ARM946E-S Revision r1p1 Errata List

IP Products Division

CPU Group

Document number:	ARM946-PRDC-000592 5.0
Date of Issue:	12th March 2007
Author:	Andrew Burdass
Authorised by:	Steven Poole

Abstract

This document describes the known errata in the ARM946E-S revision r1p1 design.

Keywords

ARM946, ARM9E, Errata

This is a working document throughout the product lifecycle and, as such, the content may be modified as new information is uncovered.

Proprietary notice

Document confidentiality status

This document is Non Confidential.

Web address

http://www.arm.com/

Feedback on the product

If you have any comments or suggestions about this product, contact your supplier giving:

- The product name
- A concise explanation of your comments.

Feedback on this document

If you have any comments on about this document, please send email to mailto:support-cores@arm.com giving:

- The document title
- The documents number
- The page number(s) to which your comments refer
- A concise explanation of your comments

General suggestion for additions and improvements are also welcome.

Contents

1 ABOUT THIS DOCUMENT

1.1 Current History

Issue	Date	By	Change
5.0	6th February 2007		Add new errata:

1.2 References

This document refers to the following documents.

Ref.	Document No	Author(s)	Title
1	ARM DDI 0201	ARM	ARM946E-S Technical Reference Manual
2	ARM DDI 0165	ARM	ARM9E-S Technical Reference Manual

1.3 Scope

This document describes the errata discovered in the implementation of the ARM946E-S revision r1p1, categorised by level of severity. Each description includes:

- where the implementation deviates from the specification
- the conditions under which erroneous behaviour will occur
- the implications of the erratum with respect to typical applications
- the application and limitations of a 'work-around' where possible
- the status of corrective action.

1.4 Terms and Abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
Bounced Coprocessor Instruction	An invalid coprocessor instruction that results in the Undefined Instruction trap being taken.
Breakpoint	A debugging mechanism used to halt execution due to an **instruction fetch**. For the breakpoint to cause debug state entry, the instruction must reach execution stage of the pipeline, but it will be prevented from executing. This enables a debugger to observe the state prior to that instruction's execution. A breakpoint can be made to occur by either appropriate triggering of the Embedded ICE watchpoint units or by asserting the **DBGIEBKPT** signal during an instruction fetch.
DBGDEWPT	EmbeddedICE watchpoint indicator. This signal identifies the current memory access with a debug condition.
DBGDIEBKPT	EmbeddedICE breakpoint indicator. This signal identifies the current memory access with a debug condition.
DBGRQ	Debug request. An internally synchronised input used to signal the processor to enter debug state once the current executing instruction completes.
Debugger/Debugging Tool	A debugging system that includes a program used to detect, locate and correct software faults, together with custom hardware that supports software debugging.
Single-stepping	A debugging operation used to step through the flow of program execution, instruction by instruction. This can be implemented by using a single **watchpoint unit** configured to cause a **breakpoint** on the next instruction to be executed.
Watchpoint	A debugging mechanism used to halt execution due to a memory **data access**. Watchpoints cause debug state entry once the instruction causing the data access and the directly following instruction fully complete; this may include accepting state changes due to pending exceptions. Watchpoints enable a debugger to observe program memory changes, such as updates to variables. A watchpoint can be made to occur by either appropriate triggering of the Embedded ICE watchpoint units or by asserting the **DBGDEWPT** signal during a data memory access.
Watchpoint Unit	Custom EmbeddedICE hardware capable of triggering a breakpoint or watchpoint when all its comparators match. Each watchpoint unit has several registers to configure the type of comparison desired, enabling matches against any value on the address bus, and/or the data bus and/or various bus control signals.

2 CATEGORISATION OF ERRATA

The errata listed in this document are split into three groups:

Category 1 Features which are impossible to work around and severely restrict the use of the device in all or the majority of applications rendering the device unusable.

Category 2 Features which contravene the specified behaviour and may limit or severely impair the intended use of specified features but does not render the device unusable in all or the majority of applications.

Category 3 Features that were not the originally intended behaviour but should not cause any problems in applications.

2.1 Errata Summary

The errata associated with this product are categorised in the following way. Numbers in brackets after the errata description indicate the order in which the errata were found chronologically. The ARM9E-S errata are included and have their own numbers independent from ARM946E-S specific errata.

Category 1	No known errata
Category 2	LDM of user mode registers (ARM9E - 6) Debug Request coincident with Pipeline Hazards (ARM9E - 7) Data Abort and Watchpoint with Breakpoint Following (ARM9E - 8) Watchpoint coincident with Debug Request (ARM9E - 9) External breakpoints during pipeline stall (ARM9E - 10) Incorrect register forwarding for Load and Store double instructions (ARM9E - 12) Data Cache clean from debug (1) Data Cache clean and invalidate with full write buffer (6)
Category 3	Watchpoint and Prefetch Abort (ARM9E - 11) Incorrect indication of Instruction TCM absent (3) Enabling Instruction TCM when executing code from a cacheable region (4) HLOCK indication for cache line fills (5) Coprocessor 15 operations immediately following a MSR instruction may be executed in User mode (7)

3 CATEGORY 1 ERRATA

There are no known errata in this group.

4 CATEGORY 2 ERRATA

4.1 LDM of user mode registers (ARM9E - 6)

4.1.1 Summary

Under specific conditions, a LDM to user mode registers will not operate correctly. These instructions take the form:

```
LDM{<cond>}<addressing_mode> <Rn>,<registers_without_pc>^
```

These instructions are only used in system code and not in application code.

4.1.2 Description

A LDM to user mode registers performs a load to the user mode registers whilst the processor is in a privileged mode.

Under specific conditions (see below), this instruction will fail to operate correctly. This results in not all the registers in the register list being written correctly. It may also cause further failures, dependent on the construction of the memory system, since the data memory request signals are driven in an incorrect manner in the failing situation.

Example of failing instruction

```
LDM sp,{sp,lr}^
```

4.1.3 Conditions

This errata exists in the following circumstances:

A LDM to user mode registers, where:

1. The base register is register 8 or greater and
2. The base register is the first register (lowest register number) in the register list and
3. There is more than one register in the list.

Notes:

In all privileged modes this errata exists if the base register is 8 or greater. This is not restricted to FIQ mode.

Instructions of the form

```
LDM{<cond>}<addressing_mode> <Rn>,<registers_and_pc}^
```

operate correctly since these are not LDM to user mode register instructions. These instructions perform a return from exception.

Example of failing instructions:

```
LDMIA sp,{sp,lr}^
LDMIA r8,{r8-r10}^
```

Examples that do NOT fail:

```
LDMIA sp,{r8,sp,lr}^    ; OK, since first register is not the base register
LDMIA r5,{r5,sp,lr}^    ; OK, since base register is < r8
```



```
LDMIA sp,{sp}^          ; OK, since only one register in the list
LDMIA sp,{sp,lr,pc}^    ; OK, since PC in the list and hence a return from
                        ; exception instruction, not a load of user
                        ; mode registers.
```

4.1.4 Implications

This errata only applies to hand crafted assembler code, as the ARM compiler does not generate such instructions. The use of this instruction is typically limited to a few places in exception handlers, thus limiting the scope of this erratum.

4.1.5 Workaround

This instruction is only used by hand crafted assembler code. The ARM compiler will not generate this instruction. To work-around this errata the `LDM` should be split into two separate instructions.

For example, the instruction:

```
LDMIA sp,{sp,lr}^
```

should be split into:

```
LDMIA sp,{sp}^
LDMIA sp,{lr}^
```

And:

```
LDMIA r8,{r8-lr}^
```

should be split into:

```
LDMIA r8,{r8}^
LDMIA r8,{r9-lr}^
```

4.1.6 Corrective Action

ARM has no plans to correct this erratum.

4.2 Debug Request coincident with Pipeline Hazards (ARM9E - 7)

4.2.1 Summary

The processor may return to normal program execution at an incorrect point if **EDBGRQ** or the scan chain created debug request is asserted whilst it is performing certain tasks.

4.2.2 Conditions

There are three scenarios in which this erratum can occur:

1) Debug Request occurs whilst the processor is waiting for a coprocessor instruction to be completed by a coprocessor or
2) Debug Request occurs whilst the recognition of a Data Abort is in progress or
3) Debug Request occurs whilst the recognition of an instruction causing a watchpoint is in progress.

Note Recognition of a Data Abort is said to be in progress if the last memory access asserted the **DABORT** signal, but the processor has not yet begun execution at the Data Abort vector.

Note Recognition of a watchpoint is said to be in progress if the last memory access caused a watchpoint, but debug entry has not yet completed.

If the above conditions are met then the debug entry mechanism fails to behave in the defined manner and the device may return from debug and execute from an incorrect address.

4.2.3 Implications

For each scenario the following implications are expected given the above conditions:

Busy-waiting Coprocessor Instructions

In this form the debug request supersedes the currently executing coprocessor instruction, which would normally not occur. As such the PC is not the correct value as debug entry proceeds. Correspondingly, on return from debug the standard return address calculation produces an incorrect address and thus unintended or unpredictable device behaviour may result.

Data Aborts

In this form the debug request causes correct debug entry and exit. However, the link register address calculation for the return from the Data Abort handler is incorrect. Correspondingly, on return from the Data Abort handler unintended or unpredictable device behaviour may result.

Watchpoints

In this form the debug request causes debug entry first, but the watchpoint is still pending and has yet to execute the next instruction before it takes full effect. As debug entry has already occurred this next instruction will be the first instruction within debug mode to be executed. After this instruction executes, debug entry occurs for a second time, even though the device is already in debug, and results in the premature exiting of debug. Consequently, the return from debug is to an incorrect address and thus unintended or unpredictable device behaviour may result.

4.2.4 Workarounds

There is no practical workaround for this erratum. This is due to the difficulty in getting a debugging tool to recognise the symptoms of this erratum and take the appropriate corrective action. However the likelihood of failure with this mechanism is extremely low and the impact of failure is also low as it affects debugging operations only.

4.2.5 Corrective Action

ARM has no plans to correct this erratum.

4.3 Data Abort and Watchpoint with Breakpoint Following (ARM9E - 8)

4.3.1 Summary

The processor may fail to execute the abort handler if a data abort occurs on a watchpointed instruction and a breakpoint is in the execution pipeline.

4.3.2 Conditions

The conditions for this erratum are:

1) An instruction that causes both a Data Abort and a watchpoint to occur and

2) The execution of the following instruction will cause a breakpoint to occur

Note There should be no data dependency between the two instructions such that a pipeline interlock occurs. If there is such data dependency then the processor behaves correctly.

If the above conditions are met then the Data Abort may be missed.

4.3.3 Implications

In this erratum Data Abort entry is halted by debug entry and this causes the state indicating a Data Abort to be lost on return from debug. Hence the Data Abort handler will fail to be invoked. This may result in unintended or unpredictable device behaviour.

4.3.4 Workarounds

There is no practical workaround for this erratum. This is due to the difficulty in getting a debugging tool to recognise the symptoms of this erratum and take the appropriate corrective action. However the likelihood of failure with this mechanism is extremely low and the impact of failure is also low as it affects debugging operations only.

4.3.5 Corrective Action

ARM has no plans to correct this erratum.

4.4 Watchpoint coincident with Debug Request (ARM9E - 9)

4.4.1 Summary

The processor can falsely exit debug state and continue execution if **EDBGRQ** or the scan chain created debug request is asserted whilst debug entry is occurring.

4.4.2 Conditions

A combination of two conditions is required to cause this erratum:

1) Debug request is asserted and

2) An instruction that generates a watchpoint is executing

If the above conditions are met then the debug entry mechanism fails to behave in the defined manner and the device may return from debug and execute from the incorrect address.

4.4.3 Implications

In this erratum, the debug request takes affect before the complete recognition of the watchpoint. This may result in unreliable debug entry, the watchpoint being missed and premature exit of debug state. Thus unintended or unpredictable device behaviour may result.

4.4.4 Workarounds

There is no practical workaround for this erratum. This is due to the difficulty in getting a debugging tool to recognise the symptoms of this erratum and take the appropriate corrective action. However the likelihood of failure with this mechanism is extremely low and the impact of failure is also low as it affects debugging operations only.

4.4.5 Corrective Action

ARM has no plans to correct this erratum.

4.5 External breakpoints during pipeline stall (ARM9E - 10)

4.5.1 Description

If the processor pipeline stalls between an external breakpoint (**DBGIEBKPT**) generated from the memory system, and the breakpoint instruction being decoded, the breakpoint is ignored.

4.5.2 Conditions

When the core is configured into stopping debug mode, and executing ARM or Thumb code, then if a pipeline stall occurs between an external breakpoint (**DBGIEBKPT**) being returned from the memory system and the BKPT instruction being decoded, the breakpoint that should be generated by that instruction is lost.

Some example code sequences that would highlight this problem are as follows:

[original example, LDM/STM causes pipeline stall]

```
STM r0, {r0-r5}
BKPT
```

and

[similar, NOP is in pseudo-decode and doesn't advance into Ex when BKPT returned]

```
STM r0, {r0-r5}
NOP
BKPT
```

and

[similar, multi-cycle instruction]

```
MRS r0, cpsr
BKPT
```

and

```
MUL r0, r1, r2
BKPT
```

and

[interlock]

```
LDR  r0, [r1]
ADD  r2, r0, r0
BKPT
```

4.5.3 Implications

External breakpoints may not halt program execution as expected.

4.5.4 Workaround

Software breakpoints may be used to in place of hardware breakpoints. This requires the debug tools to replace the target instruction with an undefined instruction and use vector catching to detect this.

4.5.5 Corrective Action

ARM has no plans to correct this erratum.

4.6 Incorrect register forwarding for Load and Store double instructions (ARM9E - 12)

4.6.1 Description

Any STRD or LDRD with base register writeback, which was immediately preceded by a load multiple where the base register was the penultimate register in the list, causes the execute stage forwarding path for the following instruction to contain an incorrect value for the base register.

This does not affect any of the other forwarding paths, or the actual value written to the register bank.

4.6.2 Conditions

This situation occurs for STRD or LDRD instructions with base register writeback that are immediately preceded by a load multiple where the base register is the penultimate register in the list. If the base register value is used in the next instruction an incorrect register value will be used.

The following code segment highlights this problem:

```
LDMIA r0, {r8-r9}
STRD  r0, [r8], #0x4
MOV   r7, r8            ; <-- incorrect value of r8 forwarded here
CMP   r7, r8            ; <-- CMP fails as 'this' r8 differs
```

4.6.3 Implications

This erratum only applies to hand crafted assembler code, as the ARM compiler does not generate such instruction sequences.

This behaviour can result in incorrect execution of code sequences that use in-line assembler STRD/LDRD instructions.

4.6.4 Workaround

This instruction sequence is only used by hand crafted assembler code. The ARM compiler does not currently generate STRD or LDRD instructions.

To work-around this errata the forwarding paths should be broken by the addition of NOP instructions after the load multiple or LDRD/STRD instructions in the code sequence.

```
LDMIA r0, {r8-r9}
MOV   r0, r0            ; NOP
STRD  r0, [r8], #0x4
MOV   r7, r8
CMP   r7, r8
```

OR

```
LDMIA r0, {r8-r9}
STRD  r0, [r8], #0x4
MOV   r0, r0            ; NOP
MOV   r7, r8
CMP   r7, r8
```

4.6.5 Corrective Action

ARM has no plans to correct this erratum.

4.7 Data Cache clean from debug (1)

4.7.1 Description

Data cache cleans can be initiated via cache line evictions, cache maintenance operations from the ARM9E-S core to CP15, or the JTAG debug scan chain 15.

Data Cache cleans initiated via JTAG debug scan chain 15 copy the correct cache contents to main memory, but may clear the dirty bits for an unrelated cache entry.

The ARM debug tools do not use this functionality, and it is unlikely to be used by other tools.

Cache cleaning via cache line evictions and cache maintenance operations from the ARM9E-S core to CP15 function correctly.

4.7.2 Conditions

This situation only occurs for data cache clean operations that are performed via JTAG debug scan chain 15.

Dirty cache entries can only exist in situations where address regions are or have been marked as Write-Back cache regions. The cache location that is marked as clean will depend on the value present on the Data Address bus of the ARM9E core as this value will be used as the index for the cache entry that is marked as clean.

4.7.3 Implications

This behaviour will result in data being lost if the incorrectly updated cache entry was also dirty.

As this functionality is not used by the ARM debug tools, this behaviour will not occur whilst using the ARM debug tools.

4.7.4 Workaround

The ARM debug tools do not use the cache clean functionality.

If other tools use this functionality, it is necessary to ensure that the value on the Data Address bus of the ARM9E core matches the index for the cache entry to be cleaned. This may be achieved by initialising an ARM9E register with the correct index value and then performing a debug load using the initialised register as the address. These instructions can be executed by the ARM9E JTAG debug scan chain 1, before selecting JTAG debug scan chain 15 for the cache clean operation.

This is described in Appendix C of the ARM9E-S Technical Reference Manual (ARM DDI 0165).

4.7.5 Corrective Action

ARM has no plans to correct this erratum.

4.8 Data Cache clean and invalidate with full write buffer (6)

4.8.1 Summary

A cache clean and invalidate operation may not mark a cache line as invalid if the write buffer is full.

4.8.2 Description

The problem is observed when a cache clean and invalidate operation is performed while the write buffer is full. The cache line concerned is clean at the point the cache maintenance operation is started, but the write buffer is full. The cache entry is not marked as invalid. A subsequent access to the same address will result in a cache hit. In the case of a read the cache data will be used rather than fetching a cache line from the system memory.

Clean and invalidate operations of dirty cache lines when the write buffer is full function correctly.

4.8.3 Conditions

1. A cache clean and invalidate operation is performed
2. The cache entry for the operation is clean
3. The write buffer is full

4.8.4 Implications

This may affect coherency between the data cache and external memory. Subsequent accesses to the memory location which has not been invalidated will be fetched from the cache, rather than from external memory which can contain newer data.

4.8.5 Workaround

Draining the write buffer before each clean and invalidate operation will ensure that the cache entry is correctly invalidated. For example:

```
MCR p15, 0, Rd, c7, c10, 4 ; Drain write buffer
MCR p15, 0, Rd, c7, c14, 1 ; Clean and invalidate by address
```

Alternatively separate clean and invalidate operations will ensure correct operation.

```
MCR p15, 0, Rd, c7, c10, 1 ; Clean single entry
MCR p15, 0, Rd, c7, c6, 1  ; Invalidate by address
```

4.8.6 Corrective Action

ARM has no plans to correct this erratum.

5 CATEGORY 3 ERRATA

5.1 Watchpoint and Prefetch Abort (ARM9E - 11)

5.1.1 Summary

The processor can enter debug state at a more advanced point in program execution than expected if a Prefetch Abort occurs whilst a watchpoint is being processed.

5.1.2 Conditions

A combination of two conditions is required to cause this erratum:

1. An instruction that will cause a watchpoint has been executed and
2. The second following instruction will cause a Prefetch Abort

If the above conditions are met then the prefetch abort handler may have been entered before the processor enters debug state.

5.1.3 Implications

In this erratum the Prefetch Abort is prematurely recognised, that is before debug entry occurs. This behaviour is unintended and is not documented, however it does not result in incorrect device behaviour.

5.1.4 Workaround

No workaround is required for this erratum since the device operates correctly with respect to normal operation.

5.1.5 Corrective Action

This behaviour will be documented in future specifications.

5.2 Incorrect indication of Instruction TCM absent (3)

5.2.1 Description

The ARM946E-S Tightly-Coupled Memory (TCM) size register contains bits that indicate that the Instruction or Data TCM is absent. The setting of the Instruction TCM absent bit (bit 2 of the register) will be incorrect if either of the TCMs is absent. The size of the Instruction TCM will be correctly reported.

5.2.2 Conditions

The Instruction TCM absent bit will be incorrect if only one of the TCMs is present.

If the Instruction TCM is present and the Data TCM is absent, the Instruction TCM absent bit will be set.

If the Instruction TCM is absent and the Data TCM is present, the Instruction TCM absent bit will be clear.

5.2.3 Implications

Application software that uses this bit to determine the presence of Instruction TCM will return an incorrect result.

5.2.4 Workarounds

The Instruction TCM size bits (bits [9:6] of the TCM size register should be used to determine the presence of Instruction TCM. If these bits are all zero, then the instruction TCM is absent.

5.2.5 Corrective Action

ARM has no plans to correct this erratum.

5.3 Enabling Instruction TCM when executing code from a cacheable region (4)

5.3.1 Summary

If execution from the Instruction Tightly-Coupled Memory (TCM) is enabled when executing code from a cacheable region, unpredictable behaviour will result. This occurs if Instruction TCM Load Mode is disabled leaving the Instruction TCM enabled, or if the Instruction TCM is enabled having been previously initialised.

5.3.2 Conditions

For this behaviour to be exhibited, the following conditions must exist:

1. The cacheable code is being executed from the Instruction TCM address space.

2. The instruction cache is enabled, and the instruction which enables the Instruction TCM is in the cache or is streamed to the core during an instruction linefill.

5.3.3 Implications

Unpredictable program execution will result from this erratum.

5.3.4 Workaround

After enabling the Instruction TCM, a branch instruction should be used to force instruction fetches from the TCM rather than the cache.

For example, if the code to enable the Instruction TCM is currently:

```
            MRC     p15, 0, r2, c1, c0, 0       ; Read Control Register
            BIC     r2, r2, #&80000             ; Clear ITCM Loadmode
            MCR     p15, 0, r2, c1, c0, 0       ; Enable ITCM
RunFromITCM MOV     r2, #0                      ; code continues
```

Should now be:

```
            MRC     p15, 0, r2, c1, c0, 0       ; Read Control Register
            BIC     r2, r2, #&80000             ; Clear ITCM Loadmode
            MCR     p15, 0, r2, c1, c0, 0       ; Enable ITCM
            B       RunFromITCM
RunFromITCM MOV     r2, #0                      ; code continues
```


5.3.5 Corrective Action

ARM has no plans to correct this erratum.

5.4 HLOCK indication for cache line fills (5)

5.4.1 Summary

HLOCK may be asserted for the first transfer of a cache linefill if it immediately follows a locked (SWP) operation.

5.4.2 Description

The AHB specification recommends that an IDLE cycle is inserted on the AHB bus at the end of locked sequence. Under certain circumstances the ARM946E-S does not do this and starts a new instruction linefill immediately after the address cycle of the write transfer of the SWP instruction. As HLOCK is used in the cycle before the transfer to which it refers, this can cause a bus arbiter to believe that the bus transfer is locked.

5.4.3 Conditions

For this behaviour to be exhibited, the following conditions must exist:

1. A SWP(B) instruction must be executed.

2. The instruction following the SWP causes a linefill on the AHB bus.

5.4.4 Implications

Treating the cache linefill as a locked transfer may result in a degradation of system performance as the linefill will prevent a higher priority bus master gaining access to the bus.

5.4.5 Workaround

No workaround exists for this erratum. This erratum should not cause a system failure.

5.4.6 Corrective Action

ARM has no plans to correct this erratum.

5.5 Coprocessor 15 Operations immediately following a MSR instruction may be executed in User mode (7)

5.5.1 Summary

The 2 instructions immediately following an MSR instruction that changes the mode from a privileged mode to User mode may be executed as if the processor was still in a privileged mode.

5.5.2 Description

An MSR instruction allows the processor mode to be changed. If an MSR is used to change from a Privileged mode to User mode, the two instructions immediately following the MSR instruction will have already been fetched

and will be present in the pipeline. If either of these instructions are Coprocessor 15 operations (including cache maintenance, System control and MPU regions) they will be executed as if the processor was in a privileged mode.

This erratum does not effect the execution of instructions after changing from User mode to privileged mode as the pipeline is flushed in these cases.

This erratum does not affect the return from exceptions as this usually involves an operation to change the program counter. Changing the program counter causes the pipeline to be flushed and so subsequent Coprocessor 15 operations will be executed with the correct privileges.

5.5.3 Conditions

For this behaviour to be exhibited, the following conditions must exist:

1. An MSR instruction is executed that changes from a privileged mode to User mode
2. The instruction following the MSR instruction is a Coprocessor 15 operation
   or
   the instruction following the MSR instruction does not modify the program counter and the second instruction after the MSR instruction is a Coprocessor 15 operation.

5.5.4 Implications

The ARM Architecture Reference Manual recommends that an Instruction Memory Barrier (IMB) sequence is executed after using an MSR to change from a privileged mode to User mode.

If this is not done the erratum allows User mode access to all Coprocessor 15 operations including reading and writing of the System Control register. The code sequence that generates this behaviour would be very unusual. In most cases changing from User to privileged mode is done on return from a procedure, and this would use an instruction that would cause the pipeline to flush. It is most unusual for User code to immediately follow privileged code, where the last privileged instruction is to change to User mode.

5.5.5 Workaround

Coprocessor 15 operations should not immediately follow MSR instructions that change to User mode. This erratum can be avoided by placing a branch instruction immediately after an MSR instruction

```
      MSR   CPSR_c, Rm
      B     NextInst
NextInst
      …….
```

Alternatively two NOP instructions can be inserted following the MSR instruction. This requires more memory, but executes in less cycles than the branch insertion.

```
      MSR   CPSR_c, Rm
      NOP
      NOP
NextInst
```

