From the stacks: Programming the cache on the PowerPC 750GX/FX

Use cache management instructions to improve performance

Chris Harrison January 04, 2005

While many programs can obtain acceptable performance by simply letting the processor manage its own caching, programs with special requirements may obtain dramatically improved performance by giving the processor explicit instructions and manipulating the cache directly. More typically, boot firmware may need to flush and enable the cache. This article, an excerpt from an Application Note in the IBM Microelectronics Technical Library, discusses the basics on how you can manipulate the caches of the PowerPC 750GX and 750FX processors.

The IBM® PowerPC® 750GX processor contains separate instruction and data caches, along with a unified L2 cache. In order to use the caches efficiently, system software must properly initialize, configure, and when appropriate, flush and disable one or more of the caches. The code samples presented here are written for the 750GX processor, but unless otherwise noted, the basic algorithms remain the same for the IBM PowerPC 750FX and 750CXe processors. Refer to the appropriate User's Manual to ensure proper implementation (see Resources).

Cache size and attributes

Both the 750GX and 750FX have two separate 32KB instruction and data caches. The two caches in each of the two processors share the following attributes:

- 8-way set associative
- Pseudo least-recently-used replacement algorithm within each set
- Physically tagged
- 32-byte cache lines (blocks)

There are two coherency state bits for each data cache block, but only one coherency state bit for each instruction cache block.

Each processor also contains a unified L2 cache. The L2 in the 750FX is implemented with an internal two-way set-associative tag memory with 4096 tags per way, and an internal 512KB SRAM for data storage. The 750GX also uses a two-way set associative tag memory with 4096 tags per way, but uses an internal 1MB SRAM for data storage. In addition to the list of features shared

with the 750FX L2 cache, the 750GX L2 cache also supports data-only and instruction-only modes (please refer to the respective chip User's Manuals listed in the Resources section for additional details).

Cache control

The 750GX's L1 caches are controlled by programming specific bits in the Hardware-Implementation-Dependent (HID0) Special Purpose Register. With these bits, the user can invalidate, disable, and lock the instruction and data caches. Access to the HID0 register is through the `mtspr` and `mfspr` instructions. To prevent accidentally overwriting other bits in the HID0 register, code should do read-modify-writes when updating the register. The HID0 cache control bits are:

- **DCFI (bit 21)** - Data Cache Flash Invalidate.
  Setting this bit will invalidate the entire data cache.
- **DCE (bit 17)** - Data Cache Enable.
  Setting this bit to 1 enables the data cache, setting to 0 disables the cache.
- **DLOCK (bit 19)** - Data Cache Lock.
  Setting this bit to 1 will lock the contents of the data cache.

The corresponding instruction cache bits in HID0 are ICFI (bit20), ICE (bit16), and ILOCK (bit 18). The following constants will be used in the code examples throughout this document.

Table 1. Cache management constants

Constant	Value
hid0	1008
hid2	1016
HID0_ICE	0x00008000
HID0_DCE	0x00004000
HID0_DCD	0xFFFFBFFF
HID0_ILOCK	0x00002000
HID0_DLOCK	0x00001000
HID0_ICFI	0x00000800
HID0_DCFI	0x00000400
HID0_BHT	0xFFFFFFFB
FBP_OFF	0xFFFFF07F
HID2_FBP	0x00000F00
HID2_FDCBP	0x00000200
HID2_PCE	0x00000007
L2_LOCK123	0x00400070
L2_DO_OFF	0xFFBFFFFF
L2_LOCK0	0x00000080

L2_ENABLE	0x80000000
L2_ULK123	0xFFFFFF8F

The HID0 register also contains a bit (Branch History Table Enable, or BHT) that controls branch prediction, and is used to control the speculative prefetch mechanism. An example of this is provided in the original Application Note.

L2 cache control is accomplished using the L2 Configuration Register, L2CR. The L2CR is discussed in detail later in this document.

Cache management instructions

The PowerPC Architecture defines instructions for controlling both the instruction and data caches, and to facilitate debugging of cache problems. The cache control instructions are listed below for convenience; please refer to the User's Manual (listed in Resources) for usage information.

Instruction	Description
dcbt	data cache block touch
dcbtst	data cache block touch for store
dcbz	data cache block zero
dcbst	data cache block store
dcbf	data cache block flush
dcbi	data cache block invalidate
icbi	instruction cache block invalidate

Enabling after reset

Both the instruction and data caches are automatically invalidated and left in a disabled state after the 750GX is powered on and during a hard reset. Assuming the caches are in a disabled state, use the following code sequence to enable the cache.

Listing 1. Flushing and enabling caches

```
!
! Code to flush and enable dcache, icache
!
 mfspr   r4,HID0 ! Read HID0
 addis   r3,r0,HID0_DCE@h               ! dcache enable bit
 ori     r3,r3,HID0_DCE@l
 or      r4,r4,r3
 addis   r3,r0,HID0_DCFI@h              ! dcache flash invalidate
 ori     r3,r3,HID0_DCFI@l
 or      r4,r4,r3
 addis   r3,r0,HID0_ICE@h               ! icache enable bit
 ori     r3,r3,HID0_ICE@l
 addis   r3,r0,HID0_ICFI@h              ! dcache flash invalidate
 ori     r3,r3,HID0_ICFI@l
 or      r4,r4,r3
 mtspr   HID0,r4 ! Write new value to HID0
```

Setting the DCFI bit invalidates the entire data cache, but does not flush any modified entries. To avoid potential coherency issues (in other words, if a non-empty data cache contains modified

data and the data cannot be discarded), the cache must be flushed before it can be invalidated. Because it is unlikely that code will know which addresses are represented in the cache, the easiest and safest way to accomplish this is by first filling the cache, ideally from unused memory, and then flushing with a series of load instructions.

Listing 2. Filling the cache

```
!
! r4  - a block-aligned address in memory used to fill the cache
!      (0x00002000 in this example)
! r5  - scratchpad reg
! CTR - the number of data blocks needed to fill the cache - save it in r3
!       32K (size of cache) / 32 (bytes per block) = 0x400
!
!
 li      r2,0x0400
 mtctr   r2
 mr      r3,r2           ! Save in r3
 li      r4,0x2000       ! start address = 0x00002000
!
! Load cache
!
loop1:
 lwz     r6,0(r4)
 addi    r4,r4,0x10      ! Move to next block
 bdnz    loop1    ! decrement ctr, branch if necessary
!
! Now flush
!
 li      r4,0x2000
 mtctr   r3
loop2:
 dcbf    r0,r4
 addi    r4,r4,0x10      ! move to next block
 bdnz    loop2    ! decrement & branch as appropriate
!
! Disable the data cache
!
 mfspr   r4,hid0
 addis   r3,r0,HID0_DCD@h
 ori     r3,r3,HID0_DCD@l
 and     r4,r4,r3
 mtspr   hid0,r4
```

Cache locking

Cache locking is a mechanism to prevent the contents of the cache from being overwritten. When the entire cache is locked, cache hits are serviced in the same manner as with an unlocked cache. Any access that misses in the cache is treated as a cache-inhibited access (in other words, the access attempt will be propagated to the L2 cache or the 60x bus). If there are any invalid entries in the cache, these are still considered unavailable to the system, and are unused. Locking the entire cache is inefficient if the size of the data to be locked is small relative to the cache size.

Way locking refers to locking only a portion of the cache. Unlike entire cache locking, a portion of the cache remains available for use by the system. Way locking of the primary instruction and data caches is not permitted on the 750GX or FX processors, but way locking of the L2 cache is possible.

For more information on locking the cache, read the original Application Note (see Resources).

The `dcbf` instruction in loop2 will place each block in the invalid state. If the contents of the cache do not need to be flushed, set the DCFI bit in the HID0 register to invalidate the entire cache (as in Listing 2). With the data cache flushed and invalidated, it can safely be enabled by setting the DCE bit in the HID0 register.

In this example, the `lwz` instruction was used to load the data cache. The `dcbt` or `dcbtst` instructions, which are used primarily to preload data into the cache, could have been used as well. However, there are considerations to be aware of. The *PowerPC Microprocessor Family: The Programming Environments Manual* (see Resources) states, "the processor is not obliged to load the addressed block into the data cache." As implemented in the 750GX, the `dcbt` and `dcbst` instructions will load data into the cache only when all of the following conditions are true:

- The address hits in the Translation Lookaside Buffer (TLB) or the BAT
- Load access is permitted from the addressed page
- The load is not directed to a direct-store segment
- The address is directed at a cacheable page

If all of these conditions are not met, the 750GX treats the two instructions as no-ops.

Parity

In the 750GX, parity is implemented for the i-cache, i-tag, d-cache, d-tag, and l2-tag arrays. For all arrays, parity for a given set of data is a 1 if there are an odd number of 1s in the data (even parity). The processor provides three parity checking enable/disable control bits in the HID2 register - one for the i-cache and i-tag arrays (ICPE, bit 29), one for the d-cache and d-tag arrays (DCPE, bit 30), and one for the l2-tag array (L2PE, bit 31). Parity is calculated and stored each time there is a write to an array, regardless of the setting of the Parity Checking control bits. The parity checking enable/disable control bits are provided to select when parity is to be checked for Reads by array group (i-cache/tag, d-cache/tag, l2-tag). Parity checking can be enabled or disabled at any time in the code stream without changing the array enable/disable state, and the arrays do not require invalidation.

If parity checking is enabled and bad parity is found on a read for the enabled array, the Machine Check Enable bit of the Machine State Register (MSR[ME]) controls the action taken by the processor for the parity error. If the ME bit is set and a parity error is detected, the processor will take a Machine Check Interrupt which allows the processor to determine the failing array. If the bit is not asserted, the processor will take a Checkstop. All parity errors are non-recoverable.

Testing cache parity

The 750GX/FX processors provide a mechanism for testing the cache parity mechanism. The HID2 register has five Force Bad Parity control bits which when set, will control the parity bits written for each write to a specific array. The HID2 register also contains three parity status bits. If a parity error is detected and Machine Check interrupts are enabled, the interrupt handler can examine the status bits to determine which cache array caused the interrupt. The HID2 register bits are:

20	FICBP	Force Instruction cache bad parity
21	FITBP	Force instruction tag bad parity
22	FDCBP	Force data cache bad parity
23	FDTBP	Force data tag bad parity
24	FL2TBP	Force L2 tag bad parity
25	ICPS	L1 instruction cache/tag parity error status
26	DCPS	L1 data cache/ tag parity error status
27	L2PS	L2 tag parity error status

Only a few lines of code are needed to generate a parity error. In the following example, the data cache will be populated with a block of data. For a portion of these cache lines, the FDCBP bit will be set, causing bad parity to be written for those entries. Subsequent reads from any of the cache lines with bad parity will invoke the Machine Check interrupt handler. Use similar code to test parity in the other cache arrays.

Listing 3. Testing parity

```
!
!
! r4 - a block-aligned address in memory used to fill the cache (0x00002000 in this example)
! r5, r6 - scratchpad regs
! CTR - the number of data blocks to load
!
 li r2,0x10  ! Load 16 lines (arbitrary for this example)
 mtctr r2
 li r4,0x2000 ! start address = 0x00002000
!
! Make sure all Force Bad Parity bits are off
!
 mfspr r5,HID2  ! read HID2
 addis r4,r0,FBP_OFF@h
 ori r4,r4,FBP_OFF@l
 and r5,r5,r4 ! Clear Force Bad Parity bits
 mtspr HID2,r5  ! Restore HID2
 isync   !
!
! Load cache
!
loop1:
 lw  r6,0(r4)
 addi r4,r4,0x20 ! Move to next block
 bdnz loop1  ! decrement ctr, branch if necessary
 mfspr r5,HID2  ! read HID2
 addis r4,r0,HID2_FDCBP@h
 ori r4,r4,HID2_FDCBP@l
 or r5,r5,r4 ! set Force Data Cache Bad Parity bit
```

```
 mtspr HID2,r5  ! Restore HID2
 isync
 mtctr r2  ! 10 more lines
loop2:
 lw  r6,0(r4)
 addi r4,r4,0x20 ! Move to next block
 bdnz loop2  ! decrement ctr, branch if necessary
 mfspr r5,HID2  ! read HID2
 addis r4,r0,FBP_OFF@h
 ori r4,r4,FBP_OFF@l
 and r5,r5,r4 ! Clear Force Data Cache Bad Parity bits,
    ! set Parity Check Enable bits
 mtspr HID2,r5  ! Restore HID0
 isync
!
! Cache is populated.
```

L2 cache

The 750GX implements the L2 cache as a 4-way set-associative tag memory with 4096 tags per way, and an internal 1MB SRAM for data storage. The 750GX L2 cache can be configured with any combination of individual ways locked, have half of the ways locked, have all of the ways locked, or have all of the ways unlocked. In addition, the 750GX can be configured for instruction-only mode (refer to the Application Note "Differences Between the PowerPC 750GX and the 750FX RISC Microprocessors" listed in Resources).

L2 cache initialization

Like the primary cache arrays, the L2 cache is initially disabled following a power-on or hard reset. Before enabling the L2 cache, configuration parameters must be set in the L2 Cache Control Register (L2CR), and the L2 tags must be globally invalidated. This is done by disabling interrupts, disabling the L2 cache by clearing the L2 Enable bit (L2CR[L2e], performing the global invalidate, setting L2CR configuration bits, and setting the L2CR[L2E] bit to 1. As with all SPRs, it is best to use read-modify-write when accessing L2CR.

Listing 4. Enabling the L2 cache

```
!
! Assuming interrupts are disabled, disable L2, perform global invalidate, enable.
!

 mfspr r3,L2CR  ! Read L2CR
 rlwinm r3,r3,0,1,31 ! Clear L2CR[L2E] (bit 0)
 mtspr L2CR,r3  ! Write L2CR
 isync
 oris r3,r3,31 ! Set bit 10, Global Invalidate
 mtspr L2CR,r3  ! Write L2CR
 isync
loop1:
 mfspr r3,L2CR  ! Read L2CR
 rlwinm r3,r3,0,31,31 ! Mask bit 31 - Invalidate in progress
 cmpi 0,r3,0  ! Still in progress?
 bc loop1  ! loop until invalidate completes
 mfspr r3,L2CR  ! Read L2CR
 oris r3,r3,32768 ! Set enable bit
 mtspr L2CR,r3
 isync
```

Conclusion

By understanding the implementation of the caches in the 750 family of processors and the specialized cache management instructions available, system software has tremendous flexibility in configuring and using the caches to optimize performance. The code samples presented in this Application Note were developed on and for an IBM PowerPC 750GX processor, and are intended to illustrate the basic features of the cache and the steps required to use them. Additional information is available in the User's Manual and in the *IBM PowerPC Microprocessor Family: The Programming Environments Manual* (see Resources).

The original Application Note this article is excerpted from gives examples of locking the primary instruction and data caches, locking the L2 cache, and examining and manipulating both L1 and L2 cache with RiscWatch.