2-36, and again uses a version of an ARM addressing mode, modified to assume a {PC,CPSR} register list.

### Change Processor State (CPS)

This instruction provides new values for the CPSR interrupt masks, mode bits, or both, and is designed to shorten and speed up the read/modify/write instruction sequence used in ARMv5 to perform such tasks. Together with the SRS instruction, it enables an exception handler to save its return information on the stack of another mode and then switch to that other mode, without modifying the stack belonging to the original mode or any registers other than the new mode stack pointer.

This instruction also streamlines interrupt mask handling and mode switches in other code. In particular it enables short code sequences to be made atomic efficiently in a uniprocessor system by disabling interrupts at their start and re-enabling interrupts at their end. A similar Thumb instruction is also provided. However, the Thumb instruction can only change the interrupt masks, not the processor mode as well, to avoid using too much instruction set space.

## 2.12.2 Exception entry and exit summary

Table 2-8 summarizes the PC value preserved in the relevant R14 on exception entry, and the recommended instruction for exiting the exception handler. Full details of Jazelle state exceptions are provided in the *Jazelle V1 Architecture Reference Manual*.

**Table 2-8 Exception entry and exit**

| Exception or entry | Return instruction | Previous state | | | Notes |
|---|---|---|---|---|---|
| | | ARM R14_x | Thumb R14_x | Jazelle R14_x | |
| SVC | `MOVS PC, R14_svc` | PC + 4 | PC+2 | - | Where the PC is the address of the SVC, SMC, or undefined instruction. Not used in Jazelle state. |
| SMC | `MOVS PC, R14_mon` | PC + 4 | - | - | |
| UNDEF | `MOVS PC, R14_und` | PC + 4 | PC+2 | - | |
| PABT | `SUBS PC, R14_abt, #4` | PC + 4 | PC+4 | PC+4 | Where the PC is the address of instruction that had the Prefetch Abort. |
| FIQ | `SUBS PC, R14_fiq, #4` | PC + 4 | PC+4 | PC+4 | Where the PC is the address of the instruction that was not executed because the FIQ or IRQ took priority. |
| IRQ | `SUBS PC, R14_irq, #4` | PC + 4 | PC+4 | PC+4 | |
| DABT | `SUBS PC, R14_abt, #8` | PC + 8 | PC+8 | PC+8 | Where the PC is the address of the Load or Store instruction that generated the Data Abort. |
| RESET | `NA` | - | - | - | The value saved in R14_svc on reset is Unpredictable. |
| BKPT | `SUBS PC, R14_abt, #4` | PC + 4 | PC+4 | PC+4 | Software breakpoint. |

### 2.12.3 Entering an ARM exception

SCR[3:1] determine the mode that the processor enters on an FIQ, IRQ, or external abort exception, see *System control and configuration* on page 3-5.

When handling an ARM exception the processor:

1. Preserves the address of the next instruction in the appropriate LR. When the exception entry is from:

   **ARM and Jazelle states:**

   The processor writes the value of the PC into the LR, offset by a value, current PC + 4 or PC + 8 depending on the exception, that causes the program to resume from the correct place on return.

   **Thumb state:**

   The processor writes the value of the PC into the LR, offset by a value, current PC + 2, PC + 4 or PC + 8 depending on the exception, that causes the program to resume from the correct place on return.

   The exception handler does not have to determine the state when entering an exception. For example, in the case of a SVC, `MOVS PC, R14_svc` always returns to the next instruction regardless of whether the SVC was executed in ARM or Thumb state.

2. Copies the CPSR into the appropriate SPSR.

3. Forces the CPSR mode bits to a value that depends on the exception.

4. Forces the PC to fetch the next instruction from the relevant exception vector.

The processor can also set the interrupt and imprecise abort disable flags to prevent otherwise unmanageable nesting of exceptions.

**Note**

Exceptions are always entered, handled, and exited in ARM state. When the processor is in Thumb state or Jazelle state and an exception occurs, the switch to ARM state takes place automatically when the exception vector address is loaded into the PC.

### 2.12.4 Leaving an ARM exception

When an exception has completed, the exception handler must move the LR, minus an offset to the PC. The offset varies according to the type of exception, as Table 2-8 on page 2-37 lists.

Typically the return instruction is an arithmetic or logical operation with the S bit set and rd = R15, so the core copies the SPSR back to the CPSR.

**Note**

The action of restoring the CPSR from the SPSR automatically resets the T bit and J bit to the values held immediately prior to the exception. The A, I, and F bits are also automatically restored to the value they held immediately prior to the exception.

### 2.12.5 Reset

When the **nRESETIN** and **nVFPRESETIN** signals are driven LOW a reset occurs, and the processor abandons the executing instruction.

When **nRESETIN** and **nVFPRESETIN** are driven HIGH again the processor:

1. Forces NS bit in SCR to 0, Secure, CPSR M[4:0] to b10011, Secure Supervisor mode, sets the A, I, and F bits in the CPSR, and clears the CPSR T bit and J bit. The E bit is set based on the state of the **BIGENDINIT** and **UBITINIT** pins. Other bits in the CPSR are indeterminate.
2. Forces the PC to fetch the next instruction from the reset vector address.
3. Reverts to ARM state, and resumes execution.

After reset, all register values except the PC and CPSR are indeterminate.

See Chapter 9 *Clocking and Resets* for more details of the reset behavior for the processor.

### 2.12.6 Fast interrupt request

The *Fast Interrupt Request* (FIQ) exception supports fast interrupts. In ARM state, FIQ mode has eight private registers to reduce, or even remove the requirement for register saving, minimizing the overhead of context switching.

An FIQ is externally generated by taking the **nFIQ** signal input LOW. The **nFIQ** input is registered internally to the processor. It is the output of this register that is used by the processor control logic.

Irrespective of whether exception entry is from ARM state, Thumb state, or Jazelle state, an FIQ handler returns from the interrupt by executing:

```
SUBS PC,R14_fiq,#4
```

You can disable FIQ exceptions within a privileged mode by setting the CPSR F flag. When the F flag is clear, the processor checks for a LOW level on the output of the nFIQ register at the end of each instruction.

The FW bit and FIQ bit in the SCR register configure the FIQ as:

- non maskable in Non-secure world, FW bit in SCR
- branch to either current FIQ mode or Secure Monitor mode, FIQ bit in SCR.

FIQs and IRQs are disabled when an FIQ occurs. You can use nested interrupts but it is up to you to save any corruptible registers and to re-enable FIQs and interrupts.

### 2.12.7 Interrupt request

The IRQ exception is a normal interrupt caused by a LOW level on the **nIRQ** input. IRQ has a lower priority than FIQ, and is masked on entry to an FIQ sequence.

Irrespective of whether exception entry is from ARM state, Thumb state, or Jazelle state, an IRQ handler returns from the interrupt by executing:

```
SUBS PC,R14_irq,#4
```

You can disable IRQ exceptions within a privileged mode by setting the CPSR I flag. When the I flag is clear, the processor checks for a LOW level on the output of the nIRQ register at the end of each instruction.

IRQs are disabled when an IRQ occurs. You can use nested interrupts but it is up to you to save any corruptible registers and to re-enable IRQs.

The IRQ bit in the SCR register configures the IRQ to branch to either the current IRQ mode or to the Secure Monitor mode.

### 2.12.8 Low interrupt latency configuration

The FI bit, bit 21, in CP15 register 1 enables a low interrupt latency configuration. This bit is not duplicated in both worlds, and can only be modified in Secure state. It applies to both worlds.

This mode reduces the interrupt latency of the processor. This is achieved by:

- disabling *Hit-Under-Miss* (HUM) functionality
- abandoning restartable external accesses so that the core can react to a pending interrupt faster than is normally the case
- recognizing low-latency interrupts as early as possible in the main pipeline.

To ensure that a change between normal and low interrupt latency configurations is synchronized correctly, the FI bit must only be changed in using the sequence:

1. Data Synchronization Barrier.
2. Change FI Bit.
3. Data Synchronization Barrier with interrupt disabled.

You must disable interrupts during this complete sequence of operations.

You must ensure that software systems only change the FI bit shortly after Reset, while interrupts are disabled. In low interrupt latency configuration, software must only use multi-word load/store instructions in ways that are fully restartable. In particular, they must not be used on memory locations that produce non-idempotent side-effects for the type of memory access concerned.

This enables, but does not require, implementations to make these instructions interruptible when in low interrupt latency configuration. If the instruction is interrupted before it is complete, the result might be that one or more of the words are accessed twice, but the idempotency of the side-effects, if any, of the memory accesses ensures that this does not matter.

**Note**

There is a similar existing requirement with unaligned and multi-word load/store instructions that access memory locations that can abort in a recoverable way. An abort on one of the words accessed can cause a previously-accessed word to be accessed twice, once before the abort, and once again after the abort handler has returned. The requirement in this case is either:

- all side-effects are idempotent
- the abort must either occur on the first word accessed or not at all.

The instructions that this rule currently applies to are:

- ARM instructions LDC, all forms of LDM, LDRD, STC, all forms of STM, STRD, and unaligned LDR, STR, LDRH, and STRH
- Thumb instructions LDMIA, PUSH, POP, and STMIA, and unaligned LDR, STR, LDRH, and STRH.

System designers are also advised that memory locations accessed with these instructions must not have large numbers of wait-states associated with them if the best possible interrupt latency is to be achieved.

### 2.12.9 Interrupt latency example

This section gives an extended example to show how the combination of new facilities improves interrupt latency. The example is not necessarily entirely realistic, but illustrates the main points. To be simpler, this example applies for legacy code, that is for code that does not use any TrustZone features. You can therefore assume the core only runs code in either Secure or Non-secure world.

The assumptions made are:

1. *Vector Interrupt Controller* (VIC) hardware exists to prioritize interrupts and to supply the address of the highest priority interrupt to the processor core on demand. In the ARMv5 system, the address is supplied in a memory-mapped I/O location, and loading the address acts as an entering interrupt handler acknowledgement to the VIC. In the ARMv6 system, the address is loaded and the acknowledgement given automatically, as part of the interrupt entry sequence. In both systems, a store to a memory-mapped I/O location is used to send a finishing interrupt handler acknowledgement to the VIC.

2. The system has the following layers:

   **Real-time layer** Contains handlers for a number of high-priority interrupts. These interrupts can be prioritized, and are assumed to be signaled to the processor core by means of the FIQ interrupt. Their handlers do not use the facilities supplied by the other two layers. This means that all memory they use must be locked down in the TLBs and caches. It is possible to use additional code to make access to nonlocked memory possible, but this example does not describe this.

   **Architectural completion layer**

   Contains Prefetch Abort, Data Abort and Undefined instruction handlers whose purpose is to give the illusion that the hardware is handling all memory requests and instructions on its own, without requiring software to handle TLB misses, virtual memory misses, and near-exceptional floating-point operations, for example. This illusion is not available to the real-time layer, because the software handlers concerned take a significant number of cycles, and it is not reasonable to have every memory access to take large numbers of cycles. Instead, the memory concerned has to be locked down.

   **Non real-time layer**

   Provides interrupt handlers for low-priority interrupts. These interrupts can also be prioritized, and are assumed to be signaled to the processor core using the IRQ interrupt.

3. The corresponding exception priority structure is as follows, from highest to lowest priority:
   a. FIQ1, highest priority FIQ
   b. FIQ2
   c. ...
   d. FIQm, lowest priority FIQ
   e. Data Abort
   f. Prefetch Abort
   g. Undefined instruction
   h. SVC
   i. IRQ1, highest priority IRQ
   j. IRQ2
   k. ...

l. IRQn, lowest priority IRQ

The processor core prioritization handles most of the priority structure, but the VIC handles the priorities within each group of interrupts.

**Note**

This list reflects the priorities that the handlers are subject to, and differs from the priorities that the exception entry sequences are subject to. The difference occurs because simultaneous Data Abort and FIQ exceptions result in the sequence:

a. Data Abort entry sequence executed, updating R14_abt, SPSR_abt, PC, and CPSR.

b. FIQ entry sequence executed, updating R14_fiq, SPSR_fiq, PC, and CPSR.

c. FIQ handler executes to completion and returns.

d. Data Abort handler executes to completion and returns.

For more information see the *ARM Architecture Reference Manual*.

4. Stack and register usage is:
   - The FIQ1 interrupt handler has exclusive use of R8_fiq to R12_fiq. In ARMv5, R13_fiq points to a memory area, that is mainly for use by the FIQ1 handler. However, a few words are used during entry for other FIQ handlers. In ARMv6, the FIQ1 interrupt handler has exclusive use of R13_fiq.
   - The Undefined instruction, Prefetch Abort, Data Abort, and non-FIQ1 FIQ handlers use the stack pointed to by R13_abt. This stack is locked down in memory, and therefore of known, limited depth.
   - All IRQ and SVC handlers use the stack pointed to by R13_svc. This stack does not have to be locked down in memory.
   - The stack pointed to by R13_usr is used by the current process. This process can be privileged or unprivileged, and uses System or User mode accordingly.
5. Timings are roughly consistent with ARM10 timings, with the pipeline reload penalty being three cycles. It is assumed that pipeline reloads are combined to execute as quickly as reasonably possible, and in particular that:
   - If an interrupt is detected during an instruction that has set a new value for the PC, after that value has been determined and written to the PC but before the resulting pipeline refill is completed, the pipeline refill is abandoned and the interrupt entry sequence started as soon as possible.
   - Similarly, if an FIQ is detected during an exception entry sequence that does not disable FIQs, after the updates to R14, the SPSR, the CPSR, and the PC but before the pipeline refill has completed, the pipeline refill is abandoned and the FIQ entry sequence started as soon as possible.

### FIQs in the example system in ARMv5

In ARMv5, all FIQ interrupts come through the same vector, at address `0x0000001C` or `0xFFFF001C`. To implement the above system, the code at this vector must get the address of the correct handler from the VIC, branch to it, and transfer to using R13_abt and the Abort mode stack if it is not the FIQ1 handler. The following code does, assuming that R8_fiq holds the address of the VIC:

```
FIQhandler
    LDR     PC, [R8,#HandlerAddress]
...
FIQ1handler
... Include code to process the interrupt ...
```

```
    STR     R0, [R8,#AckFinished]
    SUBS    PC, R14, #4
...

FIQ2handler
    STMIA   R13, {R0-R3}
    MOV     R0, LR
    MRS     R1, SPSR
    ADD     R2, R13, #8
    MRS     R3, CPSR
    BIC     R3, R3, #0x1F
    ORR     R3, R3, #0x1B   ; = Abort mode number
    MSR     CPSR_c, R3
    STMFD   R13!, {R0, R1}
    LDMIA   R2, {R0, R1}
    STMFD   R13!, {R0, R1}
    LDMDB   R2, {R0, R1}
    BIC     R3, R3, #0x40   ; = F bit
    MSR     CPSR_c, R3
... FIQs are now re-enabled, with original R2, R3, R14, SPSR on stack
... Include code to stack any more registers required, process the interrupt
... and unstack extra registers
    ADR     R2, #VICaddress
    MRS     R3, CPSR
    ORR     R3, R3, #0x40   ; = F bit
    MSR     CPSR_c, R3
    STR     R0, [R2,#AckFinished]
    LDR     R14, [R13,#12]  ; Original SPSR value
    MSR     SPSR_fsxc, R14
    LDMFD   R13!, {R2,R3,R14}
    ADD     R13, R13, #4
    SUBS    PC, R14, #4
...
```

The major problem with this is the length of time that FIQs are disabled at the start of the lower priority FIQs. The worst-case interrupt latency for the FIQ1 interrupt occurs if a lower priority FIQ2 has fetched its handler address, and is approximately:

- 3 cycles for the pipeline refill after the LDR PC instruction fetches the handler address
- + 24 cycles to get to and execute the MSR instruction that re-enables FIQs
- + 3 cycles to re-enter the FIQ exception
- + 5 cycles for the LDR PC instruction at FIQhandler
- = 35 cycles.

**Note**

FIQs must be disabled for the final store to acknowledge the end of the handler to the VIC. Otherwise, more badly timed FIQs, each occurring close to the end of the previous handler, can cause unlimited growth of the locked-down stack.

### FIQs in the example system in ARMv6

Using the VIC and the new instructions, there is no longer any requirement for everything to go through the single FIQ vector, and the changeover to a different stack occurs much more smoothly. The code is:

```
FIQ1handler
... Include code to process the interrupt ...
```

```
    STR     R0, [R8,#AckFinished]
    SUBS    PC, R14, #4
...
FIQ2handler
    SUB     R14, R14, #4
    SRSFD   R13_abt!
    CPSIE   f, #0x1B   ; = Abort mode

    STMFD   R13!, {R2, R3}
... FIQs are now re-enabled, with original R2, R3, R14, SPSR on stack
... Include code to stack any more registers required, process the interrupt
... and unstack extra registers
    LDMFD   R13!, {R2, R3}
    ADR     R14, #VICaddress
    CPSID   f
    STR     R0, [R14,#AckFinished]
    RFEFD   R13!
...
```

The worst-case interrupt latency for a FIQ1 now occurs if the FIQ1 occurs during an FIQ2 interrupt entry sequence, after it disables FIQs, and is approximately:

- 3 cycles for the pipeline refill for the FIQ2 exception entry sequence
- + 5 cycles to get to and execute the CPSIE instruction that re-enables FIQs
- + 3 cycles to re-enter the FIQ exception
- = 11 cycles.

**Note**

In the ARMv5 system, the potential additional interrupt latency caused by a long LDM or STM being in progress when the FIQ is detected was only significant because the memory system was able to stretch its cycles considerably. Otherwise, it was dwarfed by the number of cycles lost because of FIQs being disabled at the start of a lower-priority interrupt handler. In ARMv6, this is still the case, but it is a lot closer.

### Alternatives to the example system

Two alternatives to the design in *FIQs in the example system in ARMv6* on page 2-43 are:

- The first alternative is not to reserve the FIQ registers for the FIQ1 interrupt, but instead either to:
  - share them out among the various FIQ handlers

    The first restricts the registers available to the FIQ1 handler and adds the software complication of managing a global allocation of FIQ registers to FIQ handlers. Also, because of the shortage of FIQ registers, it is not likely to be very effective if there are many FIQ handlers.
  - require the FIQ handlers to treat them as normal callee-save registers.

    The second adds a number of cycles of loading important addresses and variable values into the registers to each FIQ handler before it can do any useful work. That is, it increases the effective FIQ latency by a similar number of cycles.

- The second alternative is to use IRQs for all but the highest priority interrupt, so that there is only one level of FIQ interrupt. This achieves very fast FIQ latency, 5-8 cycles, but at a cost to all the lower-priority interrupts that every exception entry sequence now disables them. You then have the following possibilities:
  - — None of the exception handlers in the architectural completion layer re-enable IRQs. In this case, all IRQs suffer from additional possible interrupt latency caused by those handlers, and so effectively are in the non real-time layer. In other words, this results in there only being one priority for interrupts in the real-time layer.
  - — All of the exception handlers in the architectural completion layer re-enable IRQs to permit IRQs to have real-time behavior. The problem in this case is that all IRQs can then occur during the processing of an exception in the architectural completion layer, and so they are all effectively in the real-time layer. In other words, this effectively means that there are no interrupts in the non real-time layer.
  - — All of the exception handlers in the architectural completion layer re-enable IRQs, but they also use additional VIC facilities to place a lower limit on the priority of IRQs that is taken. This permits IRQs at that priority or higher to be treated as being in the real-time layer, and IRQs at lower priorities to be treated as being in the non real-time layer. The price paid is some additional complexity in the software and in the VIC hardware.

**Note**

For either of the last two options, the new instructions speed up the IRQ re-enabling and the stack changes that are likely to be required.

### 2.12.10 Aborts

An abort can be caused by either:

- the MMU signalling an internal abort
- an external abort being raised from the AXI interfaces, by an AXI error response.

There are two types of abort:

- *Prefetch Abort*
- *Data Abort* on page 2-46.

IRQs are disabled when an abort occurs. When the aborts are configured to branch to Secure Monitor mode, the FIQ is also disabled.

**Note**

The Interrupt Status Register shows at any time if there is a pending IRQ, FIQ, or External Abort. For more information, see *c12, Interrupt Status Register* on page 3-123.

All aborts from the TLB are internal except for aborts from page table walks that are external precise aborts. If the EA bit is 1 for translation aborts, see *c1, Secure Configuration Register* on page 3-52, the core branches to Secure Monitor mode in the same way as it does for all other external aborts.

#### Prefetch Abort

This is signaled with the Instruction as it enters the pipeline Decode stage.

When a Prefetch Abort occurs, the processor marks the prefetched instruction as invalid, but does not take the exception until the instruction is to be executed. If the instruction is not executed, for example because a branch occurs while it is in the pipeline, the abort does not take place.

After dealing with the cause of the abort, the handler executes the following instruction irrespective of the processor operating state:

```
SUBS PC,R14_abt,#4
```

This action restores both the PC and the CPSR, and retries the aborted instruction.

### Data Abort

Data Abort on the processor can be precise or imprecise. Precise Data Aborts are those generated after performing an instruction side CP15 operation, and all those generated by the MMU:

- alignment faults
- translation faults
- access bit faults
- domain faults
- permission faults.

Data Aborts that occur because of watchpoints are imprecise in that the processor and system state presented to the abort handler is the processor and system state at the boundary of an instruction shortly after the instruction that caused the watchpoint, but before any following load/store instruction. Because the state that is presented is consistent with an instruction boundary, these aborts are restartable, even though they are imprecise.

Errors that cause externally generated Data Aborts might be precise or imprecise. Two separate FSR encodings indicate if the external abort is precise or imprecise:

- all external aborts to loads when the CP15 Register 1 FI bit, bit 21, is set are precise
- all external aborts to loads or stores to Strongly Ordered memory are precise
- all external aborts to loads to the Program Counter or the CSPR are precise
- all external aborts on the load part of a SWP are precise
- all other external aborts are imprecise.

External aborts are supported on cacheable locations. The abort is transmitted to the processor only if a word requested by the processor had an external abort.

#### *Precise Data Aborts*

A precise Data Abort is signaled when the abort exception enables the processor and system state presented to the abort handler to be consistent with the processor and system state when the aborting instruction was executed. With precise Data Aborts, the restarting of the processor after the cause of the abort has been rectified is straightforward.

The ARM1176JZF-S processor implements the *base restored Data Abort model*, that differs from the *base updated Data Abort model* implemented by the ARM7TDMI-S processor.

With the *base restored Data Abort model*, when a Data Abort exception occurs during the execution of a memory access instruction, the base register is always restored by the processor hardware to the value it contained before the instruction was executed. This removes the requirement for the Data Abort handler to unwind any base register update, that might have been specified by the aborted instruction. This simplifies the software Data Abort handler. See *ARM Architecture Reference Manual* for more details.

After dealing with the cause of the abort, the handler executes the following return instruction irrespective of the processor operating state at the point of entry:

```
SUBS PC,R14_abt,#8
```

This restores both the PC and the CPSR, and retries the aborted instruction.

#### *Imprecise Data Aborts*

An imprecise Data Abort is signaled when the processor and system state presented to the abort handler cannot be guaranteed to be consistent with the processor and system state when the aborting instruction was issued.

### 2.12.11 Imprecise Data Abort mask in the CPSR/SPSR

An imprecise Data Abort caused, for example, by an External Error on a write that has been held in a Write Buffer, is asynchronous to the execution of the causing instruction and can occur many cycles after the instruction that caused the memory access has retired. For this reason, the imprecise Data Abort can occur at a time that the processor is in Abort mode because of a precise Data Abort, or can have live state in Abort mode, but be handling an interrupt.

To avoid the loss of the Abort mode state, R14_abt and SPSR_abt, in these cases, that leads to the processor entering an unrecoverable state, the existence of a pending imprecise Data Abort must be held by the system until a time when the Abort mode can safely be entered.

A mask is added into the CPSR to indicate that an imprecise Data Abort can be accepted. This bit is referred to as the A bit. The imprecise Data Abort causes a Data Abort to be taken when imprecise Data Aborts are not masked. When imprecise Data Aborts are masked, then the implementation is responsible for holding the presence of a pending imprecise Data Abort until the mask is cleared and the abort is taken. The A bit is set automatically on entry into Abort Mode, IRQ, and FIQ Modes, and on Reset.

**Note**

You cannot change the CPSR A bit in the Non-secure world if the SCR bit 5 is reset. You can change the SPSR A bit in the Non-secure world but this does not update the CPSR if the SCR bit 5 does not permit it.

### 2.12.12 Supervisor call instruction

You can use the *Supervisor call* instruction (SVC) to enter Supervisor mode, usually to request a particular supervisor function. The SVC handler reads the opcode to extract the SVC function number. A SVC handler returns by executing the following instruction, irrespective of the processor operating state:

```
MOVS PC, R14_svc
```

This action restores the PC and CPSR, and returns to the instruction following the SVC.

IRQs are disabled when a Supervisor call occurs.

### 2.12.13 Secure Monitor Call (SMC)

When the processor executes the *Secure Monitor Call* (SMC) the core enters Secure Monitor mode to execute the Secure Monitor code. For more details on SMC and the Secure Monitor, see *The NS bit and Secure Monitor mode* on page 2-4.

**Note**

An attempt by a User process to execute an SMC makes the processor enter the Undefined exception trap.

### 2.12.14 Undefined instruction

When an instruction is encountered that neither the processor, nor any coprocessor in the system, can handle the processor takes the undefined instruction trap. Software can use this mechanism to extend the ARM instruction set by emulating undefined coprocessor instructions.

After emulating the failed instruction, the trap handler executes the following instruction, irrespective of the processor operating state:

```
MOVS PC,R14_und
```

This action restores the CPSR and returns to the next instruction after the undefined instruction.

IRQs are disabled when an undefined instruction trap occurs. For more information about undefined instructions, see the *ARM Architecture Reference Manual*.

### 2.12.15 Breakpoint instruction (BKPT)

A breakpoint (BKPT) instruction operates as though the instruction causes a Prefetch Abort.

A breakpoint instruction does not cause the processor to take the Prefetch Abort exception until the instruction reaches the Execute stage of the pipeline. If the instruction is not executed, for example because a branch occurs while it is in the pipeline, the breakpoint does not take place.

After dealing with the breakpoint, the handler executes the following instruction irrespective of the processor operating state:

```
SUBS PC,R14_abt,#4
```

This action restores both the PC and the CPSR, and retries the breakpointed instruction.

**Note**

If the EmbeddedICE-RT logic is configured into Halting debug-mode, a breakpoint instruction causes the processor to enter Debug state. See *Halting debug-mode debugging* on page 13-50.

### 2.12.16 Exception vectors

The Secure Configuration Register bits [3:1] determine the mode that is entered when an IRQ, a FIQ, or an external abort exception occur.

Three CP15 registers define the base address of the following vector tables:

- Non-secure, Non_Secure_Base_Address
- Secure, Secure_Base_Address
- Secure Monitor, Monitor_Base_Address.

If high vectors are enabled, Non_Secure_Base_Address and Secure_Base_Address registers are treated as being `0xFFFF0000`, regardless of the value of these registers.

#### Exceptions occurring in Non-secure world

The following exceptions occur in the Non-secure world:

- *Reset* on page 2-49

- *Undefined instruction*
- *Software Interrupt exception*
- *External Prefetch Abort* on page 2-50
- *Internal Prefetch Abort* on page 2-50
- *External Data Abort* on page 2-50
- *Internal Data Abort* on page 2-51
- *Interrupt request (IRQ) exception* on page 2-51
- *Fast Interrupt Request (FIQ) exception* on page 2-52
- *Secure Monitor Call Exception* on page 2-52.

#### Reset

When Reset is de-asserted:

```
/* Enter secure state */
R14_svc = UNPREDICTABLE value
SPSR_svc = UNPREDICTABLE value
CPSR [4:0] = 0b10011 /* Enter supervisor mode */
CPSR [5] = 0 /* Execute in ARM state */
CPSR [6] = 1 /* Disable fast interrupts */
CPSR [7] = 1 /* Disable interrupts */
CPSR [8] = 1 /* Disable imprecise aborts */
CPSR [9] = Secure EE-bit /* store value of Secure Control Register bit[25] */
CPSR[24] = 0 /* Clear J bit */
if high vectors configured then
    PC = 0xFFFF0000
else
    PC = 0x00000000
```

#### Undefined instruction

On an undefined instruction:

```
/* Non-secure state is unchanged */
R14_und = address of the next instruction after the undefined instruction
SPSR_und = CPSR
CPSR [4:0] = 0b11011 /* Enter undefined Instruction mode */
CPSR [5] = 0 /* Execute in ARM state */
CPSR [7] = 1 /* Disable interrupts */
CPSR [9] = Non-secure EE-bit /* store value of NS Control Reg[25] */
CPSR[24] = 0 /* Clear J bit */
if high vectors configured then
    PC = 0xFFFF0004
else
    PC = Non_Secure_Base_Address + 0x00000004
```

#### Software Interrupt exception

On an SVC:

```
/* Non-secure state is unchanged */
R14_svc = address of the next instruction after the SVC instruction
SPSR_svc = CPSR
CPSR [4:0] = 0b10011 /* Enter supervisor mode */
CPSR [5] = 0 /* Execute in ARM state */
CPSR [7] = 1 /* Disable interrupts */
CPSR [9] = Non-secure EE-bit /* store value of NS Control Reg[25] */
CPSR[24] = 0 /* Clear J bit */
if high vectors configured then
    PC = 0xFFFF0008
else
```

```
    PC = Non_Secure_Base_Address + 0x00000008
```

#### External Prefetch Abort

On an external prefetch abort:

```
if SCR[3]=1 /* external prefetch aborts trapped to Secure Monitor mode */
    R14_mon = address of the aborted instruction + 4
    SPSR_mon = CPSR
    CPSR [4:0] = 0b10110 /* Enter Secure Monitor mode */
    CPSR [5] = 0 /* Execute in ARM state */
    CPSR [6] = 1 /* Disable fast interrupts */
    CPSR [7] = 1 /* Disable interrupts */
    CPSR [8] = 1 /* Disable imprecise aborts */
    CPSR [9] = Secure EE-bit /* store value of Secure Ctrl Reg bit[25] */
    CPSR[24] = 0 /* Clear J bit */
    PC = Monitor_Base_Address + 0x0000000C
Else
    R14_abt = address of the aborted instruction + 4
    SPSR_abt = CPSR
    CPSR [4:0] = 0b10111 /* Enter abort mode */
    CPSR [5] = 0 /* Execute in ARM state */
    CPSR [7] = 1 /* Disable interrupts */
    If SCR[5]=1 (bit AW)
        CPSR [8] = 1 /* Disable imprecise aborts */
    Else
        CPSR [8] = UNCHANGED
    CPSR [9] = Non-secure EE-bit /* store value of NS Control Reg[25] */
    CPSR[24] = 0 /* Clear J bit */
    if high vectors configured then
        PC = 0xFFFF000C
    else
        PC = Non_Secure_Base_Address + 0x0000000C
```

#### Internal Prefetch Abort

On an internal prefetch abort:

```
/* Non-secure state is unchanged */
R14_abt = address of the aborted instruction + 4
SPSR_abt = CPSR
CPSR [4:0] = 0b10111 /* Enter abort mode */
CPSR [5] = 0 /* Execute in ARM state */
CPSR [7] = 1 /* Disable interrupts */
If SCR[5]=1 (bit AW)
    CPSR [8] = 1 /* Disable imprecise aborts */
Else
    CPSR [8] = UNCHANGED
CPSR [9] = Non-secure EE-bit /* store value of NS Control Reg[25] */
CPSR[24] = 0 /* Clear J bit */
if high vectors configured then
    PC = 0xFFFF000C
else
    PC = Non_Secure_Base_Address + 0x0000000C
```

#### External Data Abort

On an External Precise Data Abort or on an External Imprecise Abort with CPSR[8]=0 (A bit):

```
/* Non-secure state is unchanged */
if SCR[3]=1 /* external aborts trapped to Secure Monitor mode */
    R14_mon = address of the aborted instruction + 8
    SPSR_mon = CPSR
    CPSR [4:0] = 0b10110 /* Enter Secure Monitor mode */
```

```
    CPSR [5] = 0 /* Execute in ARM state */
    CPSR [6] = 1 /* Disable fast interrupts */
    CPSR [7] = 1 /* Disable interrupts */
    CPSR [8] = 1 /* Disable imprecise aborts */
    CPSR [9] = Secure EE-bit /* store value of secure Ctrl Reg bit[25] */
    CPSR[24] = 0 /* Clear J bit */
Else /* external Aborts trapped in abort mode */
    R14_abt = address of the aborted instruction + 8
    SPSR_abt = CPSR
    CPSR [4:0] = 0b10111 /* Enter abort mode */
    CPSR [5] = 0 /* Execute in ARM state */
    CPSR [7] = 1 /* Disable interrupts */
    If SCR[5]=1 (bit AW)
        CPSR [8] = 1 /* Disable imprecise aborts */
    Else
        CPSR [8] = UNCHANGED
    CPSR [9] = Non-secure EE-bit /* store value of NS Control Reg[25] */
    CPSR[24] = 0 /* Clear J bit */
    if high vectors configured then
        PC = 0xFFFF0010
    else
        PC = Non_Secure_Base_Address + 0x00000010
```

#### *Internal Data Abort*

On an Internal Data Abort. All aborts that are not external aborts, that is data aborts on L1 memory management occurring when a fault is detected in MMU:

```
/* Non-secure state is unchanged */
R14_abt = address of the aborted instruction + 8
SPSR_abt = CPSR
CPSR [4:0] = 0b10111 /* Enter abort mode */
CPSR [5] = 0 /* Execute in ARM state */
CPSR [7] = 1 /* Disable interrupts */
If SCR[5]=1 (bit AW)
    CPSR [8] = 1 /* Disable imprecise aborts */
Else
    CPSR [8] = UNCHANGED
CPSR [9] = Non-secure EE-bit /* store value of NS Control Reg[25] */
CPSR[24] = 0 /* Clear J bit */
if high vectors configured then
    PC = 0xFFFF0010
else
    PC = Non_Secure_Base_Address + 0x00000010
```

#### *Interrupt request (IRQ) exception*

On an Interrupt Request, and CPSR[7]=0, I bit:

```
/* Non-secure state is unchanged */
if SCR[1]=1 /* IRQ trapped in Secure Monitor mode */
    R14_mon = address of the next instruction to be executed + 4
    SPSR_mon = CPSR
    CPSR [4:0] = 0b10110 /* Enter Secure Monitor mode */
    CPSR [5] = 0 /* Execute in ARM state */
    CPSR [6] = 1 /* Disable fast interrupts */
    CPSR [7] = 1 /* Disable interrupts */
    CPSR [8] = 1 /* Disable imprecise aborts */
    CPSR [9] = Secure EE-bit /* store value of secure Ctrl Reg bit[25] */
    CPSR[24] = 0 /* Clear J bit */
    PC = Monitor_Base_Address + 0x00000018
else
    R14_irq = address of the next instruction to be executed + 4
    SPSR_irq = CPSR
```

```
    CPSR [4:0] = 0b10010 /* Enter IRQ mode */
    CPSR [5] = 0 /* Execute in ARM state */
    CPSR [7] = 1 /* Disable interrupts */
    If SCR[5]=1 (bit AW)
        CPSR [8] = 1 /* Disable imprecise aborts */
    Else
        CPSR [8] = UNCHANGED
    CPSR [9] = Non-secure EE-bit /* store value of NS Control Reg[25] */
    CPSR[24] = 0 /* Clear J bit */
    if VE == 0 /* Core with VIC port only */
        if high vectors configured then
            PC = 0xFFFF0018
        else
            PC = Non_Secure_Base_Address + 0x00000018
    else
        PC = IRQADDR
```

#### *Fast Interrupt Request (FIQ) exception*

On a Fast Interrupt Request, and CPSR[6]=0, F bit:

```
/* Non-secure state is unchanged */
if SCR[2]=1 /* FIQ trapped in Secure Monitor mode */
    R14_mon = address of the next instruction to be executed + 4
    SPSR_mon = CPSR
    CPSR [4:0] = 0b10001 /* Enter Secure Monitor mode */
    CPSR [5] = 0 /* Execute in ARM state */
    CPSR [6] = 1 /* Disable fast interrupts */
    CPSR [7] = 1 /* Disable interrupts */
    CPSR [8] = 1 /* Disable imprecise aborts */
    CPSR [9] = Secure EE-bit /* store value of secure Ctrl Reg bit[25] */
    CPSR[24] = 0 /* Clear J bit */
    PC = Monitor_Base_Address + 0x0000001C
Else
/* SCR[4] (bit FW) must be set to avoid infinite loop until FIQ is asserted */
    R14_fiq = address of the next instruction to be executed + 4
    SPSR_fiq = CPSR
    CPSR [4:0] = 0b10001 /* Enter FIQ mode */
    CPSR [5] = 0 /* Execute in ARM state */
    CPSR [6] = 1 /* Disable fast interrupts */
    CPSR [7] = 1 /* Disable interrupts */
    If SCR[5]=1 (bit AW)
        CPSR [8] = 1 /* Disable imprecise aborts */
    Else
        CPSR [8] = UNCHANGED
    CPSR [9] = Non-secure EE-bit /* store value of NS Control Reg[25] */
    CPSR[24] = 0 /* Clear J bit */
    if high vectors configured then
        PC = 0xFFFF001C
    else
        PC = Non_Secure_Base_Address + 0x0000001C
```

#### *Secure Monitor Call Exception*

On a SMC:

```
If (UserMode) /* undefined instruction */
    R14_und = address of the next instruction after the SMC instruction
    SPSR_und = CPSR
    CPSR [4:0] = 0b11011 /* Enter undefined instruction mode */
    CPSR [5] = 0 /* Execute in ARM state */
    CPSR [7] = 1 /* Disable interrupts */
    CPSR [9] = Non-secure EE-bit /* store value of NS Control Reg[25] */
    CPSR[24] = 0 /* Clear J bit */
```

```
    If high vectors configured then
        PC = 0xFFFF0004
    else
        PC = Non_Secure_Base_Address + 0x00000004
else
    R14_mon = address of the next instruction after the SMC instruction
    SPSR_mon = CPSR
    CPSR [4:0] = 0b10110 /* Enter Secure Monitor mode */
    CPSR [5] = 0 /* Execute in ARM state */
    CPSR [6] = 1 /* Disable fast interrupts */
    CPSR [7] = 1 /* Disable interrupts */
    CPSR [8] = 1 /* Disable imprecise aborts */
    CPSR [9] = Secure EE-bit /* store value of secure Ctrl Reg bit[25] */
    CPSR[24] = 0 /* Clear J bit */
    PC = Monitor_Base_Address + 0x00000008    /* SMC vectored to the */
                                              /*conventional SVC vector */
```

### Exceptions occurring in Secure world

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 and that CPSR[6], F bit, and CPSR[8], A bit, are updated regardless the bits [5:4] of the Secure Configuration Register.

Except Reset, the software model does not expect any other exception to occur in Secure Monitor mode. However, if an exception occurs in Secure Monitor mode, the NS bit in SCR register is automatically reset and the core branches either to the exception handler in Secure world or in Secure Monitor mode, Secure Monitor mode for IRQ, FIQ or external aborts with the corresponding bit set in SCR[3:1].

The following exceptions occur in the Secure world:

- *Reset*
- *Undefined instruction* on page 2-54
- *Software Interrupt exception* on page 2-54
- *External Prefetch Abort* on page 2-54
- *Internal Prefetch Abort* on page 2-55
- *External Data Abort* on page 2-50
- *Internal Data Abort* on page 2-55
- *Interrupt request (IRQ) exception* on page 2-56
- *Fast Interrupt Request (FIQ) exception* on page 2-56
- *Secure Monitor Call Exception* on page 2-57.

#### *Reset*

When Reset is de-asserted:

```
/* Stay in secure state */
R14_svc = UNPREDICTABLE value
SPSR_svc = UNPREDICTABLE value
CPSR [4:0] = 0b10011 /* Enter supervisor mode */
CPSR [5] = 0 /* Execute in ARM state */
CPSR [6] = 1 /* Disable fast interrupts */
CPSR [7] = 1 /* Disable interrupts */
CPSR [8] = 1 /* Disable imprecise aborts */
CPSR [9] = Secure EE-bit /* store value of Secure Control Register bit[25] */
CPSR[24] = 0 /* Clear J bit */
if high vectors configured then
    PC = 0xFFFF0000
else
    PC = 0x00000000
```

#### Undefined instruction

On an undefined instruction:

```
/* secure state is unchanged */
R14_und = address of the next instruction after the undefined instruction
SPSR_und = CPSR
CPSR [4:0] = 0b11011 /* Enter undefined Instruction mode */
CPSR [5] = 0 /* Execute in ARM state */
CPSR [7] = 1 /* Disable interrupts */
CPSR [9] = Secure EE-bit /* store value of secure Control Reg[25] */
CPSR[24] = 0 /* Clear J bit */
if high vectors configured then
    PC = 0xFFFF0004
else
    PC = Secure_Base_Address + 0x00000004
```

#### Software Interrupt exception

On a SVC:

```
/* secure state is unchanged */
R14_svc = address of the next instruction after the SVC instruction
SPSR_svc = CPSR
CPSR [4:0] = 0b10011 /* Enter supervisor mode */
CPSR [5] = 0 /* Execute in ARM state */
CPSR [7] = 1 /* Disable interrupts */
CPSR [9] = Secure EE-bit /* store value of secure Control Reg[25] */
CPSR[24] = 0 /* Clear J bit */
if high vectors configured then
    PC = 0xFFFF0008
else
    PC = Secure_Base_Address + 0x00000008
```

#### External Prefetch Abort

On an external prefetch abort:

```
/* secure state is unchanged */
if SCR[3]=1 /* external prefetch aborts trapped to Secure Monitor mode */
    R14_mon = address of the aborted instruction + 4
    SPSR_mon = CPSR
    CPSR [4:0] = 0b10110 /* Enter Secure Monitor mode */
    CPSR [5] = 0 /* Execute in ARM state */
    CPSR [6] = 1 /* Disable fast interrupts */
    CPSR [7] = 1 /* Disable interrupts */
    CPSR [8] = 1 /* Disable imprecise aborts */
    CPSR [9] = Secure EE-bit /* store value of secure Control Reg[25] */
    CPSR[24] = 0 /* Clear J bit */
    PC = Monitor_Base_Address + 0x0000000C
Else
    R14_abt = address of the aborted instruction + 4
    SPSR_abt = CPSR
    CPSR [4:0] = 0b10111 /* Enter abort mode */
    CPSR [5] = 0 /* Execute in ARM state */
    CPSR [7] = 1 /* Disable interrupts */
    CPSR [8] = 1 /* Disable imprecise aborts */
    CPSR [9] = Secure EE-bit /* store value of secure Control Reg[25] */
    CPSR[24] = 0 /* Clear J bit */
    if high vectors configured then
        PC = 0xFFFF000C
    else
        PC = Secure_Base_Address + 0x0000000C
```

#### Internal Prefetch Abort

On an internal prefetch abort:

```
/* secure state is unchanged */
R14_abt = address of the aborted instruction + 4
SPSR_abt = CPSR
CPSR [4:0] = 0b10111 /* Enter abort mode */
CPSR [5] = 0 /* Execute in ARM state */
CPSR [7] = 1 /* Disable interrupts */
CPSR [8] = 1 /* Disable imprecise aborts */
CPSR [9] = Secure EE-bit /* store value of secure Control Reg[25] */
CPSR[24] = 0 /* Clear J bit */
if high vectors configured then
    PC = 0xFFFF000C
else
    PC = Secure_Base_Address + 0x0000000C
```

#### External Data Abort

On an External Precise Data Abort or on an External Imprecise Abort with CPSR[8]=0 (A bit):

```
/* secure state is unchanged */

if SCR[3]=1 /* external aborts trapped to Secure Monitor mode */
    R14_mon = address of the aborted instruction + 8
    SPSR_mon = CPSR
    CPSR [4:0] = 0b10110 /* Enter Secure Monitor mode */
    CPSR [5] = 0 /* Execute in ARM state */
    CPSR [6] = 1 /* Disable fast interrupts */
    CPSR [7] = 1 /* Disable interrupts */
    CPSR [8] = 1 /* Disable imprecise aborts */
    CPSR [9] = Secure EE-bit /* store value of secure Control Reg[25] */
    CPSR[24] = 0 /* Clear J bit */
    PC = Monitor_Base_Address + 0x00000010
Else /* external Aborts trapped in abort mode */
    R14_abt = address of the aborted instruction + 8
    SPSR_abt = CPSR
    CPSR [4:0] = 0b10111 /* Enter abort mode */
    CPSR [5] = 0 /* Execute in ARM state */
    CPSR [7] = 1 /* Disable interrupts */
    CPSR [8] = 1 /* Disable imprecise aborts */
    CPSR [9] = Secure EE-bit /* store value of secure Control Reg[25]  */
    CPSR[24] = 0 /* Clear J bit  */
    if high vectors configured then
        PC = 0xFFFF0010
    else
        PC = Secure_Base_Address + 0x00000010
```

#### Internal Data Abort

On an Internal Data Abort. All aborts that are not external aborts, i.e. data aborts on L1 memory management occurring when a fault is detected in MMU:

```
/* secure state is unchanged */
R14_abt = address of the aborted instruction + 8
SPSR_abt = CPSR
CPSR [4:0] = 0b10111 /* Enter abort mode */
CPSR [5] = 0 /* Execute in ARM state */
CPSR [7] = 1 /* Disable interrupts */
CPSR [8] = 1 /* Disable imprecise aborts  */
CPSR [9] = Secure EE-bit /* store value of secure Control Reg[25]  */
CPSR[24] = 0 /* Clear J bit  */
if high vectors configured then
```

```
    PC = 0xFFFF0010
else
    PC = Secure_Base_Address + 0x00000010
```

#### Interrupt request (IRQ) exception

On an Interrupt Request, and CPSR[7]=0, I bit:

```
/* secure state is unchanged */
if SCR[1]=1    /* IRQ trapped in Secure Monitor mode */
    R14_mon = address of the next instruction to be executed + 4
    SPSR_mon = CPSR
    CPSR [4:0] = 0b10110 /* Enter Secure Monitor mode */
    CPSR [5] = 0 /* Execute in ARM state */
    CPSR [6] = 1 /* Disable fast interrupts */
    CPSR [7] = 1 /* Disable interrupts */
    CPSR [8] = 1 /* Disable imprecise aborts  */
    CPSR [9] = Secure EE-bit /* store value of secure Control Reg[25]  */
    CPSR[24] = 0 /* Clear J bit  */
    PC = Monitor_Base_Address + 0x00000018
else
    R14_irq = address of the next instruction to be executed + 4
    SPSR_irq = CPSR
    CPSR [4:0] = 0b10010 /* Enter IRQ mode */
    CPSR [5] = 0 /* Execute in ARM state */
    CPSR [7] = 1 /* Disable interrupts */
    CPSR [8] = 1 /* Disable imprecise aborts  */
    CPSR [9] = Secure EE-bit /* store value of secure Control Reg[25]  */
    CPSR[24] = 0 /* Clear J bit  */
    if VE == 0 /* Core with VIC port only */
        if high vectors configured then
            PC = 0xFFFF0018
        else
            PC = Secure_Base_Address + 0x00000018
    else
        PC = IRQADDR
```

#### Fast Interrupt Request (FIQ) exception

On a Fast Interrupt Request, and CPSR[6]=0, F bit:

```
/* secure state is unchanged */
if SCR[2]=1 /* FIQ trapped in Secure Monitor mode */
    R14_mon = address of the next instruction to be executed + 4
    SPSR_mon = CPSR
    CPSR [4:0] = 0b10110 /* Enter Secure Monitor mode */
    CPSR [5] = 0 /* Execute in ARM state */
    CPSR [6] = 1 /* Disable fast interrupts */
    CPSR [7] = 1 /* Disable interrupts */
    CPSR [8] = 1 /* Disable imprecise aborts  */
    CPSR [9] = Secure EE-bit /* store value of secure Control Reg[25]  */
    CPSR[24] = 0 /* Clear J bit  */
    PC = Monitor_Base_Address + 0x0000001C
else
    R14_fiq = address of the next instruction to be executed + 4
    SPSR_fiq = CPSR
    CPSR [4:0] = 0b10001 /* Enter FIQ mode */
    CPSR [5] = 0 /* Execute in ARM state */
    CPSR [6] = 1 /* Disable fast interrupts */
    CPSR [7] = 1 /* Disable interrupts */
    CPSR [8] = 1 /* Disable imprecise aborts  */
    CPSR [9] = Secure EE-bit /* store value of secure Control Reg[25]  */
    CPSR[24] = 0 /* Clear J bit  */
    if high vectors configured then
```

```
        PC = 0xFFFF001C
    else
        PC = Non_Secure_Base_Address + 0x0000001C
```

#### *Secure Monitor Call Exception*

On a SMC:

```
If (UserMode) /* undefined instruction */
    R14_und = address of the next instruction after the SMC instruction
    SPSR_und = CPSR
    CPSR [4:0] = 0b11011 /* Enter undefined instruction mode */
    CPSR [5] = 0 /* Execute in ARM state */
    CPSR [7] = 1 /* Disable interrupts */
    CPSR [9] = Secure EE-bit /* store value of secure Control Reg[25]  */
    CPSR[24] = 0 /* Clear J bit  */
    If high vectors configured then
        PC = 0xFFFF0004
    else
        PC = Secure_Base_Address + 0x00000004
else
    R14_mon = address of the next instruction after the SMC instruction
    SPSR_mon = CPSR
    CPSR [4:0] = 0b10110 /* Enter Secure Monitor mode */
    CPSR [5] = 0 /* Execute in ARM state */
    CPSR [6] = 1 /* Disable fast interrupts */
    CPSR [7] = 1 /* Disable interrupts */
    CPSR [8] = 1 /* Disable imprecise aborts */
    CPSR [9] = Secure EE-bit /* store value of secure Control Reg[25]  */
    CPSR[24] = 0 /* Clear J bit  */
    PC = Monitor_Base_Address + 0x00000008    /* SMC vectored to the */
                                              /*conventional SVC vector */
```

### 2.12.17 Exception priorities

When multiple exceptions arise at the same time, a fixed priority system determines the order that they are handled. Table 2-9 lists the order of exception priorities.

**Table 2-9 Exception priorities**

| Priority | | Exception |
|---|---|---|
| Highest | 1 | Reset |
| | 2 | Precise Data Abort |
| | 3 | FIQ |
| | 4 | IRQ |
| | 5 | Prefetch Abort |
| | 6 | Imprecise Data Abort |
| Lowest | 7 | BKPT<br>Undefined Instruction<br>SVC<br>SMC |

Some exceptions cannot occur together:

- The BKPT, undefined instruction, SMC, and SVC exceptions are mutually exclusive. Each corresponds to a particular, non-overlapping, decoding of the current instruction.

- When FIQs are enabled, and a precise Data Abort occurs at the same time as an FIQ, the processor enters the Data Abort handler, and proceeds immediately to the FIQ vector.

  A normal return from the FIQ causes the Data Abort handler to resume execution.

  Precise Data Aborts must have higher priority than FIQs to ensure that the transfer error does not escape detection. You must add the time for this exception entry to the worst-case FIQ latency calculations in a system that uses aborts to support virtual memory.

  The FIQ handler must not access any memory that can generate a Data Abort, because the initial Data Abort exception condition is lost if this happens.

**Note**

If the data abort is a precise external abort and bit 3 (EA) of SCR is set, the processor enters Secure Monitor mode where aborts and FIQs are disabled automatically. Therefore, the processor does not proceed to FIQ vector immediately afterwards.