```
MRC p15, 0, <Rd>, c3, c0, 0    ; Read Domain Access Control Register
MCR p15, 0, <Rd>, c3, c0, 0    ; Write Domain Access Control Register
```

3.2.17 c5, Data Fault Status Register

The purpose of the Data Fault Status Register is to hold the source of the last data fault.

Table 3-61 lists the purposes of the individual bits in the Data Fault Status Register.

The Data Fault Status Register is:

- in CP15 c5
- a 32-bit read/write register banked for Secure and Non-secure worlds
- accessible in privileged modes only.

Figure 3-36 shows the bit arrangement in the Data Fault Status Register.

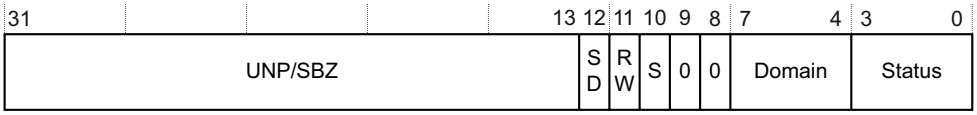

Figure 3-36 Data Fault Status Register format

Table 3-61 shows how the bit values correspond with the Data Fault Status Register functions.

Table 3-61 Data Fault Status Register bit functions

Bits	Field name	Function
[31:13]	-	UNP/SBZ.
[12]	SD	Indicates if an AXI Decode or Slave error caused an abort. This is only valid for external aborts. For all other aborts this Should Be Zero. See *Fault status and address* on page 6-34: 0 = AXI Decode error caused the abort, reset value 1 = AXI Slave error caused the abort.
[11]	RW	Indicates whether a read or write access caused an abort: 0 = Read access caused the abort, reset value 1 = Write access caused the abort.
[10]	S	Part of the Status field. See Bits [3:0] in this table. The reset value is 0.
[9:8]	-	Always read as 0. Writes ignored.

Table 3-61 Data Fault Status Register bit functions (continued)

Bits	Field name	Function
[7:4]	Domain	Indicates the domain from the 16 domains, D15-D0, is accessed when a data fault occurs. Takes values 0-15. The reset value is 0.
[3:0] with bit[10] = 0	Status	Indicates type of fault generated. See *Fault status and address* on page 6-34 for full details of Domain and FAR validity, and priorities: b0000 = no function, reset value b0001 = Alignment fault b0010 = Instruction debug event fault b0011 = Access Bit fault on Section b0100 = Instruction cache maintenance operation fault b0101 = Translation Section fault b0110 = Access Bit fault on Page b0111 = Translation Page fault b1000 = Precise external abort b1001 = Domain Section fault b1010 = no function b1011 = Domain Page fault b1100 = External abort on translation, first level b1101 = Permission Section fault b1110 = External abort on translation, second level b1111 = Permission Page fault.
[3:0] with bit[10] = 1	Status	Indicates type of fault generated. See *Fault status and address* on page 6-34 for full details of Domain and FAR validity, and priorities: b0000 = no function, reset value b0001 = no function b0010 = no function b0011 = no function b0100 = no function b0101 = no function b0110 = Imprecise external abort b0111 = no function b1000 = no function b1001 = no function b1010 = no function b1011 = no function b1100 = no function b1101 = no function b1110 = no function b1111 = no function.

Table 3-62 lists the results of attempted access for each mode.

Table 3-62 Results of access to the Data Fault Status Register

Secure Privileged		Non-secure Privileged		User
Read	Write	Read	Write	
Secure data	Secure data	Non-secure data	Non-secure data	Undefined exception

Note

When the SCR EA bit is set, see *c1, Secure Configuration Register* on page 3-52, the processor writes to the Secure Data Fault Status Register on a Secure Monitor entry caused by an external abort.

To use the Data Fault Status Register read or write CP15 with:

- Opcode_1 set to 0
- CRn set to c5
- CRm set to c0
- Opcode_2 set to 0.

For example:

```
MRC p15, 0, <Rd>, c5, c0, 0    ; Read Data Fault Status Register
MCR p15, 0, <Rd>, c5, c0, 0    ; Write Data Fault Status Register
```

3.2.18 c5, Instruction Fault Status Register

The purpose of the *Instruction Fault Status Register* (IFSR) is to hold the source of the last instruction fault.

Table 3-63 on page 3-67 lists the purposes of the individual bits in IFSR.

The Instruction Fault Status Register is:

- in CP15 c5
- a 32-bit read/write register banked for Secure and Non-secure worlds
- accessible in privileged modes only.

Figure 3-37 shows the bit arrangement of the Instruction Fault Status Register.

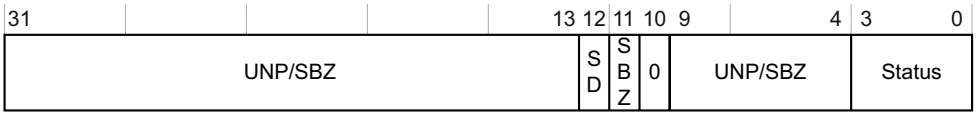

Figure 3-37 Instruction Fault Status Register format

Table 3-63 lists how the bit values correspond with the Instruction Fault Status Register functions.

Table 3-63 Instruction Fault Status Register bit functions

Bits	Field name	Function
[31:13]	-	UNP/SBZ.
[12]	SD	Indicates whether an AXI Decode or Slave error caused an abort. This bit is only valid for external aborts. For all other aborts this bit Should Be Zero. See *Fault status and address* on page 6-34: 0 = AXI Decode error caused the abort, reset value 1 = AXI Slave error caused the abort.
[11]	-	UNP/SBZ.
[10]	-	Part of the Status field, see bits [3:0] in this table. Always 0.
[9:4]	-	UNP/SBZ.
[3:0] with bit[10] = 0	Status	Indicates type of fault generated. See *Fault status and address* on page 6-34 for full details of Domain and FAR validity, and priorities: b0000 = no function, reset value b0001= Alignment fault b0010 = Instruction debug event fault b0011 = Access Bit fault on Section b0100 = no function b0101 = Translation Section fault b0110 = Access Bit fault on Page b0111 = Translation Page fault b1000 = Precise external abort b1001 = Domain Section fault b1010 = no function b1011 = Domain Page fault b1100 = External abort on translation, first level b1101 = Permission Section fault b1110 = External abort on translation, second level b1111 = Permission Page fault.

Table 3-64 lists the results of attempted access for each mode.

Table 3-64 Results of access to the Instruction Fault Status Register

Secure Privileged		Non-secure Privileged		User
Read	Write	Read	Write	
Secure data	Secure data	Non-secure data	Non-secure data	Undefined exception

Note

When the SCR EA bit is set, see *c1, Secure Configuration Register* on page 3-52, the processor writes to the Secure Instruction Fault Status Register on a Secure Monitor entry caused by an external abort.

To use the IFSR read or write CP15 with:

- Opcode_1 set to 0
- CRn set to c5
- CRm set to c0
- Opcode_2 set to 1.

For example:

```
MRC p15, 0, <Rd>, c5, c0, 1    ; Read Instruction Fault Status Register
MCR p15, 0, <Rd>, c5, c0, 1    ; Write Instruction Fault Status Register
```

3.2.19 c6, Fault Address Register

The purpose of the *Fault Address Register* (FAR) is to hold the *Modified Virtual Address* (MVA) of the fault when a precise abort occurs.

The FAR is:

- in CP15 c6
- a 32-bit read/write register banked for Secure and Non-secure worlds
- accessible in privileged modes only.

The Fault Address Register bits [31:0] contain the MVA that the precise abort occurred on. The reset value is 0.

Table 3-65 lists the results of attempted access for each mode.

Table 3-65 Results of access to the Fault Address Register

Secure Privileged		Non-secure Privileged		User
Read	Write	Read	Write	
Secure data	Secure data	Non-secure data	Non-secure data	Undefined exception

To use the FAR read or write CP15 with:

- Opcode_1 set to 0
- CRn set to c6
- CRm set to c0
- Opcode_2 set to 0.

For example:

```
MRC p15, 0, <Rd>, c6, c0, 0    ; Read Fault Address Register
MCR p15, 0, <Rd>, c6, c0, 0    ; Write Fault Address Register
```

A write to this register sets the FAR to the value of the data written. This is useful for a debugger to restore the value of the FAR.

The ARM1176JZF-S processor also updates the FAR on debug exception entry because of watchpoints, see *Effect of a debug event on CP15 registers* on page 13-34 for more details.

3.2.20 c6, Watchpoint Fault Address Register

Access to the Watchpoint Fault Address register through the system control coprocessor is deprecated, see *CP14 c6, Watchpoint Fault Address Register (WFAR)* on page 13-12.

3.2.21 c6, Instruction Fault Address Register

The purpose of the *Instruction Fault Address Register* (IFAR) is to hold the address of instructions that cause a prefetch abort.

The IFAR is:

- in CP15 c6
- a 32-bit read/write register banked for Secure and Non-secure worlds
- accessible in privileged modes only.

The Instruction Fault Address Register bits [31:0] contain the Instruction Fault MVA. The reset value is 0.

Table 3-66 lists the results of attempted access for each mode.

Table 3-66 Results of access to the Instruction Fault Address Register

Secure Privileged		Non-secure Privileged		User
Read	Write	Read	Write	
Secure data	Secure data	Non-secure data	Non-secure data	Undefined exception

To use the IFAR read or write CP15 with:

- Opcode_1 set to 0
- CRn set to c6
- CRm set to c0
- Opcode_2 set to 2.

For example:

```
MRC p15, 0, <Rd>, c6, c0, 2    ; Read Instruction Fault Address Register
MCR p15, 0, <Rd>, c6, c0, 2    ; Write Instruction Fault Address Register
```

A write to this register sets the IFAR to the value of the data written. This is useful for a debugger to restore the value of the IFAR.

3.2.22 c7, Cache operations

The purpose of c7 is to:

- control these operations:
  - clean and invalidate instruction and data caches, including range operations
  - prefetch instruction cache line
  - Flush Prefetch Buffer
  - flush branch target address cache
  - virtual to physical address translation.
- implement the *Data Synchronization Barrier* (DSB) operation
- implement the *Data Memory Barrier* (DMB) operation

- implement the *Wait For Interrupt* clock control function.

Note

Cache operations also depend on:

- the C, W, I and RR bits, see *c1, Control Register* on page 3-44.
- the RA and RV bits, see *c1, Auxiliary Control Register* on page 3-48.

The following cache operations globally flush the BTAC:

- Invalidate Entire Instruction Cache
- Invalidate Both Caches.

c7 consists of one 32-bit register that performs 28 functions. Figure 3-38 shows the arrangement of the 24 functions in this group that operate with the MCR and MRC instructions.

3-79

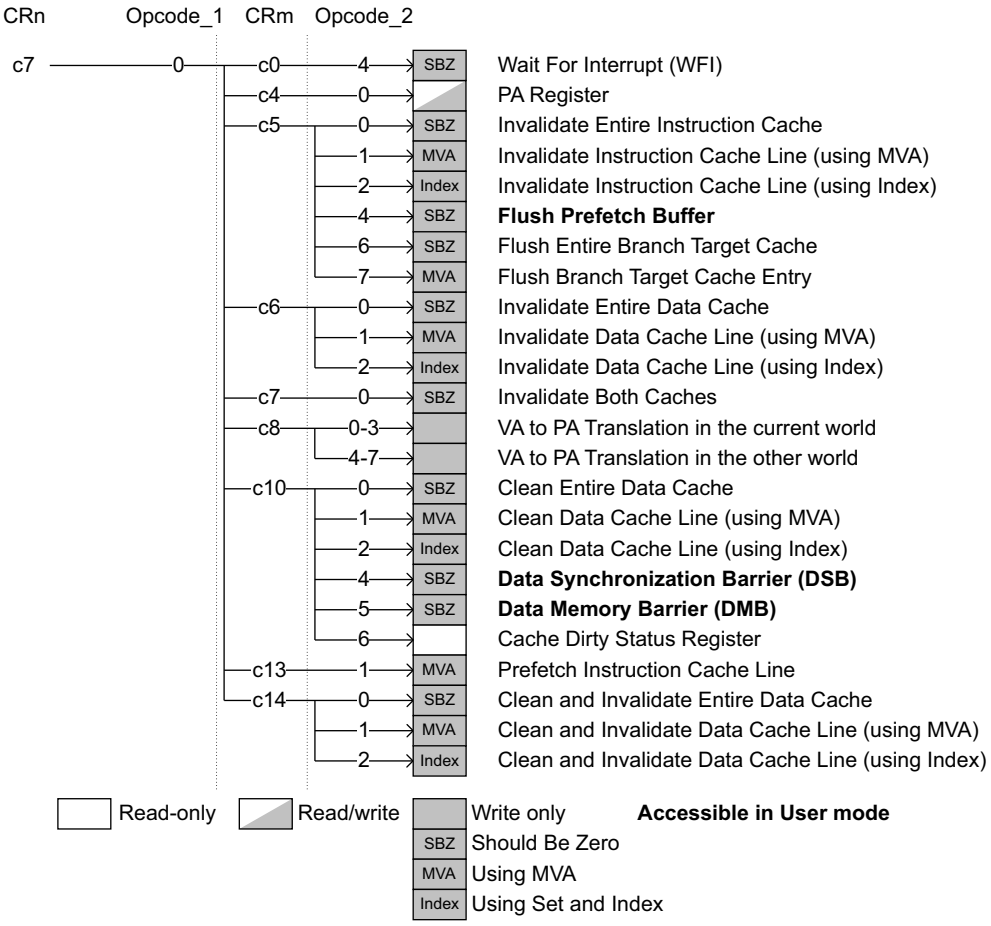

Figure 3-38 Cache operations

Figure 3-39 on page 3-71 shows the arrangement of the 4 functions in this group that operate with the MCRR instruction.

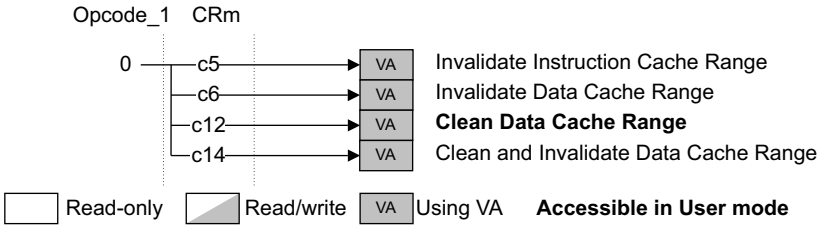

Figure 3-39 Cache operations with MCRR instructions

Note

- Writing c7 with a combination of CRm and Opcode_2 not listed in Figure 3-38 on page 3-70 or CRm not listed in Figure 3-39 results in an Undefined exception apart from the following operations, that are architecturally defined as unified cache operations and have no effect:
  - `MCR p15,0,<Rd>,c7,c7,{1-7}`
  - `MCR p15,0,<Rd>,c7,c11,{0-7}`
  - `MCR p15,0,<Rd>,c7,c15,{0-7}`.
- In the ARM1176JZF-S processor, reading from c7, except for reads from the Cache Dirty Status Register or PA Register, causes an Undefined instruction trap.
- Writes to the Cache Dirty Status Register cause an Undefined exception.
- If Opcode_1 = 0, these instructions are applied to a level one cache system. All other Opcode_1 values are reserved.
- All accesses to c7 can only be executed in a privileged mode of operation, except Data Synchronization Barrier, Flush Prefetch Buffer, Data Memory Barrier, and Clean Data Cache Range. These can be operated in User mode. Attempting to execute a privileged instruction in User mode results in the Undefined instruction trap being taken.

There are three ways to use c7:

- For the Cache Dirty Status Register, read c7 with the MRC instruction.
- For range operations use the MCRR instruction with the value of CRm to select the required operation.
- For all other operations use the MCR instruction to write to c7 with the combination of CRm and Opcode_2 to select the required operation.

  Depending on the operation you require set <Rd> for MCR instructions or <Rd> and <Rn> for MCRR instructions to:
  - *Virtual Address* (VA)
  - *Modified Virtual Address* (MVA)
  - Set and Index
  - Should Be Zero.

Invalidate, Clean, and Prefetch operations

The purposes of the invalidate, clean, and prefetch operations that c7 provides are to:

- Invalidate part or all of the Data or Instruction caches
- Clean part or all of the Data cache
- Clean and Invalidate part or all of the Data cache

- Prefetch code into the Instruction cache.

The terms used to describe the invalidate, clean, and prefetch operations are as defined in the *Caches and Write Buffers* chapter of the *ARM Architecture Reference Manual*.

For details of the behavior of c7 in the Secure and Non-secure worlds, see *TrustZone behavior* on page 3-77.

When it controls invalidate, clean, and prefetch operations c7 appears as a 32-bit write only register. There are four possible formats for the data that you write to the register that depend on the specific operation:

- Set and Index format
- MVA
- VA
- SBZ.

Set and Index format

Figure 3-40 shows the Set and Index format for invalidate and clean operations.

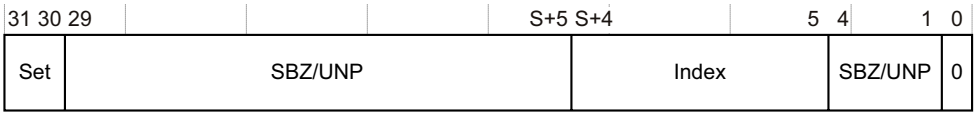

Figure 3-40 c7 format for Set and Index

Table 3-67 lists how the bit values correspond with the Cache Operation functions for Set and Index format operations.

Table 3-67 Functional bits of c7 for Set and Index

Bits	Field name	Function
[31:30]	Set	Selects the cache set to operate on, from the four cache sets. Value is the cache set number.
[29:S+5]	-	UNP/SBZ.
[S+4:5]	Index	Selects the cache line to operate on. Value is the index number.
[4:1]	-	SBZ.
[0]	0	For the ARM1176JZF-S, this Should Be Zero.

The value of S in Table 3-68 depends on the cache size. Table 3-68 lists the relationship of cache sizes and S.

Table 3-68 Cache size and S parameter dependency

Cache size	S
4KB	5
8KB	6
16KB	7
32KB	8
64KB	9

The value of S is given by:

$$S = \log_2\left(\frac{\text{cache size}}{\text{Associativity x line length in bytes}}\right)$$

See *c0, Cache Type Register* on page 3-21 for details of instruction and data cache size.

Note

If the data is stated to be Set and Index format, see Figure 3-40 on page 3-72, it identifies the cache line that the operation applies to by specifying the cache Set that it belongs to and what its Index is within the Set. The Set corresponds to the number of the cache way, and the Index number corresponds to the line number within a cache way.

MVA format

Figure 3-41 shows the MVA format for invalidate, clean, and prefetch operations.

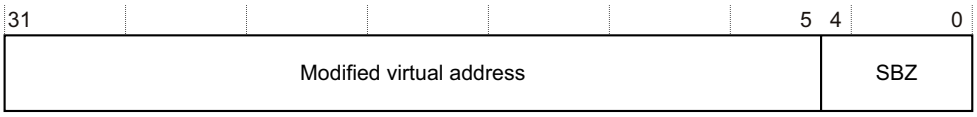

Figure 3-41 c7 format for MVA

Table 3-69 lists how the bit values correspond with the Cache Operation functions for MVA format operations.

Table 3-69 Functional bits of c7 for MVA

Bits	Field name	Function
[31:5]	MVA	Specifies address to invalidate, clean, or prefetch. Holds the MVA of the cache line.
[4:0]	-	Ignored. This means that the lower 5 bits of MVA are ignored and these bits are not used for the cache operations. Only the top bits are necessary to determine whether or not the cache line is present in the cache. Even if the MVA is not aligned to the cache line, the cache operation is performed by ignoring the lower 5 bits.

Note

- Invalidation and cleaning operations have no effect if they miss in the cache.
- If the corresponding entry is not in the TLB, these instructions can cause a TLB miss exception or hardware page table walk, depending on the miss handling mechanism.
- For the cache control operations, the MVAs that are passed to the cache are not translated by the FCSE extension.

VA format

Figure 3-42 shows the VA format for invalidate and clean operations. All VA format operations use the MCRR instruction.

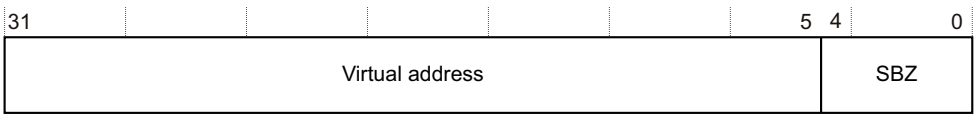

Figure 3-42 Format of c7 for VA

Table 3-70 lists how the bit values correspond with the Cache Operation functions for VA format operations.

Table 3-70 Functional bits of c7 for VA format

Bits	Field name	Function
[31:5]	Virtual address	Specifies the start or end address to invalidate or clean. Holds the true VA of the start or end of a memory block before any modification by FCSE.
[4:0]	-	SBZ.

You can perform invalidate, clean, and prefetch operations on:

- single cache lines
- entire caches
- address ranges in cache.

Note

- Clean, invalidate, and clean and invalidate operations apply regardless of the lock applied to entries.
- An explicit flush of the relevant lines in the branch target cache must be performed after invalidation of Instruction Cache lines or the results are Unpredictable. There is no impact on security. This is not required after an entire Instruction Cache invalidation because the entire branch target cache is flushed automatically.
- A small number of CP15 c7 operations can be executed by code while in User mode. Attempting to execute a privileged operation in User mode using CP15 c7 results in an Undefined instruction trap being taken.

To determine if the cache is dirty use the Cache Dirty Status Register, see *Cache Dirty Status Register* on page 3-78.

Entire cache

Table 3-71 lists the instructions and operations that you can use to clean and invalidate the entire cache.

Table 3-71 Cache operations for entire cache

Instruction	Data	Function
`MCR p15, 0, <Rd>, c7, c5, 0`	SBZ	Invalidate Entire Instruction Cache. Also flushes the branch target cache and globally flushes the BTAC.
`MCR p15, 0, <Rd>, c7, c6, 0`	SBZ	Invalidate Entire Data Cache.
`MCR p15, 0, <Rd>, c7, c7, 0`	SBZ	Invalidate Both Caches. Also flushes the branch target cache and globally flushes the BTAC.
`MCR p15, 0, <Rd>, c7, c10, 0`	SBZ	Clean Entire Data Cache.
`MCR p15, 0, <Rd>, c7, c14, 0`	SBZ	Clean and Invalidate Entire Data Cache.

Register c7 specifies operations for cleaning the entire Data Cache, and also for performing a clean and invalidate of the entire Data Cache. These are blocking operations that can be interrupted. If they are interrupted, the R14 value that is

captured on the interrupt is the address of the instruction that launched the cache clean operation + 4. This enables the standard return mechanism for interrupts to restart the operation.

If it is essential that the cache is clean, or clean and invalid, for a particular operation, the sequence of instructions for cleaning, or cleaning and invalidating, the cache for that operation must handle the arrival of an interrupt at any time when interrupts are not disabled. This is because interrupts can write to a previously clean cache. For this reason, the Cache Dirty Status Register indicates if the cache has been written to since the last clean of the cache was started, see *Cache Dirty Status Register* on page 3-78. You can interrogate the Cache Dirty Status Register to determine if the cache is clean, and if this is done while interrupts are disabled, the following operations can rely on having a clean cache. The following sequence shows this approach:

```
        ; interrupts are assumed to be enabled at this point
Loop1   MOV R1, #0
        MCR CP15, 0, R1, C7, C10, 0      ; Clean (or Clean & Invalidate) Cache
        MRS R2, CPSR
        CPSID iaf                        ; Disable interrupts
        MRC CP15, 0, R1, C7, C10, 6      ; Read Cache Dirty Status Register
        ANDS R1, R1, #1                  ; Check if it is clean
        BEQ UseClean
        MSR CPSR, R2                     ; Re-enable interrupts
        B Loop1                          ; - clean the cache again
UseClean Do_Clean_Operations             ; Perform whatever operation relies on
                                         ; the cache being clean/invalid.
                                         ; To reduce impact on interrupt
                                         ; latency, this sequence should be
                                         ; short
        MSR CPSR, R2                     ; Re-enable interrupts
```

The long cache clean operation is performed with interrupts enabled throughout this routine.

Single cache lines

There are two ways to perform invalidate or clean operations on cache lines:

- by use of Set and Index format
- by use of MVA format.

Table 3-72 lists the instructions and operations that you can use for single cache lines.

Table 3-72 Cache operations for single lines

Instruction	Data	Function
`MCR p15, 0, <Rd>, c7, c5, 1`	MVA	Invalidate Instruction Cache Line, using MVA
`MCR p15, 0, <Rd>, c7, c5, 2`	Set/Index	Invalidate Instruction Cache Line, using Index
`MCR p15, 0, <Rd>, c7, c6, 1`	MVA	Invalidate Data Cache Line, using MVA
`MCR p15, 0, <Rd>, c7, c6, 2`	Set/Index	Invalidate Data Cache Line, using Index
`MCR p15, 0, <Rd>, c7, c10, 1`	MVA	Clean Data Cache Line, using MVA
`MCR p15, 0, <Rd>, c7, c10, 2`	Set/Index	Clean Data Cache Line, using Index

Table 3-72 Cache operations for single lines (continued)

Instruction	Data	Function
MCR p15, 0, <Rd>, c7, c13, 1	MVA	Prefetch Instruction Cache Line
MCR p15, 0, <Rd>, c7, c14, 1	MVA	Clean and Invalidate Data Cache Line, using MVA
MCR p15, 0, <Rd>, c7, c14, 2	Set/Index	Clean and Invalidate Data Cache Line, using Index

Example 3-1 shows how to use Clean and Invalidate Data Cache Line with Set and Index to clean and invalidate one whole cache way, in this example, way 3. The example works with any cache size because it reads the cache size from the Cache Type Register.

Example 3-1 Clean and Invalidate Data Cache Line with Set and Index

```
                MRC     p15,0,R0,c0,c0,1        ; Read cache type reg
                AND     R0,R0,#0x1C0000         ; Extract D cache size
                MOV     R0,R0, LSR #18          ; Move to bottom bits
                ADD     R0,R0,#7                ; Get Index loop max
                MOV     r1,#3:SHL:30            ; Set up Set = 3
                MOV     R2,#0                   ; Set up Index counter
                MOV     R3,#1
                MOV     R3,R3, LSL R0           ; Set up Index loop max
index_loop
                ORR     R4,R2,r1                ; Set and Index format
                MCR     p15,0,R4,c7,c14,2       ; Clean&inval D cache line
                ADD     R2,R2,#1:SHL:5          ; Increment Index
                CMP     R2,R3                   ; Done all index values?
                BNE     index_loop              ; Loop until done
```

Address ranges

Table 3-73 lists the instructions and operations that you can use to clean and invalidate the address ranges in cache.

Table 3-73 Cache operations for address ranges

Instruction	Data	Function
MCRR p15,0,<End Address>,<Start Address>,c5	VA	Invalidate Instruction Cache Range
MCRR p15,0,<End Address>,<Start Address>,c6	VA	Invalidate Data Cache Range
MCRR p15,0,<End Address>,<Start Address>,c12	VA	Clean Data Cache Range[a]
MCRR p15,0,<End Address>,<Start Address>,c14	VA	Clean and Invalidate Data Cache Range

a. This operation is accessible in both User and privileged modes of operation. All other operations listed here are only accessible in privileged modes of operation.

The operations in Table 3-73 can only be performed using an MCRR or MCRR2 instruction, and all other operations to these registers are ignored.

The `End Address` and `Start Address` in Table 3-73 is the true VA before any modification by the *Fast Context Switch Extension* (FCSE). This address is translated by the FCSE logic. Each of the range operations operates between cache lines containing the `Start Address` and the `End Address`, inclusive of `Start Address` and `End Address`.

Because the least significant address bits are ignored, the transfer automatically adjusts to a line length multiple spanning the programmed addresses.

The `Start Address` is the first VA of the block transfer. It uses the VA bits [31:5]. The `End Address` is the VA where the block transfer stops. This address is at the start of the line containing the last address to be handled by the block transfer. It uses the VA bits [31:5].

If the Start Address is greater than the End Address the effect is architecturally Unpredictable. The ARM1176JZF-S processor does not perform cache operations in this case. All block transfers are interruptible. When Block transfers are interrupted, the R14 value that is captured is the address of the instruction that launched the block operation + 4. This enables the standard return mechanism for interrupts to restart the operation.

Exception behavior

The blocking block transfers cause a Data Abort on a translation fault if a valid page table entry cannot be fetched. The FAR indicates the address that caused the fault, and the DFSR indicates the reason for the fault.

TrustZone behavior

TrustZone affects cache operations as follows:

Secure world operations

In the Secure world cache operations can affect both Secure and Non-secure cache lines:

- Clean, invalidate, and clean and invalidate operations affect all cache lines regardless of their status as locked or unlocked.
- For clean, invalidate, and clean and invalidate operations with the Set and Index format, the selected cache line is affected regardless of the Secure tag.
- For MVA operations clean, invalidate, and clean and invalidate:
  - when the MVA is marked as Non-secure in the page table, only Non-secure entries are affected
  - when the MVA is marked as Secure in the page table, only Secure entries are affected.

Non-secure world operations

In the Non-secure world:

- Clean, invalidate, and clean and invalidate operations only affect Non-secure cache lines regardless of the method used.
- Any attempt to access Secure cache lines is ignored.
- Invalidate Entire Data Cache and Invalidate Both Caches operations cause an Undefined exception. This prevents invalidating lockdown entries that might be configured as Secure.
  - the Invalidate Both Caches operation globally flushes the BTAC.
- Invalidate Entire Instruction Cache operations:
  - cause an Undefined exception if lockdown entries are reserved for the Secure world
  - affect all Secure and Non-secure cache entries if the lockdown entries are not reserved for the Secure world
  - globally flush the BTAC.

Cache Dirty Status Register

The purpose of the Cache Dirty Status Register is to indicate when the Cache is dirty.

The Cache Dirty Status Register is:
- in CP15 c7
- a 32-bit read only register, banked for Secure and Non-secure worlds
- accessible in privileged modes only.

Figure 3-43 shows the arrangement of bits in the Cache Dirty Status Register.

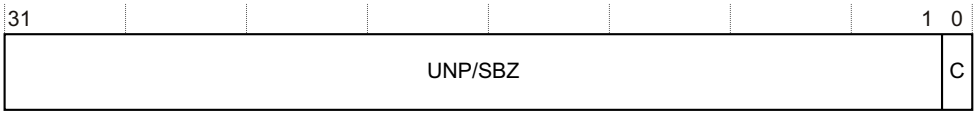

Figure 3-43 Cache Dirty Status Register format

Table 3-74 lists how the bit value corresponds with the Cache Dirty Status Register function.

Table 3-74 Cache Dirty Status Register bit functions

Bits	Field name	Function
[31:1]	-	UNP/SBZ.
[0]	C	The C bit indicates if the cache is dirty. 0 = indicates that no write has hit the cache since the last cache clean, clean and invalidate, or invalidate all operation, or reset, successfully left the cache clean. This is the reset value. 1 = indicates that the cache might contain dirty data.

The Cache Dirty Status Register behaves in this way with regard to the Secure and Non-secure cache:

- clean, invalidate, and clean and invalidate operations of the whole cache in the Non-secure world clear the Non-secure Cache Dirty Status Register
- clear, invalidate, and clean and invalidate operations of the whole cache in the Secure world clear both the Secure and Non-secure Cache Dirty Status Registers
- if the core is in the Non-secure world or targets Non-secure data from the Secure world, stores that write a dirty bit in the cache set both the Secure and the Non-secure Cache Dirty Status Register
- all stores that write a dirty bit in the cache set the Secure Cache Dirty Status Register.

All writes and User mode reads of the Cache Dirty Status Register cause an Undefined exception.

To use the Cache Dirty Status Register read CP15 with:
- Opcode_1 set to 0
- CRn set to c7
- CRm set to c10
- Opcode_2 set to 6.

For example:

```
MRC p15, 0, <Rd>, c7, c10, 6    ; Read Cache Dirty Status Register.
```

Flush operations

Table 3-75 lists the flush operations and instructions available through c7.

Table 3-75 Cache operations flush functions

Instruction	Data	Function
`MCR p15, 0, <Rd>, c7, c5, 4`	SBZ	Flush Prefetch Buffer[a].
`MCR p15, 0, <Rd>, c7, c5, 6`	SBZ	Flush Entire Branch Target Cache[b].
`MCR p15, 0, <Rd>, c7, c5, 7`	MVA[c]	Flush Branch Target Cache Entry with MVA.

a. These operations are accessible in both User and privileged modes of operation. All other operations are only accessible in privileged modes of operation.
b. This operation is accessible in both Privileged and User modes of operation when in Debug state.
c. The range of MVA bits used in this function is different to the range of bits used in other functions that have MVA data.

must do after changing cp14 register (and other co-processor regs too)

The Flush Branch Target Entry using MVA operation uses a different MVA format to that used by Clean and Invalidate operations. Figure 3-44 shows the MVA format for the Flush Branch Target Entry operation.

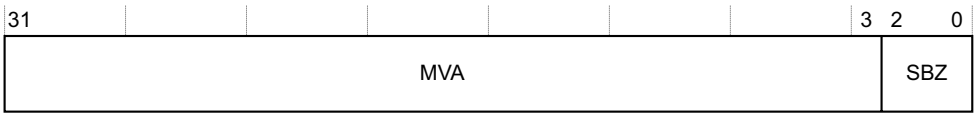

Figure 3-44 c7 format for Flush Branch Target Entry using MVA

Table 3-76 lists how the bit values correspond with the Flush Branch Target Entry using MVA functions.

Table 3-76 Flush Branch Target Entry using MVA bit functions

Bits	Field name	Function
[31:3]	MVA	Specifies address to flush. Holds the MVA of the Branch Target Cache line.
[2:0]	-	SBZ.

Note

The MVA does not have to be cache line aligned.

Flushing the prefetch buffer has the effect that all instructions occurring in program order after this instruction are fetched from the memory system after the execution of this instruction, including the level one cache or TCM. This operation is useful for ensuring the correct execution of self-modifying code. See *Explicit Memory Barriers* on page 6-25.

VA to PA translation operations

The purpose of the VA to PA translation operations is to provide a Secure means to determine address translation in the Secure and Non-secure worlds and for address translation between the Secure and Non-secure worlds. VA to PA translations operate through:

- *PA Register* on page 3-80