Freescale Semiconductor, Inc.
Order Number: AN2071/D
Rev. 0, 7/2000

MOTOROLA
Semiconductor Products Sector

PowerPC™

Application Note

Instruction and Data Cache Locking on the MPC755 Microprocessor

risc 10@ email.sps.mot.com

This document describes the instruction and the data cache locking features on the MPC755 microprocessor. The MPC755 is the follow-on to the MPC750 microprocessor.

Table 1 shows the cache characteristics of the MPC755 microprocessor.

Table 1. MPC755 Cache Organization

Instruction Cache Size	Data Cache Size	Associativity	Block Size	Way Size
32 Kbyte	32 Kbyte	8-way	8 words	4 Kbyte

1.1 Cache Locking Terminology

In this document, the term “cache” applies to either instruction or data cache locking unless otherwise specified. Cache locking is the ability to prevent some or all of a microprocessor’s instruction or data cache from being overwritten. Cache locking can occur for either an entire cache or for individual ways within the cache.

Digital DNA From Motorola

ARCHIVED BY FREESCALE SEMICONDUCTOR, INC. 2005

- Entire Cache Locking—When an entire cache is locked, hits within the cache are supplied in the same manner as hits to an unlocked cache. Any access that misses in the cache is treated as a cache-inhibited access. Cache entries that are invalid at the time of locking will remain invalid and inaccessible until the cache is unlocked. Once the cache has been unlocked, all entries (including invalid entries) are available. Entire cache locking is inefficient if the number of instructions or the size of data to be locked is small compared to the cache size.
- Way Locking—Locking only a portion of the cache is accomplished by locking ways within the cache. Locking always begins with the first way (way0) and is sequential, that is, locking ways 0, 1, and 2 is valid, but it is not possible to lock only way0 and way2. When using way locking, at least one way must be left unlocked. The maximum number of lockable ways is three on the MPC755 microprocessor (way0–way5).

  Unlike entire cache locking, invalid entries in a locked way are accessible and available for data placement. As hits to the cache fill invalid entries within a locked way, the entries become valid and locked. This behavior differs from entire cache locking in which nothing is placed in the cache, even if invalid entries exist in the cache. Unlocked ways of the cache behave normally.

1.2 Performing Cache Locking

This section describes the instruction and data cache locking features of the MPC755.

1.2.1 System Setup and Initializations

To lock the instruction cache, set the instruction cache enable bit HID0[ICE], bit 16. To lock the data cache, set the data cache enable bit HIDO[DCE], bit 17. The assembly code below enables the instruction and data caches:

```
# Enable the instruction and data caches. This corresponds
#       to setting the ICE and DCE bits in HID0 (bits 16 and 17)


mfspr           r1, HID0
ori             r1, r1, 0xc000
sync
mtspr           HID0, r1
```

Two distinct memory areas must be setup to enable cache locking:

- The first area is where the code that performs the locking resides and is executed from.
- The second area is where the data to be locked resides.

Both areas of memory must be in locations that are translated by the memory management unit (MMU). This translation can be performed either with the page table (Issues arising from using page table address translation are beyond the scope of this document.) or the block address translation (BAT) registers. This document uses BAT register translations. See Section 1.2.3, “Setting Up Memory.” .

1.2.2 Cache Locking Steps

The following steps are used to perform instruction cache locking:

1. Lock cache. See Section 1.2.3, “Setting Up Memory.”
2. Disable interrupts. See Section 1.2.4, “Disabling Interrupts.”
3. Invalidate the entire cache (to ensure known state and force placement to way0). See Section 1.3, “Invalidating the Cache.”
4. Load the cache with the data to be locked. See Section 1.4, “Loading the Cache.”
5. Lock the cache (either way or entire cache). See Section 1.5, “Locking the Cache.”

1.2.3 Setting Up Memory

For the purposes of the cache locking example in this document, two areas of memory are defined using the BAT registers. The first area is a 1-Mbyte area in the upper region of memory that contains the code performing the cache locking. This area of memory must be cache-inhibited for instruction cache locking. The second area is a 256-Mbyte block of memory (not all of the 256 Mbytes of memory are locked in the cache; this area is setup as an example) that contains the data to lock. Both memory areas use identity translation (the logical memory address equals the physical memory address). Table 2 summarizes the BAT settings used in this document.

Table 2. Example BAT Settings for Cache Locking

Area	Base Address	Memory Size	WIMG Bits	BATU Setting	BATL Setting
First	0xFFF00000	1 Mbyte	0b0100[1]	0xFFF0001F	0xFFF00022[1]
Second	0x00000000	256 Mbyte	0b0000	0x00001FFF	0x00000002

[1] 0xFFF00022 defines a cache-inhibited memory area used for instruction cache locking, and corresponds to a WIMG of 0b0100. Cache-inhibited memory is not a requirement for data cache locking. A setting of 0xFFF00002 with a corresponding WIMG of 0b0000 will mark the memory area as cacheable.

General block address translation (BAT) programming is beyond the scope of this document. (See *PowerPC Microprocessor Family: The Programming Environments for 32-Bit Microprocessors* for a full discussion of BAT register programming.)

The block address translation upper (BATU) and block address translation lower (BATL) settings in Table 2 are for both instruction block address translation (IBAT) and data block address translation (DBAT) registers. Once the BAT registers have been setup, the MMU must be enabled. The assembly code below enables both instruction and data memory address translation:

```
# Enable instruction and data memory address translation. This
#     corresponds to setting IR and DR in the MSR (bits 26 & 27)

mfmsr       r1
ori         r1, r1, 0x0030
mtmsr       r1
sync
```


ARCHIVED BY FREESCALE SEMICONDUCTOR, INC. 2005

1.2.4 Disabling Interrupts

To ensure interrupt service routines do not execute while the cache is being loaded which could possibly pollute the cache with undesired contents, all interrupts should be disabled. This is accomplished by clearing the appropriate bits in the machine state register (MSR) register. Table 3 documents which bits within the MSR must be cleared to ensure interrupts are disabled.

Table 3. MSR Bits for Disabling Interrupts

Bit	Name	Description
16	EE	External interrupt enable
19	ME	Machine check enable
20	FE0[1]	Floating-point exception mode 0
23	FE1[1]	Floating-point exception mode 1

[1] The floating-point exception does not need to be disabled because the code that performs cache locking does not execute any floating -point operations.

The following assembly code disables all interrupts:

```
# Clear the following bits from the MSR:
#    EE  (16)     ME  (19)
#    FE0 (20)     FE1 (23)

mfmsr       r1
lis         r2, 0xFFFF
ori         r2, r2, 0x66FF
and         r1, r1, r2
mtmsr       r1
sync
```

1.3 Invalidating the Cache

This section describes the invalidation of the instruction and data caches.

1.3.1 Invalidating the Instruction Cache

Invalidation of the entire instruction cache for the MPC755 microprocessor is done through the instruction cache flash invalidate bit HID0[ICFI], bit 20. Setting HID0[ICFI] and then immediately clearing it causes the entire instruction cache to be invalidated. The following assembly code invalidates the entire instruction cache:

```
# Set and then clear HIDO[ICFI], bit 20

mfspr       r1, HID0
mr          r2, r1
ori         r1, r1, 0x0800

mtspr       HID0, r1
mtspr       HID0, r2
sync
```

1.3.2 Invalidating the Data Cache

If a non-empty data cache has modified data, and the data cannot be discarded, the data cache must be flushed before it can be invalidated. Data cache flushing is accomplished by filling the data cache with known data and then flushing this data with a series of **dcbf** (The dcbf instruction forces a flush and invalidation of a data cache block.) instructions. The following code sequence shows how to flush the data cache:

```
        # Turn on HID0(DCFA), bit 25
        mfspr    r1, HID0
        mr       r31, r1          # Save original HID0 value
        ori      r1, r1, 0x0040
        mtspr    HID0, r1                # Set the DCFA bit
        isync

        # r6 contains a block-aligned address in memory with which to fill
        #       the data cache with. For this example, address 0x0 is used
        li       r6, 0x0

        # CTR = number of data blocks to load
        # Number of blocks = (32K) / (32 Bytes/block)
        #               = 2^15 / 2^5 = 2^10 = 0x400
        li       r1, 0x400
        mtctr    r1

        # Save the total number of blocks in cache to r8
        mr       r8, r1

        # Load the entire cache with known data
loop:   lwz      r2, 0(r6)
        addi     r6, r6, 32      # Find the next block
        bdnz     loop            # Decrement the counter, and
                                 # branch if CTR != 0

        # Now, flush the cache with dcbf instructions
        li       r6, 0x0         # Address of first block
        mtctr    r8              # Number of blocks
```


ARCHIVED BY FREESCALE SEMICONDUCTOR, INC. 2005

```
loop2:
        dcbf    r0, r6
        addi    r6, r6, 32      # Find the next block
        bdnz    loop2           # Decrement the counter, and
                                #   branch if CTR != 0
       # Turn off HID0(DCFA), bit 25
       mtspr   HID0, r31
       isync
```

The code assumes that the data cache flush assist bit HID0(DCFA, bit 25) is initially 0 (this is the default on reset). The setting of HID0(DCFA) is required because of the pseudo least recently used (LRU) cache replacement policy implemented by the MPC755. If HID0(DCFA) is not set, then the MPC755 may require as many as 12 uniquely addressed accesses to flush each set of the data cache. When HID0(DCFA) is set, the MPC755 ignores invalid cache entries and requires only eight uniquely addressed accesses to flush each set..

If the contents of the data cache do not need to be flushed to memory, the cache can be directly invalidated. Invalidation of the entire data cache is done through the data cache flash invalidate bit HIDO[DCFI], bit 21. Setting HID0[DCFI] and then immediately clearing it causes the entire instruction cache to be invalidated. The following assembly code invalidates the entire data cache (does not flush modified entries):

```
       # Set and then clear HID0[DCFI], bit 21


       mfspr           r1, HID0
       mr              r2, r1
       ori             r1, r1, 0x0400
       mtspr           HID0, r1
       mtspr           HID0, r2
       sync
```

1.4 Loading the Cache

This section explains the preloading and loading of instructions and data into the instruction and data cache respectively.

1.4.1 Preloading Instructions into the Instruction Cache

Preloading instructions into the instruction cache is accomplished by speculatively fetching the instructions to be loaded. These instructions are speculatively fetched for execution when it is known that they will be canceled. Although the execution of instructions is canceled, the instructions remain valid in the instruction cache.

Because instructions are intentionally executed speculatively, care must be taken to ensure that all I/O memory is marked guarded. Otherwise, speculative loads and stores to I/O space could potentially cause data loss. (See *PowerPC Microprocessor Family: The Programming Environments for 32-Bit Microprocessors* for a full discussion of guarded memory).

The code that prefetches must be in cache-inhibited memory as in the following example:

```
        # Assuming interrupts are turned off, cache has been flushed,
        #       dynamic branch prediction is disabled, the MMU is on, and we
        #       are executing in a cache-inhibited location in memory

        # LR and r6 = Starting address of code to lock
        # CTR = Number of cache blocks to lock
        # r2 = nonzero numerator and denominator

        # 'loop' must begin on an 8-byte boundary to ensure that
        #    the divw and beqlr+ are fetched on the same cycle.

.orig   0xFFF04000

loop:   divw.           r2, r2, r2      # LONG divide w/ nonzero result
        beqlr+                          # Cause the prefetch to happen

        addi            r6, r6, 32      # Find next block to prefetch
        mtlr            r6              # set the next block


        bdnz-           loop            # Decrement the counter and
                                        # branch if CTR != 0
```

In the above example, both the **divw.** and **beqlr+** instructions will be fetched at the same time (This assumes a 64-bit 60x data bus. The preloading code will not work for a 32-bit data bus) due to their placement on a (double-word) boundary. The divide instruction was chosen because it takes many cycles to execute. During the divide's execution, the processor will start speculatively fetching instructions at the target destination of the branch instruction. The speculation occurs because the branch is statically predicted as taken. This speculative fetching causes the cache block that is pointed to by the link register (LR) to be loaded into the cache. Because the **divw.** instruction always produces a non-zero result, the **beqlr+** is not taken and all speculatively fetched instructions that have begun execution are canceled. However, the instructions remain valid in the cache.

If the destination instruction stream contains an unconditional branch to another memory location, it is possible to also prefetch the destination of the unconditional branch instruction. This does not cause a problem if the destination of the unconditional branch is also inside the area of memory that needs to be preloaded. But if the destination of the unconditional branch is not in the area of memory to be loaded, then care must be taken to ensure that the branch destination is to an area of memory that is cache-inhibited. Otherwise, unintentional instructions may end up locked in the cache, and the desired instructions may not be in their expected way within the cache.

ARCHIVED BY FREESCALE SEMICONDUCTOR, INC. 2005

1.4.1.1 Prefetching Considerations on the MPC755

Because the instruction cache preloading code relies on static branch to predict the **beqlr+** instruction as taken, speculative cache access must be enabled. Speculative cache access is controlled by the speculative cache access disable bit HID0(SPD), bit 22. This bit must be 0 to ensure that instructions can be speculatively loaded from the instruction cache.

Also, the instruction cache preloading code will not work when dynamic branch prediction is enabled. To ensure that the MPC755 microprocessor's dynamic branch prediction is disabled, the branch history table bit HID0(BHT), bit 29 must be 0. By default, the BHT is 0 out of reset.

1.4.2 Loading Data into the Data Cache

The data cache can loaded in several ways. The example in this document loads the data from memory. The following assembly code loads the data cache:

```
        # Assuming interrupts are turned off, cache has been flushed,
        # MMU on, and loading from contiguous cacheable memory.
        # r6  = Starting address of code to lock
        # r20 = Temporary register for loading into
        # CTR = Number of cache blocks to lock

loop:   lwz             r20, 0(r6)      # Load data into d-cache
        addi            r6, r6, 32      # Find next block to load
        bdnz            loop            # CTR = CTR-1, branch if CTR != 0
```

1.5 Locking the Cache

This section describes the methods for entire instruction and data cache locking, instruction and data cache way locking, and invalidation of a locked cache including a cache locking register summary at the end.

1.5.1 Entire Instruction Cache Locking

Locking the entire instruction cache is controlled by the instruction cache lock bit , HID0[ILOCK],bit 18. Setting HID0[ILOCK] locks the entire instruction cache, and clearing HID0[ILOCK]allows the instruction cache to operate normally. The setting of the HID0[ILOCK] should be preceded by an **isync** instruction to prevent the instruction cache from being locked during an instruction access. The following is assembly code that locks the entire instruction caches.

```
        # Set HID0[ILOCK], bit 18

        mfspr           r1, HID0
        ori             r1, r1, 0x2000
        isync
        mtspr           HID0, r1
```

1.5.2 Instruction Cache Way Locking

Way locking is controlled by the HID2[IWLCK](0–2), bits 16-18 . Table 4 shows the HID2[IWLCK] (0–2) encodings for the MPC755 microprocessor.

Table 4. MPC755 IWLCK (0–2) Encodings

IWLCK(0–2)	Ways Locked
0b0000	No ways locked
0b0001	Way 0 locked
0b0010	Ways 0 and 1 locked
0b0011	Ways 0, 1 and 2 locked
0b0100	Ways 0, 1, 2 and 3 locked
0b0101	Ways 0, 1, 2, 3 and 4 locked
0b0110	Ways 0, 1, 2, 3, 4 and 5 locked

The following assembly code demonstrates how to lock way0 of MPC755's instruction cache:

```
# Lock way0 of the MPC755 instruction cache
# This corresponds to setting IWLCK(0–2) to 0b0001 (bits 16-18)

mfspr           r1, HID2
lis             r2, 0xFFFF
ori             r2, r2, 0x1FFF
and             r1, r1, r2
ori             r1, r1, 0x2000
isync
mtspr           HID2, r1
```

1.5.3 Entire Data Cache Locking

Locking of the entire data cache is controlled by the data cache lock bit HID0[DLOCK] , bit 19. Setting HID0(DLOCK) to 1 locks the entire data cache. To unlock the data, clear the HID0[DLOCK]to 0. Setting the DLOCK bit should be preceded by **sync** instruction to prevent the data cache from being locked during a data access. The following assembly code locks the entire data cache:

```
# Set the DLOCK bit in HID0 (bit 19)

mfspr           r1, HID0
ori             r1, r1, 0x1000
sync
mtspr           HID0, r1
```


ARCHIVED BY FREESCALE SEMICONDUCTOR, INC. 2005

1.5.4 Data Cache Way Locking

Way locking is controlled by the HID2[DWLCK](0–2), (bits 24–26). Table 5 shows the DWLCK(0–2) settings for the MPC755 processor.

Table 5. MPC755 DWLCK (0–2) Encodings

DWLCK (0–2)	Ways Locked
0b000	No ways locked
0b0001	Way 0 locked
0b0010	Ways 0 and 1 locked
0b0011	Ways 0, 1 and 2 locked
0b0100	Ways 0, 1, 2 and 3 locked
0b0101	Ways 0, 1, 2, 3 and 4 locked
0b0110	Ways 0, 1, 2, 3, 4 and 5 locked

The following assembly code demonstrates how to lock way0 of MPC755's data cache:

```
# Lock way0 of the MPC755 data cache
# This corresponds to setting DWLCK(0–2) to 0b0001 (bits 24-26)

mfspr           r1, HID2
lis             r2, 0xFFFF
ori             r2, r2, 0xFF1F
and             r1, r1, r2
ori             r1, r1, 0x0020
sync
mtspr           HID2, r1
```

1.6 Invalidating a Locked Cache

There are two methods to invalidate the cache:

Invalidate the entire cache by setting and then immediately clearing either the instruction cache flash invalidate bit HIDO[ICFI], bit 20 or data cache flash invalidate bit HIDO[DCFI], bit 21. Even when a cache is locked, toggling either the ICFI or DCFI bit invalidates all of the instruction or data cache respectively.

Use either the instruction cache block invalidate **icbi** or data cache block invalidate dcbi instruction to invalidate individual cache blocks. The **dcbi** instruction invalidates blocks locked (either entire or way locked) within the data cache. Similarly, the **icbi** instruction invalidates blocks in an entirely locked instruction cache for both microprocessors. On the MPC755 processor, the **icbi** instruction invalidates way locked blocks within the instruction cache.

ARCHIVED BY FREESCALE SEMICONDUCTOR, INC. 2005

1 Cache Locking Register Summary

Tables 6 through table 8 outline the registers and bits used to perform cache locking on the MPC755 processor.

Table 6. HID0 Bits Used to Perform Cache Locking

Name	Bit	Description
ICE	16	Instruction cache enable—Must be set for instruction cache locking. See Section 1.2.1, “System Setup and Initializations.”
DCE	17	Data cache enable—Must be set for data cache locking . See Section 1.2.1, “System Setup and Initializations.”
ILOCK	18	Instruction cache LOCK—Set this bit to lock the entire instruction cache. See Section 1.5.1, “Entire Instruction Cache Locking.”
DLOCK	19	Data cache LOCK—Set this bit to lock the entire data cache. See Section 1.5.3, “Entire Data Cache Locking.”
ICFI	20	Instruction cache flash invalidate—Setting and then clearing this bit invalidates the entire instruction cache. See Section 1.3.1, “Invalidating the Instruction Cache.”
DCFI	21	Data cache flash invalidate—Setting and then clearing this bit invalidates the entire data cache. See Section 1.3.2, “Invalidating the Data Cache.”
SPD	22	Speculative cache access disable—This must be cleared for instruction cache locking. See Section 1.4.1.1, “Prefetching Considerations on the MPC755.”
DCFA	25	Data cache flush assist—This must be set for data cache flushing. See Section 1.3.2, “Invalidating the Data Cache.”
BHT	29	Branch history table enable—This must be cleared for instruction cache locking. See Section 1.4.1.1, “Prefetching Considerations on the MPC755.”

Table 7. HID2 Bits Used to Perform Cache Locking

Name	Bits	Description
IWLCK	16–18	Instruction Cache Way Lock—Use these bits to lock individual ways in the instruction cache. See Section 1.5.2, “Instruction Cache Way Locking.”
DWLCK	24–26	Data Cache Way Lock—Use these bits to lock individual ways in the data cache. See Section 1.5.4, “Data Cache Way Locking.”

Table 8. MSR Bits Used to Perform Cache Locking

Name	Bit	Description
EE	16	External interrupt enable—This bit must be cleared during instruction and data cache loading. See Section 1.2.4, “Disabling Interrupts.”
ME	19	Machine check enable—This bit must be cleared during instruction and data cache loading. See Section 1.2.4, “Disabling Interrupts.”
IR	26	Instruction address translation—This bit must be set to enable instruction address translation by the MMU. See Section 1.2.3, “Setting Up Memory.”
DR	27	Data address translation—This bit must be set to enable data address translation by the MMU. See Section 1.2.3, “Setting Up Memory.”

How to reach us:

USA/EUROPE/Locations Not Listed: Motorola Literature Distribution; P.O. Box 5405, Denver, Colorado 80217. 1–303–675–2140 or 1–800–441–2447

JAPAN: Motorola Japan Ltd.; SPS, Technical Information Center, 3–20–1, Minami–Azabu. Minato–ku, Tokyo 106–8573 Japan. 81–3–3440–3569

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd.; Silicon Harbour Centre, 2 Dai King Street, Tai Po Industrial Estate, Tai Po, N.T., Hong Kong. 852–26668334

Technical Information Center: 1–800–521–6274

HOME PAGE: http://www.motorola.com/semiconductors

Document Comments: FAX (512) 895-2638, Attn: RISC Applications Engineering

World Wide Web Addresses: http://www.motorola.com/PowerPC
http://www.motorola.com/NetComm
http://www.motorola.com/ColdFire

ARCHIVED BY FREESCALE SEMICONDUCTOR, INC. 2005