Order Number: MPC755UM/D
Rev. 0.1, 10/2000

MOTOROLA
Semiconductor Products Sector

PowerPC™

Addendum to **MPC750 RISC Microprocessor User's Manual:** MPC755 Embedded G3 Microprocessor Supplement

The MPC755 is a derivative of the MPC750 microprocessor design and is intended primarily for use in embedded systems. All of the information in the *MPC750 RISC Microprocessor User's Manual* applies to the MPC755 microprocessor with the exceptions and additions noted in this document. In the event the two documents conflict with each other, this document supersedes the information in the *MPC750 RISC Microprocessor User's Manual*.

The MPC745 is a lower-pin-count device that operates identically to the MPC755, except that it doesn't implement the L2 cache interface. In the same way that the *MPC750 User's Manual* also describes the functionality of the MPC740, this document describes the functionality of the MPC745. All information herein applies to the MPC745, except where otherwise noted (in particular, the L2 cache information does not apply to the MPC745).

This document describes specific details about the implementation of the MPC755 as a low-power, 32-bit member of the PowerPC™ processor family, and how it differs from the MPC750. Note that the individual section headings indicate the chapters in the *MPC750 User's Manual* to which they correspond. The sections are as follows:

- Part I, "MPC755 Overview," describes general features of the MPC755 with respect to the PowerPC architecture.
- Part II, "The MPC755 Programming Model (Chapter 2)," describes the differences between the programming model of the MPC750 and MPC755.

DigitalDNA From Motorola

- Part III, “MPC755 L1 Instruction and Data Cache Operation (Chapter 3),” describes the aspects of the L1 instruction and data cache operation that are specific to the MPC755.
- Part IV, “MPC755 Exceptions (Chapter 4),” describes how the MPC755 embedded processor implements the exception model defined by the operating environment architecture (OEA) for PowerPC processors.
- Part V, “MPC755 Memory Management (Chapter 5),” describes the MPC755 embedded processor’s implementation of the memory management unit (MMU) specifications provided by the operating environment architecture (OEA) for PowerPC processors.
- Part VI, “MPC755 Instruction Timing (Chapter 6),” describes how the MPC755 embedded processor fetches, dispatches, and executes instructions and how it reports the results of instruction execution.
- Part VII, “MPC755 Signal Descriptions (Chapter 7),” describes the MPC755 embedded processor’s external signals.
- Part VIII, “MPC755 System Interface Operation (Chapter 8),” describes the MPC755 embedded processor bus interface and its operation.
- Part IX, “MPC755 L2 Cache Interface Operation (Chapter 9),” describes the L2 cache interface and the private memory features of the MPC755.
- Part X, “Power and Thermal Management (Chapter 10),” describes the hardware support provided by the MPC755 for power and thermal management.
- Part XI, “Performance Monitor (Chapter 11),” describes the performance monitor of the MPC755.

Part I MPC755 Overview

This section is an overview of the MPC755. The following list of functional additions to the MPC755 from the MPC750 summarizes the changes visible either to a programmer or a system designer.

- Instruction and data cache locking mechanism added
- Four IBAT and four DBAT entries added
- Software table search mode added
- Four special-purpose (SPRG) registers added
- Parity generation and detection on L2 address bus added
- Instruction-only mode to L2 cache added
- Private SRAM capability to L2 cache interface added
- PB3-type SRAM support to L2 cache interface added
- 32-bit data bus mode added
- Bus voltage select (BVSEL) and L2 cache interface voltage select (L2VSEL) added

1.1 MPC755 Functional Description

This section summarizes some of the functional differences between the MPC750 and the MPC755. For information about the MPC755 L1 cache, see Part III, "MPC755 L1 Instruction and Data Cache Operation (Chapter 3)."

The MPC755 has independent on-chip, 32-Kbyte, eight-way set-associative, physically addressed caches for instructions and data and independent instruction and data memory management units (MMUs). Each MMU has a 128-entry, two-way set-associative translation lookaside buffer (DTLB and ITLB) that saves recently used page address translations. Block address translation on the MPC755 is performed by either two four-entry or two eight-entry BAT arrays—one for instruction and one for data block address translation (IBAT and DBAT arrays). Note that the IBAT and DBAT arrays defined by the PowerPC architecture only contain four entries each. During block translation, effective addresses are compared simultaneously with all enabled BAT entries. The MPC755 also optionally supports software table search operations.

The L2 cache is implemented with an on-chip, two-way set-associative tag memory, and with external, synchronous SRAMs for data storage. The external SRAMs are accessed through a dedicated L2 cache port that supports a single bank of up to 1 Mbyte of synchronous SRAMs. For information about the L2 cache implementation, see Part IX, "MPC755 L2 Cache Interface Operation (Chapter 9)."

The MPC755 has a 32-bit address bus and a 32/64-bit data bus. Multiple devices compete for system resources through a central external arbiter. The MPC755's three-state cache-coherency protocol (MEI) supports the exclusive, modified, and invalid states, a compatible subset of the modified/exclusive/shared/invalid (MESI) four-state protocol, and it operates coherently in systems with four-state caches. The MPC755 supports single-beat and burst data transfers for memory accesses and memory-mapped I/O operations. The system interface is described in Part VII, "MPC755 Signal Descriptions (Chapter 7)," and Part VIII, "MPC755 System Interface Operation (Chapter 8)."

The MPC755 has four software-controllable power-saving modes. Three static modes (doze, nap, and sleep) progressively reduce power dissipation. When functional units are idle, a dynamic power management mode causes those units to enter a low-power mode automatically without affecting operational performance, software execution, or external hardware. The MPC755 also provides a thermal assist unit (TAU) and a way to reduce the instruction fetch rate for limiting power dissipation. Power management is described in Part X, "Power and Thermal Management (Chapter 10)."

Figure 1 shows the block diagram of the MPC755 and the parallel organization of the execution units (shaded in the diagram). The instruction unit fetches, dispatches, and predicts branch instructions. Note that this is a conceptual model that shows basic features rather than attempting to show how features are implemented physically.

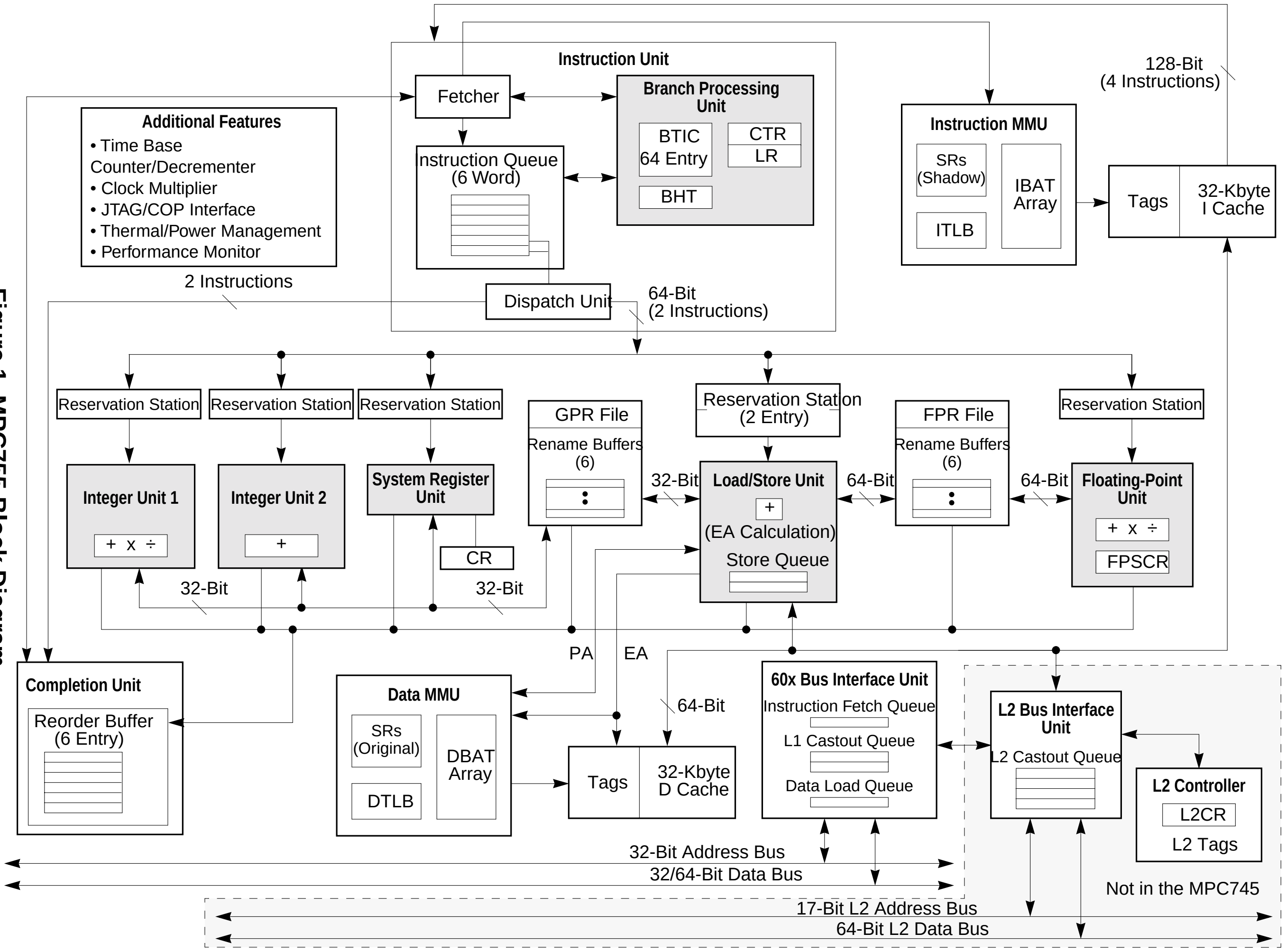


Figure 1. MPC755 Block Diagram

1.2 MPC755 Features

This section lists the features of the MPC755. The interrelationship of these features is shown in Figure 1. The major features of the MPC755 are as follows:

- High-performance, superscalar microprocessor
  - As many as four instructions can be fetched from the instruction cache per clock cycle
  - As many as two instructions can be dispatched per clock
  - As many as six instructions can execute per clock (including two integer instructions)
  - Single-clock-cycle execution for most instructions
- Six independent execution units and two register files
  - BPU featuring both static and dynamic branch prediction
    - 64-entry (16-set, four-way set-associative) branch target instruction cache (BTIC), a cache of branch instructions that have been encountered in branch/loop code sequences. If a target instruction is in the BTIC, it is fetched into the instruction queue a cycle sooner than it can be made available from the instruction cache. Typically, if a fetch access hits the BTIC, it provides the first two instructions in the target stream.
    - 512-entry branch history table (BHT) with two bits per entry for four levels of prediction—not-taken, strongly not-taken, taken, strongly taken
    - Branch instructions that do not update the count register (CTR) or link register (LR) are removed from the instruction stream
  - Two integer units (IUs) that share thirty-two 32-bit GPRs for integer operands
    - IU1 can execute any integer instruction
    - IU2 can execute all integer instructions except multiply and divide instructions (shift, rotate, arithmetic, and logical instructions). Most instructions that execute in the IU2 take one cycle to execute. The IU2 has a single-entry reservation station.
  - Three-stage floating-point unit (FPU)
    - Fully IEEE 754-1985-compliant FPU for both single- and double-precision operations
    - Supports non-IEEE mode for time-critical operations
    - Hardware support for denormalized numbers
    - Single-entry reservation station
    - Thirty-two 64-bit FPRs for single- or double-precision operands

- Two-stage load/store unit (LSU)
  - Two-entry reservation station
  - Single-cycle, pipelined cache access
  - Dedicated adder performs EA calculations
  - Performs alignment and precision conversion for floating-point data
  - Performs alignment and sign extension for integer data
  - Three-entry store queue
  - Supports both big- and little-endian modes
- System register unit (SRU) handles miscellaneous instructions
  - Executes CR logical and Move to/Move from SPR instructions (**mtspr** and **mfspr**)
  - Single-entry reservation station

• Rename buffers
  - Six GPR rename buffers
  - Six FPR rename buffers
  - Condition register buffering supports two CR writes per clock

• Completion unit
  - The completion unit retires an instruction from the six-entry reorder buffer (completion queue) when all instructions ahead of it have been completed, the instruction has finished execution, and no exceptions are pending.
  - Guarantees sequential programming model (precise exception model)
  - Monitors all dispatched instructions and retires them in order
  - Tracks unresolved branches and flushes instructions from the mispredicted branch
  - Retires as many as two instructions per clock

• Separate on-chip instruction and data caches (Harvard architecture)
  - 32-Kbyte, eight-way set-associative instruction and data caches
  - Pseudo least-recently-used (PLRU) replacement algorithm
  - 32-byte (eight-word) cache block
  - Physically indexed/physical tags.
  - Cache write-back or write-through operation programmable on a per-page or per-block basis
  - Instruction cache can provide four instructions per clock; data cache can provide two words per clock
  - Caches can be disabled in software
  - Caches can be locked 6 of 8 ways or the entire cache can be locked in software
  - Data cache coherency (MEI) maintained in hardware

  - The critical double word is made available to the requesting unit when it is burst into the line-fill buffer. The cache is nonblocking, so it can be accessed during this operation.
- Level 2 (L2) cache interface (The L2 cache interface is not supported in the MPC745.)
  - On-chip two-way set-associative L2 cache controller and tags
  - External data SRAMs
  - Support for 256-Kbyte, 512-Kbyte, and 1-Mbyte L2 caches
  - 64-byte (256-Kbyte/512-Kbyte) and 128-byte (1 Mbyte) sectored line size
  - Supports flow-through (register-buffer), both PB2 and PB3 pipelined (register-register), and pipelined late-write (register-register) synchronous burst SRAMs
- Separate memory management units (MMUs) for instructions and data
  - 52-bit virtual address; 32-bit physical address
  - Address translation for 4-Kbyte pages, variable-sized blocks, and 256-Mbyte segments
  - Memory programmable as write-back/write-through, cacheable/noncacheable, and coherency enforced/coherency not enforced on a page or block basis
  - Separate IBATs and DBATs (selectable four or eight each) also defined as SPRs
  - Separate instruction and data translation lookaside buffers (TLBs)
    - Both TLBs are 128-entry, two-way set-associative, and use PLRU replacement algorithm
  - TLBs are reloaded by the hardware or optionally, by software
- Separate bus interface units for system memory and for the L2 cache
  - Bus interface features include the following:
    - Selectable bus-to-core clock frequency ratios as described in the *MPC755 Hardware Specification*
    - 32/64-bit, split-transaction external data bus with burst transfers with 32-bit mode selectable at reset
    - Support for address pipelining and limited out-of-order bus transactions
    - Single-entry load queue
    - Single-entry instruction fetch queue
    - Two-entry L1 cache castout queue
    - No-$\overline{\text{DRTRY}}$ mode eliminates the $\overline{\text{DRTRY}}$ signal from the qualified bus grant. This allows the forwarding of data during load operations to the internal core one bus cycle sooner than if the use of $\overline{\text{DRTRY}}$ is enabled.

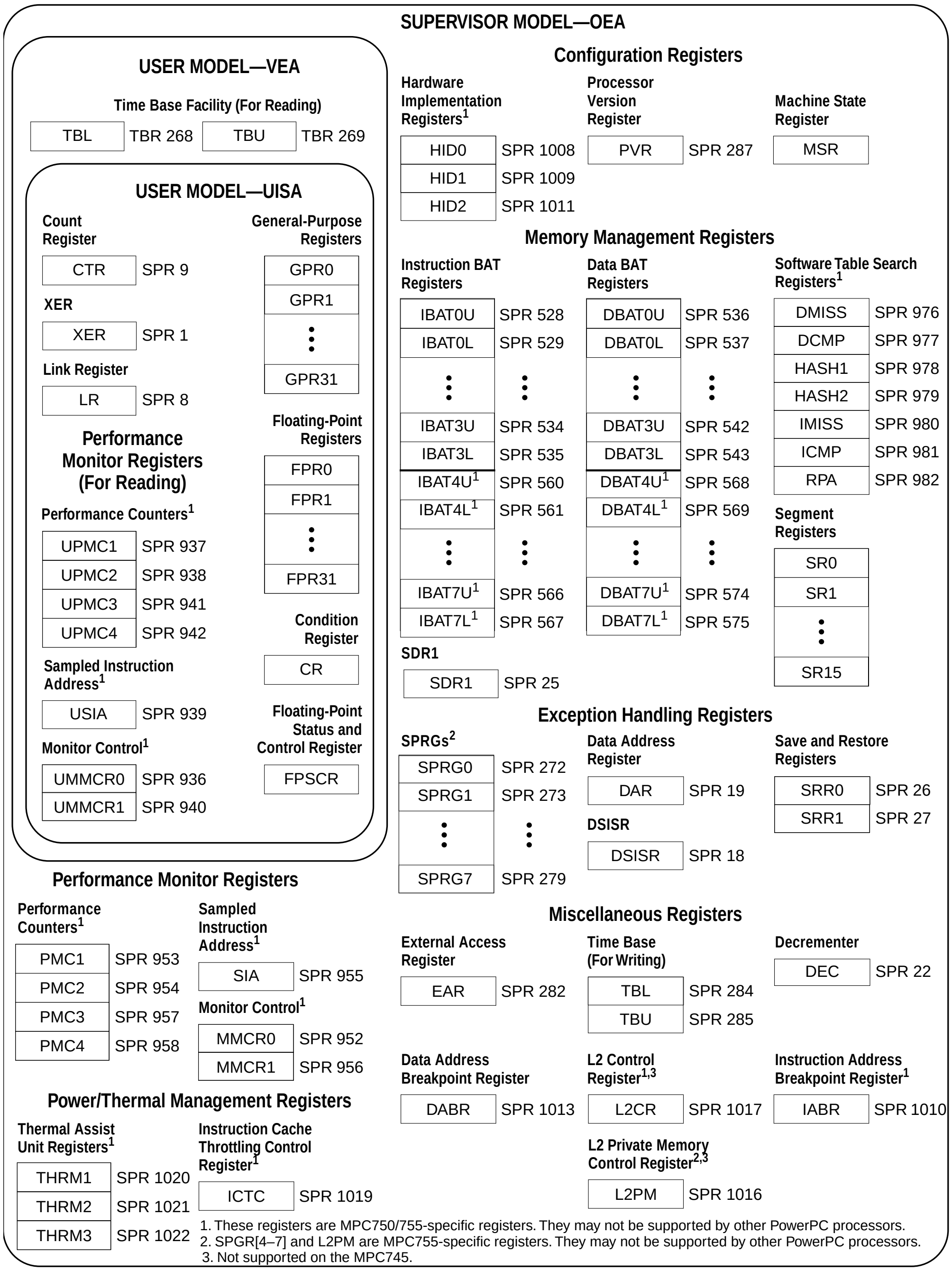
  - — L2 cache interface features (which are not implemented on the MPC745) include the following:
    - – Core-to-L2 frequency divisors as described in the *MPC755 Hardware Specification*
    - – Four-entry L2 cache castout queue in L2 cache BIU
    - – 17-bit address bus
    - – 64-bit data bus
    - – 8-bit parity for address and data
    - – Private memory mode, allowing software to access L2 SRAM as private memory space
- Multiprocessing support features include the following:
  - — Hardware-enforced, three-state cache coherency protocol (MEI) for data cache
  - — Load/store with reservation instruction pair for atomic memory references, semaphores, and other multiprocessor operations
- Power and thermal management
  - — Three static modes (doze, nap, and sleep) progressively reduce power dissipation:
    - – Doze—All the functional units are disabled except for the time base/decrementer registers and the bus snooping logic.
    - – Nap—The nap mode further reduces power consumption by disabling all functional units, disabling snooping, and leaving only the time base register and the PLL in a powered state. If snooping is required, the $\overline{\text{QACK}}$ input signal can be negated to wake up the processor and snooping logic.
    - – Sleep—All internal functional units are disabled, after which external system logic may disable the PLL and SYSCLK.
  - — Thermal management facility provides software-controllable thermal management. Thermal management is performed through the use of three supervisor-level registers and an MPC755-specific thermal management exception.
  - — Instruction cache throttling provides control of instruction fetching to limit power consumption.
- Performance monitor can be used to help debug system designs and improve software efficiency.
- In-system testability and debugging features through JTAG boundary-scan capability

Part II The MPC755 Programming Model (Chapter 2)

This section describes the differences between the programming model of the MPC750 and MPC755. For detailed information about architecture-defined features, see *PowerPC RISC Microprocessor Family: The Programming Environments Manual*. This section is organized as follows:

- Section 2.1, “MPC755-Specific Registers,”
- Section 2.2, “MPC750 and MPC755 Instruction Use,” and
- Section 2.3, “tlbld and tlbli Instructions.”

Figure 2 shows the registers implemented in the MPC755, indicating those that are defined by the PowerPC architecture and those that are MPC755-specific.



1. These registers are MPC750/755-specific registers. They may not be supported by other PowerPC processors.
2. SPGR[4–7] and L2PM are MPC755-specific registers. They may not be supported by other PowerPC processors.
3. Not supported on the MPC745.

Figure 2. Programming Model—MPC755 Microprocessor Registers

2.1 MPC755-Specific Registers

The MPC755 processor programming model is functionally identical to that of the MPC750 except for some differences in the PVR (described in Section 2.1.2, “Processor Version Register (PVR)”) and the L2CR (described in Section 9.4.1, “L2 Cache Control Register (L2CR)”). Additionally, the following special-purpose registers are added in the MPC755 that are not defined by the PowerPC architecture:

- Special-purpose registers used for general purpose (SPRG[4–7])—Four additional SPRG registers have been implemented to assist in searching the page tables in software. This is a replacement for having the MSR[TGPR] bit of the MPC603e and four temporary general purpose registers. Note that the MSR[TGPR] bit is not implemented in the MPC755. If software table searching is not enabled, then these registers may be used for any supervisor purpose. The format of these registers is the same as that of SPRG[0–3] defined in the *MPC750 User’s Manual*.
- Hardware implementation-dependent register 2 (HID2)—This register, which is not implemented in the MPC750, is used to enable L2 address parity, software table search operations, IBAT[4–7] and DBAT[4–7], and instruction and data cache way locking. This register is described in Section 2.1.3, “Hardware Implementation-Dependent Register 2 (HID2).”
- Instruction and data block address translation entries (IBAT[4–7] and DBAT[4–7]) which are optionally enabled in HID2—BATs are software-controlled arrays that store the available block address translations on-chip. BAT array entries are implemented as pairs of BAT registers that are accessible as supervisor special-purpose registers (SPRs). Four additional IBATs and four additional DBATs array entries provide a mechanism for translating additional blocks as large as 256 Mbytes from the 32-bit effective address space into the physical memory space. This can be used for translating large address ranges whose mappings do not change frequently. The format of these registers is the same as that of IBAT[0–3] and DBAT[0–3] defined in the *MPC750 User’s Manual*. The SPR numbers for accessing these registers are outlined in Table 1.
- The software table search registers are as follows (see Part V, “MPC755 Memory Management (Chapter 5),” for more detailed information):
  - Data and instruction TLB miss registers (DMISS and IMISS)—The DMISS and IMISS registers contain the effective page address of the access that caused the TLB miss exception. The contents are used by the MPC755 when calculating the values of HASH1 and HASH2, and by the **tlbld** and **tlbli** instructions when loading a new TLB entry.
  - Data and instruction TLB compare registers (DCMP and ICMP)—These registers contain the first word in the required page table entry (PTE). The contents are constructed automatically from the contents of the segment registers and the effective address (DMISS or IMISS) when a TLB miss exception occurs. Each PTE read from the tables during the table search process should be compared with this value to determine whether or not the PTE is a match. Upon execution of a **tlbld** or **tlbli** instruction the upper 25 bits of the DCMP or ICMP register and 11 bits of the effective address operand are loaded into the first word of the selected TLB entry.

- — Primary and secondary hash address registers (HASH1 and HASH2)—These registers contain the physical addresses of the primary and secondary page table entry groups (PTEGs) for the access that caused the TLB miss exception. For convenience, the MPC755 automatically constructs the full physical address by routing bits 0–6 of SDR1 into HASH1 and HASH2 and clearing the lower 6 bits. These registers are read-only and are constructed from the contents of the DMISS or IMISS register (the register choice is determined by which miss was last acknowledged).
- — Required physical address register (RPA)—During a page table search operation, the software must load the RPA with the second word of the correct PTE. When the **tlbld** or **tlbli** instruction is executed, the contents of the RPA register and the DMISS or IMISS register are merged and loaded into the selected TLB entry. The referenced (R) bit is ignored when the write occurs (no location exists in the TLB entry for this bit). The RPA register is read and write to the software.

• L2 private memory control register (L2PM)—The L2 cache private memory control register allows a portion of the physical address space to be directly mapped into a portion of the L2 SRAM. It is a supervisor-only, read/write, implementation-specific special purpose register (SPR) which is accessed as SPR 1016 (decimal). The L2PM is initialized to all 0s during power-on reset and is described more completely in Section 9.4.2, “L2 Private Memory Control Register (L2PM).”

2.1.1 The MPC755 Additional SPR Encodings

Table 1 describes the encodings of the MPC755’s register set additions described in this section.

Table 1. Additional SPR Encodings

SPR			Register	Access
Decimal	SPR[5–9]	SPR[0–4]		
276	01000	10100	SPRG4	Supervisor
277	01000	10101	SPRG5	Supervisor
278	01000	10110	SPRG6	Supervisor
279	01000	10111	SPRG7	Supervisor
560	10001	10000	IBAT4U	Supervisor
561	10001	10001	IBAT4L	Supervisor
562	10001	10010	IBAT5U	Supervisor
563	10001	10011	IBAT5L	Supervisor
564	10001	10100	IBAT6U	Supervisor
565	10001	10101	IBAT6L	Supervisor
566	10001	10110	IBAT7U	Supervisor
567	10001	10111	IBAT7L	Supervisor

Table 1. Additional SPR Encodings (Continued)

SPR			Register	Access
Decimal	SPR[5–9]	SPR[0–4]		
568	10001	11000	DBAT4U	Supervisor
569	10001	11001	DBAT4L	Supervisor
570	10001	11010	DBAT5U	Supervisor
571	10001	11011	DBAT5L	Supervisor
572	10001	11100	DBAT6U	Supervisor
573	10001	11101	DBAT6L	Supervisor
574	10001	11110	DBAT7U	Supervisor
575	10001	11111	DBAT7L	Supervisor
976	11110	10000	DMISS	Supervisor
977	11110	10001	DCMP	Supervisor
978	11110	10010	HASH1	Supervisor
979	11110	10011	HASH2	Supervisor
980	11110	10100	IMISS	Supervisor
981	11110	10101	ICMP	Supervisor
982	11110	10110	RPA	Supervisor
1011	11111	10011	HID2	Supervisor
1016	11111	11000	L2PM	Supervisor

2.1.2 Processor Version Register (PVR)

The processor version register (PVR) is a 32-bit, read-only register present in the MPC750 but initialized to a different value. It contains a value identifying the specific version (model) and revision level of the PowerPC processor (see Table 3). The contents of the PVR can be copied to a GPR by the **mfspr** instruction. Read access to the PVR is supervisor-level only; write access is not provided.

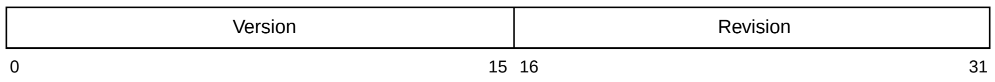


Figure 3. Processor Version Register (PVR)

The PVR consists of two 16-bit fields:

- Version (bits 0–15)—A 16-bit number that uniquely identifies a particular processor version. This number can be used to determine the version of a processor; it may not distinguish between different end product models if more than one model uses the same processor.

- Revision (bits 16–31)—A 16-bit number that distinguishes between various releases of a particular version (that is, an engineering change level). The value of the revision portion of the PVR is implementation-specific. The processor revision level is changed for each revision of the device.

Software can distinguish between the MPC750 and the MPC755 by reading the PVR. The MPC755 PVR reads as 0x0008_3100. The version is 0x0008 and the revision level starts at 0x3100.

2.1.3 Hardware Implementation-Dependent Register 2 (HID2)

The MPC755 implements an additional hardware implementation-dependent register not described in the *MPC750 User's Manual*, shown in Figure 4. It is a supervisor-only, read/write, implementation-specific special purpose register (SPR) which is accessed as SPR 1011 (decimal).

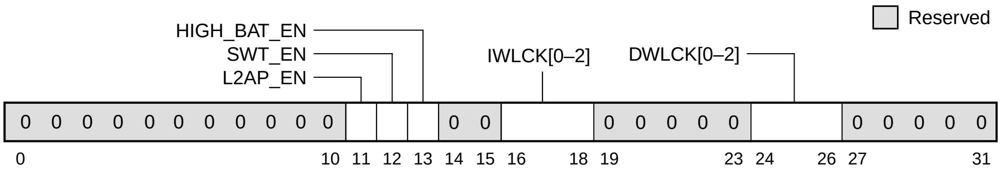


Figure 4. Hardware Implementation-Dependent Register 2 (HID2)

Table 2 describes the HID2 fields.

Table 2. Hardware Implementation Dependent Register 2 (HID2) Field Descriptions

Bit	Name	Description
0–10	—	Reserved
11	L2AP_EN	L2 address parity enable. When this bit is set, some of the L2 address signals are used in the parity generated on L2DP[0:7]. See Section 9.5, "L2 Address and Data Parity Signals," for the combinations supported.
12	SWT_EN	Software table search enable. Setting this bit causes one of three new exceptions when a TLB miss occurs. See Part IV, "MPC755 Exceptions (Chapter 4)," and Part V, "MPC755 Memory Management (Chapter 5)," for more information on the use of software table search operations.
13	HIGH_BAT_EN	IBAT[4–7] and DBAT[4–7] enable. When this bit is set, four more IBAT and DBAT entries are available for translating blocks of memory. See Part II, "The MPC755 Programming Model (Chapter 2)," for more information on the SPR numbers used for accessing the new BATs.
14–15	—	Reserved

Table 2. Hardware Implementation Dependent Register 2 (HID2) Field Descriptions

Bit	Name	Description
16–18	IWLCK[0–2]	Instruction cache way lock. Useful for locking blocks of instructions into the instruction cache for time-critical applications that require deterministic behavior. See Section 3.2.3, “Performing Cache Locking.” 000 = no ways locked 001 = way0 locked 010 = way0 thru way1 locked 011 = way0 thru way2 locked 100 = way0 thru way3 locked 101 = way0 thru way4 locked 110 = way0 thru way5 locked 111 = Reserved
19–23	—	Reserved
24–26	DWLCK[0–2]	Data cache way lock. Useful for locking blocks of data into the data cache for time-critical applications where deterministic behavior is required. See Section 3.2.3, “Performing Cache Locking.” 000 = no ways locked 001 = way0 locked 010 = way0 thru way1 locked 011 = way0 thru way2 locked 100 = way0 thru way3 locked 101 = way0 thru way4 locked 110 = way0 thru way5 locked 111 = Reserved
27–31	—	Reserved

2.2 MPC750 and MPC755 Instruction Use

This section describes some restrictions of the **stdf**, **mtsr**, and **mtsrin** instructions on both the MPC750 and MPC755. In addition, the **dcbz** instruction has cache coherency implications described in Section 3.1.2, “dcbz and L1 Cache Coherency.”

2.2.1 stfd Instruction Use

The MPC750 and MPC755 require that the FPRs be initialized with floating-point values before the **stfd** instruction is used. Otherwise, a random power-on value for an FPR may cause unpredictable device behavior when the **stfd** instruction is executed. Note that any floating-point value loaded into the FPRs is acceptable.

2.2.2 isync Instruction Use with mtsr and mtsrin

The MPC750 and MPC755 have a restriction on the use of the **mtsr** and **mtsrin** instructions not described in *The Programming Environments Manual* or in the *MPC750 User’s Manual*. The MPC750 and MPC755 require that an **isync** instruction be executed after either an **mtsr** or **mtsrin** instruction. This **isync** instruction must occur after the execution of the **mtsr** or **mtsrin** and before the data address translation mechanism uses any of the on-chip segment registers.

2.3 tlbld and tlbli Instructions

This section provides a detailed description of the two implementation-specific instructions used for software table search operations—**tlbld** and **tlbli** (same as the MPC603e).

The address translation mechanism is defined in terms of segment descriptors and page table entries (PTEs) used by PowerPC processors to locate the effective-to-physical address mapping for a particular access. The PTEs reside in page tables in memory. As defined for 32-bit implementations by the PowerPC architecture, segment descriptors reside in 16 on-chip segment registers.

Similar to the MPC603e, the MPC755 provides two implementation-specific instructions (**tlbld** and **tlbli**) that are used by software table search operations following TLB misses to load TLB entries on-chip (not provided by the MPC750 because the MPC750 does not support software table search operations).

Refer to Part V, "MPC755 Memory Management (Chapter 5)," for more information about the TLB registers and software table search operations with the MPC755. Table 3 lists the TLB instructions implemented in the MPC755.

Table 3. Translation Lookaside Buffer Management Instructions

Name	Mnemonic	Operand Syntax
TLB Invalidate Entry	**tlbie**	**r**B
TLB Synchronize	**tlbsync**	—
Load Data TLB Entry	**tlbld**	**r**B
Load Instruction TLB Entry	**tlbli**	**r**B

Because the presence and exact semantics of the translation lookaside buffer management instructions are implementation-dependent, system software should incorporate uses of the instructions into subroutines to maximize compatibility with programs written for other processors.

For more information on the PowerPC instruction set, refer to Chapter 4, "Addressing Modes and Instruction Set Summary," and Chapter 8, "Instruction Set," in *The Programming Environments Manual*.

tlbld tlbld

Load Data TLB Entry Integer Unit

tlbld **r**B

▨ Reserved

31	▨ 0 0 0 0 0	▨ 0 0 0 0 0	B	978	▨ 0
0 5	6 10	11 15	16 20	21 30	31

EA ← (**r**B)
TLB entry created from DCMP and RPA
DTLB entry selected by EA[15-19] and SRR1[WAY] ← created TLB entry

The EA is the contents of **r**B. The **tlbld** instruction loads the contents of the data PTE compare (DCMP) and required physical address (RPA) registers into the first word of the selected data TLB entry. The specific DTLB entry to be loaded is selected by the EA and the SRR1[WAY] bit.

The **tlbld** instruction should only be executed when address translation is disabled (MSR[IR] = 0 and MSR[DR] = 0).

Note that it is possible to execute the **tlbld** instruction when address translation is enabled; however, extreme caution should be used in doing so. If data address translation is enabled (MSR[DR] = 1) **tlbld** must be preceded by a **sync** instruction and succeeded by a context synchronizing instruction.

Note also that care should be taken to avoid modification of the instruction TLB entries that translate current instruction prefetch addresses.

This is a supervisor-level instruction; it is also a MPC755-specific instruction, and not part of the PowerPC instruction set.

Other registers altered:

- None

tlbli tlbli

Load Instruction TLB Entry Integer Unit

tlbli **r**B

▨ Reserved

31	0 0 0 0 0	0 0 0 0 0	B	1010	0
0 5	6 10	11 15	16 20	21 30	31

EA ← (**r**B)
TLB entry created from ICMP and RPA
ITLB entry selected by EA[15-19] and SRR1[WAY] ← created TLB entry

The EA is the contents of **r**B. The **tlbli** instruction loads the contents of the instruction PTE compare (ICMP) and required physical address (RPA) registers into the first word of the selected instruction TLB entry. The specific ITLB entry to be loaded is selected by the EA and the SRR1[WAY] bit.

The **tlbli** instruction should only be executed when address translation is disabled (MSR[IR] = 0 and MSR[DR] = 0).

Note that it is possible to execute the **tlbli** instruction when address translation is enabled; however, extreme caution should be used in doing so. If instruction address translation is enabled (MSR[IR] = 1), **tlbli** must be followed by a context synchronizing instruction such as **isync** or **rfi**.

Note also that care should be taken to avoid modification of the instruction TLB entries that translate current instruction prefetch addresses.

This is a supervisor-level instruction; it is also a MPC755-specific instruction, and not part of the PowerPC instruction set.

Other registers altered:

- None

Part III MPC755 L1 Instruction and Data Cache Operation (Chapter 3)

This section describes L1 cache coherency issues and also describes the new instruction and data cache way locking features of the MPC755 embedded processor. Otherwise, the L1 instruction and data cache operation is the same as the MPC750.

The MPC755 includes a mechanism for allocating cache entries for a particular group of ways for both the instruction and data caches. If a way is locked, the data loaded in that cache way will not be replaced by an access to another address; that is, none of the entries in a locked cache way are re-allocated. One to six of the eight ways in a cache can be locked with the IWLCK and DWLCK bits of the HID2 register. All eight ways of a cache can be locked using the ILOCK or DLOCK bits of the HID0 register.

Note that integrated devices based on the MPC603e G2 processor core may also implement entire and cache way locking. However, the G2-based processor caches are only four-way set-associative, so only up to three ways can be locked. Additionally, the bit encodings in HID2 for enabling way-locking differ from the encodings used in the MPC755 and they do not correspond. Even though the G2 core processors also define similar IWLCK[0–2] and DWLCK[0–2] fields in HID2, the encodings are distinctly different.

3.1 L1 Cache Coherency

This section describes some L1 coherency precautions for the MPC755 in addition to that described in the *MPC750 User's Manual*.

3.1.1 Coherency Precautions in Single Processor Systems

Note that as described in the *MPC750 User's Manual*, great care must be taken when the WIMG bits are changed in the MMU. The following coherency paradoxes can be encountered within a single-processor system:

- Load or store to a caching-inhibited page (WIMG = x1xx) and a cache hit occurs.

  The MPC755 ignores any hits to an L1 cache block in a memory space marked caching-inhibited (WIMG = x1xx). The L1 cache is bypassed and the access is performed externally as if there were no hit. The data in the cache is not pushed, and the cache block is not invalidated.

  This operation is similar to that of the MPC750 except that in the case of the MPC750, the access is performed to the 60x bus. In the case of the MPC755, the access is performed to the private memory space if private memory is enabled, and if the upper order address bits match the value in L2PM[PMBA]. Alternatively, the access may hit in the L2 cache if it was previously designated as cacheable but the WIMG bits were changed so that the access is cache-inhibited. Although the access may hit in the L2 (if the data was previously loaded when the WIMG bits were set to caching-allowed), the L2 cache does not allocate any new entries for caching-inhibited data. This L2 cache behavior is different than that of the MPC750 for this case.

- Store to a page marked write-through (WIMG = 1xxx) and a cache hit occurs to a modified cache block.

  The MPC750 and MPC755 work identically in this case and ignore the modified bit in the cache tag. The cache block is updated during the write-through operation but the block remains in the modified state (M).

Note that when WIM bits are changed in the page tables or BAT registers, it is critical that the cache contents reflect the new WIM bit settings. For example, if a block or page that had allowed caching becomes caching-inhibited, software should ensure that the appropriate cache blocks are flushed to memory and invalidated.

3.1.2 dcbz and L1 Cache Coherency

Both the MPC750 and MPC755 processors require protection in the use of the **dcbz** instruction in order to guarantee cache coherency in a multiprocessor system. Specifically, the **dcbz** instruction must be:

- Either enveloped by high-level software synchronization protocols (such as semaphores), or
- Preceded by execution of a **dcbf** instruction to the same address.

One of these precautions must be taken in order to guarantee that there are no simultaneous cache hits from a **dcbz** instruction and a snoop to that address. If these two events occur simultaneously, stale data may occur, causing system failures.

3.2 Cache Locking

This section describes the cache locking and cache-way locking features of the MPC755.

3.2.1 Cache Locking Terminology

Cache locking is the ability to prevent some or all of a microprocessor’s instruction or data cache from being overwritten. Cache locking can be set for either an entire cache or for individual ways within the cache as follows:

- Entire Cache Locking—When an entire cache is locked, data for read hits within the cache are supplied to the requesting unit in the same manner as hits from an unlocked cache. Similarly, writes that hit in the data cache are written to the cache in the same way as write hits to an unlocked cache. However, any access that misses in the cache is treated as a cache-inhibited access. Cache entries that are invalid at the time of locking remain invalid and inaccessible until the cache is unlocked. When 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 possible, but it is not possible to lock

only way0 and way2. When using way locking, at least two ways must be left unlocked. The maximum number of lockable ways is six on the MPC755 embedded processor (way0–way5).

Unlike entire cache locking, invalid entries in a locked way are accessible and available for data replacement. 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 invalid entries cannot be allocated. Unlocked ways of the cache behave normally.

Table 4 summaries the MPC755 cache organization.

Table 4. Cache Organization

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

3.2.2 Cache Locking Register Summary

Table 5 through Table 7 outline the registers and bits used to perform cache locking on the MPC755 embedded processor. Refer to the *MPC750 RISC Microprocessor User's Manual* for a complete description of the HID0 and MSR registers. Refer to Part II, “The MPC755 Programming Model (Chapter 2),” for a complete description of the HID2 register.

Table 5. HID0 Bits Used to Perform Cache Locking

Bit	Name	Description
16	ICE	Instruction cache enable. This bit must be set for instruction cache locking. See Section 3.2.3.1.1, “Enabling the Data Cache.”
17	DCE	Data cache enable. This bit must be set for data cache locking. See Section 3.2.3.1.1, “Enabling the Data Cache.”
18	ILOCK	Instruction cache lock. Set to lock the entire instruction cache. See Section 3.2.3.2.6, “Entire Instruction Cache Locking.”
19	DLOCK	Data cache lock. Set to lock the entire data cache. See Section 3.2.3.1.6, “Entire Data Cache Locking.”
20	ICFI	Instruction cache flash invalidate. Setting and then clearing this bit invalidates the entire instruction cache. See Section 3.2.3.2.8, “Invalidating the Instruction Cache (Even if Locked).”
21	DCFI	Data cache flash invalidate. Setting and then clearing this bit invalidates the entire data cache. See Section 3.2.3.1.4, “Invalidating the Data Cache.”
22	SPD	Speculative cache access disable. This bit must be cleared for instruction cache locking. See Section 3.2.3.2.5, “MPC755 Prefetching Considerations.”
25	DCFA	Data cache flush assist. This bit must be set for data cache flushing. See Section 3.2.3.1.4, “Invalidating the Data Cache.”
29	BHT	Branch history table enable. This bit must be cleared for instruction cache locking. See Section 3.2.3.2.5, “MPC755 Prefetching Considerations.”

Table 6. HID2 Bits Used to Perform Cache Locking

Bits	Name	Description
16–18	IWLCK	Instruction cache way lock. These bits are used to lock individual ways in the instruction cache. See Section 3.2.3.2.7, “Instruction Cache Way Locking.”
24–26	DWLCK	Data cache way lock. These bits are used to lock individual ways in the data cache. See Section 3.2.3.1.7, “Data Cache Way Locking.”

Table 7. MSR Bits Used to Perform Cache Locking

Bit	Name	Description
16	EE	External interrupt enable. This bit must be cleared during instruction and data cache loading. See Section 3.2.3.1.3, “Disabling Exceptions for Data Cache Locking.”
19	ME	Machine check enable. This bit must be cleared during instruction and data cache loading. See Section 3.2.3.1.3, “Disabling Exceptions for Data Cache Locking.”
26	IR	Instruction address translation. This bit must be set to enable instruction address translation by the MMU. See Section 3.2.3.1.2, “Address Translation for Data Cache Locking.”
27	DR	Data address translation. This bit must be set to enable data address translation by the MMU. See Section 3.2.3.1.2, “Address Translation for Data Cache Locking.”

3.2.3 Performing Cache Locking

This section outlines the basic procedures for locking the data and instruction caches and provides some example code for locking the caches. The procedures for the data cache are described first, followed by the corresponding sections for locking the instruction cache.

The basic procedures for cache locking are:

- Enabling the cache
- Enabling address translation for example code
- Disabling exceptions
- Loading the cache
- Locking the cache (entire cache locking or cache way locking)

In addition, this section describes how to invalidate the data and instruction caches, even when they are locked.

3.2.3.1 Data Cache Locking

This section describes the procedures for performing data cache locking on the MPC755.

3.2.3.1.1 Enabling the Data Cache

To lock the data cache, the data cache enable bit HID0[DCE], bit 17, must be set. The assembly code below enables the data cache:

```
# Enable the data cache. This corresponds
# to setting DCE bit in HID0 (bit 17)

mfspr   r1, HID0
ori     r1, r1, 0x4000
sync
mtspr   HID0, r1
```

3.2.3.1.2 Address Translation for Data Cache Locking

Two distinct memory areas must be set up 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 or the block address translation (BAT) registers.

For the purposes of the cache locking example in this document, the 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. The second area is a 256-Mbyte block of memory (not all of the 256-Mbytes of memory is locked in the cache; this area is set up 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 8 summarizes the BAT settings used in this example.

Table 8. Example BAT Settings for Cache Locking

Area	Base Address	Memory Size	WIMG Bits	BATU Setting	BATL Setting
First	0xFFF0_0000	1 Mbyte	0b0100[1]	0xFFF0_001F	0xFFF0_0002[1]
Second	0x0000_0000	256 Mbyte	0b0000	0x0000_1FFF	0x0000_0002

[1] Cache-inhibited memory is not a requirement for data cache locking. A setting of 0xFFF0_0002 with a corresponding WIMG of 0b0000 marks the memory area as cacheable.

The block address translation upper (BATU) and block address translation lower (BATL) settings in Table 8 can be used for both instruction block address translation (IBAT) and data block address translation (DBAT) registers. After the BAT registers have been set up, 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
```

3.2.3.1.3 Disabling Exceptions for Data Cache Locking

To ensure that exception handler routines do not execute while the cache is being loaded (which could possibly pollute the cache with undesired contents) all exceptions must be disabled. This is accomplished by clearing the appropriate bits in the machine state register (MSR). See Table 9 for the bits within the MSR that must be cleared to ensure that exceptions are disabled.

Table 9. MSR Bits for Disabling Exceptions

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 may not need to be disabled because the example code shown in this document that performs cache locking does not execute any floating-point operations.

The following assembly code disables all asynchronous exceptions:

```
    # 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
```

3.2.3.1.4 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 performing a flash invalidate or a series of **dcbf** instructions that force a flush and invalidation of the data cache block.

The following code sequence shows how to flush the data cache:

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

    # CTR = number of data blocks to load
    # Number of blocks = (16K) / (32 Bytes/block)
    #             = 2^14 / 2^5 = 2^9 = 0x200
        li      r1, 0x200
        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
        loop2:
        dcbf    r0, r6
        addi    r6, r6, 32      # Find the next block
        bdnz    loop2           # Decrement the counter, and
                                #   branch if CTR != 0
```

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

```
    # Set and then clear the HID0[DCFI] bit, bit 21
        mfspr   r1, HID0
        mr      r2, r1
        ori     r1, r1, 0x0400
        mtspr   HID0, r1
        mtspr   HID0, r2
        sync
```

3.2.3.1.5 Loading the Data Cache

This section explains loading data into the data cache. The data cache can be 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
```

3.2.3.1.6 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, the HID0[DLOCK] must be cleared to 0. Setting the DLOCK bit must be preceded by a **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
```

3.2.3.1.7 Data Cache Way Locking

Data cache way locking is controlled by HID2[DWLCK], bits 24–26. Table 10 shows the HID2[DWLCK 0–2] settings for the MPC755 embedded processor.

Table 10. MPC755 DWLCK[0–2] Encodings

DWLCK[0–2]	Ways Locked
0b000	No ways locked
0b001	Way 0 locked
0b010	Ways 0 and 1 locked
0b011	Ways 0, 1, and 2 locked

Table 10. MPC755 DWLCK[0–2] Encodings (Continued)

DWLCK[0–2]	Ways Locked
0b100	Ways 0, 1, 2, and 3 locked
0b101	Ways 0, 1, 2, 3, and 4 locked
0b110	Ways 0, 1, 2, 3, 4, and 5 locked
0b111	Reserved

The assembly code below locks way0 of the MPC755 data cache:

```
# Lock way0 of the data cache
# This corresponds to setting dwlck(0:2) 0b001 (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
```

3.2.3.1.8 Invalidating the Data Cache (Even if Locked)

There are two methods to invalidate the instruction or data cache:

- Invalidate the entire cache by setting and then immediately clearing the data cache flash invalidate bit HIDO[DCFI], bit 21. Even when a cache is locked, toggling DCFI bit invalidates all of the data cache.
- The data cache block invalidate (**dcbi**) instruction can be used to invalidate individual cache blocks. The **dcbi** instruction invalidates blocks locked (either entire or way-locked) within the data cache.

3.2.3.2 Instruction Cache Locking

This section describes the procedures for performing instruction cache locking on the MPC755.

3.2.3.2.1 Enabling the Instruction Cache

To lock the instruction cache, the instruction cache enable bit HID0[ICE], bit 16 must be set.

```
# Enable the data cache. This corresponds
# to setting DCE bit in HID0 (bit 17)

    mfspr   r1, HID0
    ori     r1, r1, 0x8000
    sync
    mtspr   HID0, r1
```

3.2.3.2.2 Address Translation for Instruction Cache Locking

Two distinct memory areas must be set up 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 instructions to be locked reside.

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 or the block address translation (BAT) registers.

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 is locked in the cache; this area is set up as an example) that contains the instructions to lock. Both memory areas use identity translation (the logical memory address equals the physical memory address). Table 11 summarizes the BAT settings used in this example.

Table 11. Example BAT Settings for Cache Locking

Area	Base Address	Memory Size	WIMG Bits	BATU Setting	BATL Setting
First	0xFFF0_0000	1 Mbyte	0b0100[1]	0xFFF0_001F	0xFFF0_0022[1]
Second	0x0000_0000	256 Mbyte	0b0000	0x0000_1FFF	0x0000_0002

[1] 0xFFF0_0022 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 0xFFF0_0002 with a corresponding WIMG of 0b0000 marks the memory area as cacheable.

The block address translation upper (BATU) and block address translation lower (BATL) settings in Table 11 can be used for both instruction block address translation (IBAT) and data block address translation (DBAT) registers. After the BAT registers have been set up, 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
```

3.2.3.2.3 Disabling Exceptions for Instruction Cache Locking

To ensure that exception handler routines do not execute while the cache is being loaded (which could possibly pollute the cache with undesired contents) all exceptions must be disabled. This is accomplished by clearing the appropriate bits in the machine state register (MSR). See Table 12 for the bits within the MSR that must be cleared to ensure that exceptions are disabled.

Table 12. MSR Bits for Disabling Exceptions

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 may not need to be disabled because the example code shown in this document that performs cache locking does not execute any floating-point operations.

The following assembly code disables all asynchronous exceptions:

```
    # 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
```

3.2.3.2.4 Preloading Instructions into the Instruction Cache

To optimize performance, PowerPC processors automatically prefetch instructions into the instruction cache. This feature can be used to preload explicit instructions into the cache even when it is known that their execution will be canceled. Although the execution of the 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 exceptions are disabled, cache has been flushed,
    # 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 = non-zero 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/ non-zero 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 are fetched at the same time (this assumes a 64-bit 60x data bus; the preloading code does 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 execution of the divide, the processor starts fetching instructions speculatively 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 execution of all speculatively fetched instructions is 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 be locked in the cache and the desired instructions may not be in their expected way within the cache.

3.2.3.2.5 MPC755 Prefetching Considerations

Because the instruction cache preloading code relies on static branch prediction to ensure that the **beqlr+** instruction is predicted 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 cleared 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 MPC755 dynamic branch prediction is disabled, the branch history table bit HID0[BHT], bit 29, must be cleared. By default, the BHT is cleared out of reset.

3.2.3.2.6 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 assembly code locks the contents of the entire instruction cache.

```
# Set the ILOCK bit in HID0 (bit 18)

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

3.2.3.2.7 Instruction Cache Way Locking

Instruction cache way locking is controlled by the HID2[IWLCK], bits 16–18. Table 13 shows the HID2[IWLCK 0–2] settings for the MPC755 embedded processor.

Table 13. MPC755 IWLCK[0–2] Encodings

IWLCK [0–2]	Ways Locked
0b000	No ways locked
0b001	Way 0 locked
0b010	Ways 0 and 1 locked
0b011	Ways 0, 1, and 2 locked
0b100	Ways 0, 1, 2, and 3 locked
0b101	Ways 0, 1, 2, 3, and 4 locked
0b110	Ways 0, 1, 2, 3, 4, and 5 locked
0b111	Reserved

The assembly code below locks way0 of the MPC755 instruction cache:

```
# Lock way0 of the instruction cache
# This corresponds to setting iwlck(0:2) to 0b001 (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
```

3.2.3.2.8 Invalidating the Instruction Cache (Even if Locked)

There are two methods to invalidate the instruction cache. In the first way, invalidate the entire cache by setting and then immediately clearing the instruction cache flash invalidate bit (HID0[ICFI], bit 20). Even when a cache is locked, toggling the ICFI bit invalidates all of the instruction cache. The following assembly code invalidates the entire instruction cache:

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

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

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

In the second method, the instruction cache block invalidate (**icbi)** instruction can be used to invalidate individual cache blocks. The **icbi** instruction invalidates blocks in an entirely locked instruction cache for the MPC750 and the MPC755 microprocessors. On the MPC755 embedded processor, the **icbi** instruction invalidates way-locked blocks within the instruction cache.

Part IV MPC755 Exceptions (Chapter 4)

The exception model for the MPC755 is the same as that described in the *MPC750 User's Manual* except as described in this section. For both the MPC750 and MPC755, no combination of the thermal assist unit, the decrementer register, and the performance monitor can be used at any one time. If exceptions for any two of these functional blocks are enabled together, multiple exceptions caused by any of these three blocks cause unpredictable results.

The MPC755 has three new exceptions used to support software table search operations (the same as the MPC603e). Software table searching is enabled with the setting of HID2[SWT_EN], bit 12. When this bit is cleared, the MPC755 uses the hardware table searching mechanism of the MPC750 when a miss occurs in an on-chip TLB. When HID2[SWT_EN] = 1, software table searching is enabled and a TLB miss causes one of the exceptions described in this section.

See Section 2.3, “tlbld and tlbli Instructions,” for a detailed explanation of the **tlbli** and **tlbld** instructions used to load the TLBs. See Part V, “MPC755 Memory Management (Chapter 5),” for a more detailed explanation of the other resources used to perform table search operations in software and example exception handlers.

The three MMU exceptions used for software table search operations are described in Table 14.

Table 14. Software Table Search Exceptions and Conditions

Exception Type	Vector Offset (hex)	Causing Conditions
Instruction TLB miss	01000	An instruction TLB miss exception is caused when an effective address for an instruction fetch cannot be translated by the ITLB.
Data TLB miss for load	01100	A data TLB miss for load exception is caused when an effective address for a data load operation cannot be translated by the DTLB.
Data TLB miss for store	01200	A data TLB miss for store exception is caused when an effective address for a data store operation cannot be translated by the DTLB, or where a DTLB hit occurs, and the change bit in the PTE must be set due to a data store operation.

The SRR0, SRR1, and MSR registers are used by the MPC755 when an exception occurs. Register settings for the instruction and data TLB miss exceptions are described in Table 15.

Table 15. Instruction and Data TLB Miss Exceptions—Register Settings

Register	Setting Description			
SRR0	Set to the address of the next instruction to be executed in the program for which the TLB miss exception was generated.			
SRR1	0–3 Loaded from condition register CR0 field 4–11 Cleared 12 KEY. Key for TLB miss (either Ks or Kp from segment register, depending on whether the access is a user or supervisor access). See Figure 5. 13 D/I. Data or instruction access 0 = data TLB miss 1 = instruction TLB miss 14 WAY. Next TLB set to be replaced (set per LRU) 0 = replace TLB associativity set 0 1 = replace TLB associativity set 1 15 S/L. Store or load data access 0 = data TLB miss on load 1 = data TLB miss on store (or C = 0) 16–31 Loaded from bits 16–31 of the MSR			
MSR[1]	POW 0 ILE — IP —	EE 0 PR 0 FP 0 ME —	FE0 0 SE 0 BE 0 FE1 0	IR 0 DR 0 RI 0 LE Set to value of ILE

[1] MSR[14] (the TGPR bit) of the MPC603e processor provided control for a separate set of four temporary GPRs that could be used as general-purpose registers in the TLB miss exception handler routines. MSR[14] is reserved on the MPC755, and the new SPRG[4–7] can be used for the TLB miss handler code.

The MPC755 automatically saves the values of CR[CR0] of the executing context to SRR1[0–3]. Thus, the exception handler can set CR[CR0] bits and branch accordingly in the exception handler routine, without having to save the existing CR[CR0] bits. However, the exception handler must restore these bits to CR[CR0] before executing the **rfi** instruction.

Also saved in SRR1 are two bits identifying the type of miss (SRR1[D/I] identifies instruction or data, and SRR1[S/L] identifies a store or load). Additionally, SRR1[WAY] identifies the associativity class of the TLB entry selected for replacement by the LRU algorithm. The software can change this value, effectively overriding the replacement algorithm. Finally, the SRR1[KEY] bit is used by the table search software to determine if there is a protection violation associated with the access (useful on data write misses for determining if the C bit should be updated in the table).

The key bit, saved in SRR1 for a TLB miss exception, is derived as shown in Figure 5.

```
Select KEY from segment register:
        If MSR[PR] = 0, KEY = Ks
        If MSR[PR] = 1, KEY = Kp
```

Figure 5. Derivation of Key Bit for SRR1

4.1 Instruction TLB Miss Exception (0x01000)

When the effective address for an instruction fetch operation cannot be translated by the ITLBs or IBATs, an instruction TLB miss exception is generated. Register settings for the instruction and data TLB miss exceptions are described in Table 15. If the instruction TLB miss exception handler fails to find the desired PTE, then a page fault must be synthesized. The handler must restore the machine state before invoking the ISI exception (0x00400).

When an instruction TLB miss exception is taken, instruction execution for the handler begins at offset 0x01000 from the physical base address indicated by MSR[IP].

4.2 Data TLB Miss for Load Exception (0x01100)

When the effective address for a data load or cache operation cannot be translated by the DTLBs or DBATs, a data TLB miss for load exception is generated. Register settings for the instruction and data TLB miss exceptions are described in Table 15. If the data TLB miss exception handler fails to find the desired PTE, then a page fault must be synthesized. The handler must restore the machine state before invoking the DSI exception (0x00300).

When a data TLB miss for load exception is taken, instruction execution for the handler begins at offset 0x01100 from the physical base address indicated by MSR[IP].

4.3 Data TLB Miss for Store Exception (0x01200)

When the effective address for a data store or cache operation cannot be translated by the DTLBs or DBATs, a data TLB miss for store exception is generated. The data TLB miss for store exception is also taken when the changed bit (C = 0) for a DTLB entry needs to be updated for a store operation. Register settings for the instruction and data TLB miss exceptions are described in Table 15.

If the data TLB miss exception handler fails to find the desired PTE, then a page fault must be synthesized. The handler must restore the machine state before invoking the DSI exception (0x00300).

When a data TLB miss for store exception is taken, instruction execution for the handler begins at offset 0x01200 from the physical base address indicated by MSR[IP].

Part V MPC755 Memory Management (Chapter 5)

The MPC755 implements a virtual memory management scheme that is compliant with the PowerPC architecture for 32-bit microprocessors and that implements the software table searching features of the MPC603e. The organization of the memory management unit (MMU) hardware is as follows:

- Same as MPC750
  - 128-entry, two-way set associative data TLB
  - 128-entry, two-way set associative instruction TLB
  - Sixteen segment registers
  - Automatic hardware table search operations

- New features in the MPC755
  - 4- or 8-entry (HID2-selectable), fully associative instruction BAT array
  - 4- or 8-entry (HID2-selectable), fully associative data BAT array
  - Selectable software table search functionality by setting HID2[SWT_EN], bit 12.

The MPC755 has a set of implementation-specific registers, exceptions, and instructions that facilitate very efficient software searching of the page tables in memory. This section describes those resources and provides three example code sequences that can be used in an MPC755 system for an efficient search of the translation tables in software. These three code sequences can be used as handlers for the three exceptions requiring access to the PTEs in the page tables in memory—instruction TLB miss, data TLB miss on load, and data TLB miss on store exceptions.

Note that the remainder of the MMU definition and rules about updating the page tables in memory for the MPC755 are the same as that for the MPC750.

5.1 Software Table Search Resources

In addition to setting up the translation page tables in memory, the system software must assist the processor in loading PTEs into the on-chip TLBs. When a required TLB entry is not found in the appropriate TLB, the processor vectors to one of the three TLB miss exception handlers so that the software can perform a table search operation and load the TLB. When this occurs, the processor automatically saves information about the access and the executing context. Table 16 provides a summary of the implementation-specific exceptions, registers, and instructions that can be used by the TLB miss exception handler software in MPC755 systems. See Section 2.3, “tlbld and tlbli Instructions,” for detailed information about the operation of the **tlbli** and **tlbld** instructions and Part IV, “MPC755 Exceptions (Chapter 4),” for more information about exception processing on the MPC755.

Table 16. Implementation-Specific Resources for Software Table Search Operations—Summary

Resource	Name	Description
Exceptions	Instruction TLB miss exception (vector offset 0x1000)	No matching entry found in ITLB
	Data TLB miss for load exception (vector offset 0x1100)	No matching entry found in DTLB for a load data access
	Data TLB miss for store exception—also caused when changed bit must be updated (vector offset 0x1200)	No matching entry found in DTLB for a store data access or matching DLTB entry has C = 0 and access is a store

Table 16. Implementation-Specific Resources for Software Table Search Operations—Summary

Resource	Name	Description
Registers	IMISS and DMISS	When a TLB miss exception occurs, the IMISS or DMISS register contains the 32-bit effective address of the instruction or data access that caused the miss exception.
	ICMP and DCMP	The ICMP and DCMP registers contain the word to be compared with the first word of a PTE in the table search software routine to determine if a PTE contains the address translation for the instruction or data access. The contents of ICMP and DCMP are automatically derived by the MPC755 when a TLB miss exception occurs.
	HASH1 and HASH2	The HASH1 and HASH2 registers contain the primary and secondary PTEG addresses that correspond to the address causing a TLB miss. These PTEG addresses are automatically derived by the MPC755 by performing the primary and secondary hashing function on the contents of IMISS or DMISS, for an ITLB or DTLB miss exception, respectively.
	RPA	The system software loads a TLB entry by loading the second word of the matching PTE entry into the RPA register and then executing the **tlbli** or **tlbld** instruction (for loading the ITLB or DTLB, respectively).
Instructions	**tlbli r**B	Loads the contents of the ICMP and RPA registers into the ITLB entry selected by <ea> and SRR1[WAY]
	tlbld rB	Loads the contents of the DCMP and RPA registers into the DTLB entry selected by <ea> and SRR1[WAY]

In addition, the MPC755 contains four additional SPRG registers SPRG[4–7] that have been implemented to save and restore general-purpose registers used by the exception handler. This is a replacement for having the MSR[TGPR] bit of the MPC603e and four temporary general-purpose registers. Note that the MSR[TGPR] bit is not implemented in the MPC755. If software table searching is not enabled, then these registers may be used for any supervisor purpose.

5.2 Software Table Search Registers

This section describes the format of the implementation-specific SPRs that are not defined by the PowerPC architecture, but are used by the TLB miss exception handlers. These registers can be accessed by supervisor-level instructions only. Any attempt to access these SPRs with user-level instructions results in a privileged instruction program exception. Because DMISS, IMISS, DCMP, ICMP, HASH1, HASH2, and RPA are used to access the translation tables for software table search operations, they should only be accessed when address translation is disabled (that is, MSR[IR] = 0 and MSR[DR] = 0). Note that MSR[IR] and MSR[DR] are cleared by the processor whenever an exception occurs.

5.2.1 Data and Instruction TLB Miss Address Registers (DMISS and IMISS)

The DMISS and IMISS registers have the same format as shown in Figure 6. They are loaded automatically upon a data or instruction TLB miss. The DMISS and IMISS contain the effective page address of the access that caused the TLB miss exception. The contents are used by the processor when calculating the values of HASH1 and HASH2, and by the **tlbld** and **tlbli** instructions when loading a new TLB entry. Note that the MPC755 always loads a big-endian address into the DMISS register. These registers are read-only to the software.

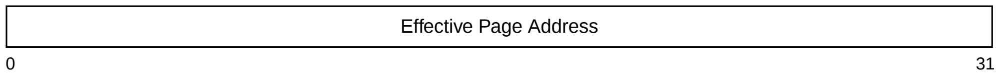


Figure 6. DMISS and IMISS Registers

5.2.2 Data and Instruction TLB Compare Registers (DCMP and ICMP)

The DCMP and ICMP registers are shown in Figure 7. These registers contain the first word in the required PTE. The contents are constructed automatically from the contents of the segment registers and the effective address (DMISS or IMISS) when a TLB miss exception occurs. Each PTE read from the tables in memory during the table search process should be compared with this value to determine whether or not the PTE is a match. Upon execution of a **tlbld** or **tlbli** instruction, the contents of the DCMP or ICMP register are loaded into the first word of the selected TLB entry.

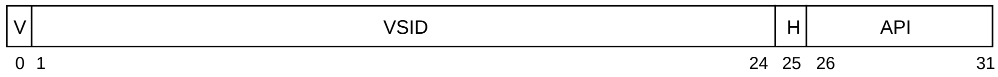


Figure 7. DCMP and ICMP Registers

Table 17 describes the bit settings for the DCMP and ICMP registers.

Table 17. DCMP and ICMP Bit Settings

Bit	Name	Description
0	V	Valid bit. Set by the processor on a TLB miss exception.
1–24	VSID	Virtual segment ID. Copied from VSID field of corresponding segment register.
25	H	Hash function identifier. Cleared by the processor on a TLB miss exception.
26–31	API	Abbreviated page index. Copied from API of effective address.

5.2.3 Primary and Secondary Hash Address Registers (HASH1 and HASH2)

HASH1 and HASH2 contain the physical addresses of the primary and secondary PTEGs for the access that caused the TLB miss exception. Only bits 7–25 differ between them. For convenience, the processor automatically constructs the full physical address by routing bits 0–6 of SDR1 into HASH1 and HASH2 and clearing the lower six bits. These registers are read-only and are constructed from the contents of the DMISS or IMISS register. The format for the HASH1 and HASH2 registers is shown in Figure 8.

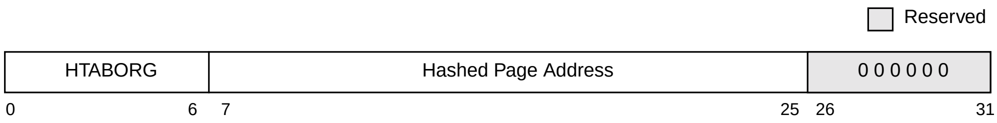


Figure 8. HASH1 and HASH2 Registers

Table 18 describes the bit settings of the HASH1 and HASH2 registers.

Table 18. HASH1 and HASH2 Bit Settings

Bit	Name	Description
0–6	HTABORG[0–6]	Copy of the upper 7 bits of the HTABORG field from SDR1
7–25	Hashed page address	Address bits 7–25 of the PTEG to be searched.
26–31	—	Reserved

5.2.4 Required Physical Address Register (RPA)

The RPA is shown in Figure 9. During a page table search operation, the software must load the RPA with the second word of the correct PTE. When the **tlbld** or **tlbli** instruction is executed, data from the IMISS and ICMP (or DMISS and DCMP) and the RPA registers is merged and loaded into the selected TLB entry. The TLB entry is selected by the effective address of the access (loaded by the table search software from the DMISS or IMISS register) and the SRR1[WAY] bit.

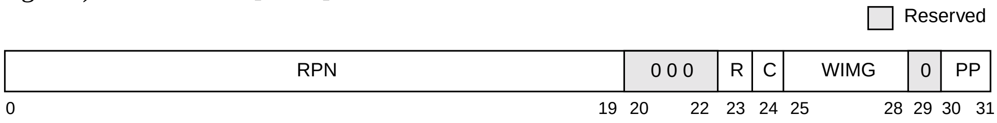


Figure 9. Required Physical Address (RPA) Register

Table 19 describes the bit settings of the RPA register.

Table 19. RPA Bit Settings

Bit	Name	Description
0–19	RPN	Physical page number from PTE
20–22	—	Reserved
23	R	Referenced bit from PTE
24	C	Changed bit from PTE
25–28	WIMG	Memory/cache access attribute bits
29	—	Reserved
30–31	PP	Page protection bits from PTE

5.3 Software Table Search Operation

When a TLB miss occurs and software table searching is enabled, the instruction or data MMU loads the IMISS or DMISS register, respectively, with the effective address of the access. The processor completes all instructions dispatched prior to the exception, status information is saved in SRR1, and one of the three TLB miss exceptions is taken. In addition, the processor loads the ICMP or DCMP register with the value to be compared with the first word of PTEs in the tables in memory.

The software should then access the first PTE at the address pointed to by HASH1. The first word of the PTE should be loaded and compared to the contents of DCMP or ICMP. If there is a match, then the required PTE has been found and the second word of the PTE is loaded from memory into the RPA register. Then the **tlbli** or **tlbld** instruction is executed, which loads the contents of the ICMP (or DCMP) and RPA registers into the selected TLB entry. The TLB entry is selected by the effective address of the access and the SRR1[WAY] bit.

If the compare did not result in a match, however, the PTEG address is incremented to point to the next PTE in the table and the above sequence is repeated. If none of the eight PTEs in the primary PTEG matches, the sequence is then repeated using the secondary PTEG (at the address contained in HASH2).

If the PTE is also not found in the eight entries of the secondary page table, a page fault condition exists, and a page fault exception must be synthesized. Thus the appropriate bits must be set in SRR1 (or DSISR) and the TLB miss handler must branch to either the ISI or DSI exception handler, which handles the page fault condition.

5.3.1 Flow for Example Exception Handlers

This section provides a flow diagram outlining some example software that can be used to handle the three TLB miss exceptions.

Figure 10 shows the flow for the example TLB miss exception handlers. The flow shown is common for the three exception handlers, except that the IMISS and ICMP registers are used for the instruction TLB miss exception while the DMISS and DCMP registers are used for the two data TLB miss exceptions. Also, for the cases of store instructions that cause either a TLB miss or require a table search operation to update the C bit, the flow shows that the C bit is set in both the TLB entry and the PTE in memory. Finally, in the case of a page fault (no PTE found in the table search operation), the setup for the ISI or DSI exception is slightly different for these two cases.

Figure 11 shows the flow for checking the R and C bits and setting them appropriately. Figure 12 shows the flow for synthesizing a page fault exception when no PTE is found. Figure 13 shows the flow for managing the cases of a TLB miss on an instruction access to guarded memory, and a TLB miss when C = 0 and a protection violation exists. The set up for these protection violation exceptions is very similar to that of page fault conditions (as shown in Figure 12) except that different bits in SRR1 (and DSISR) are set.

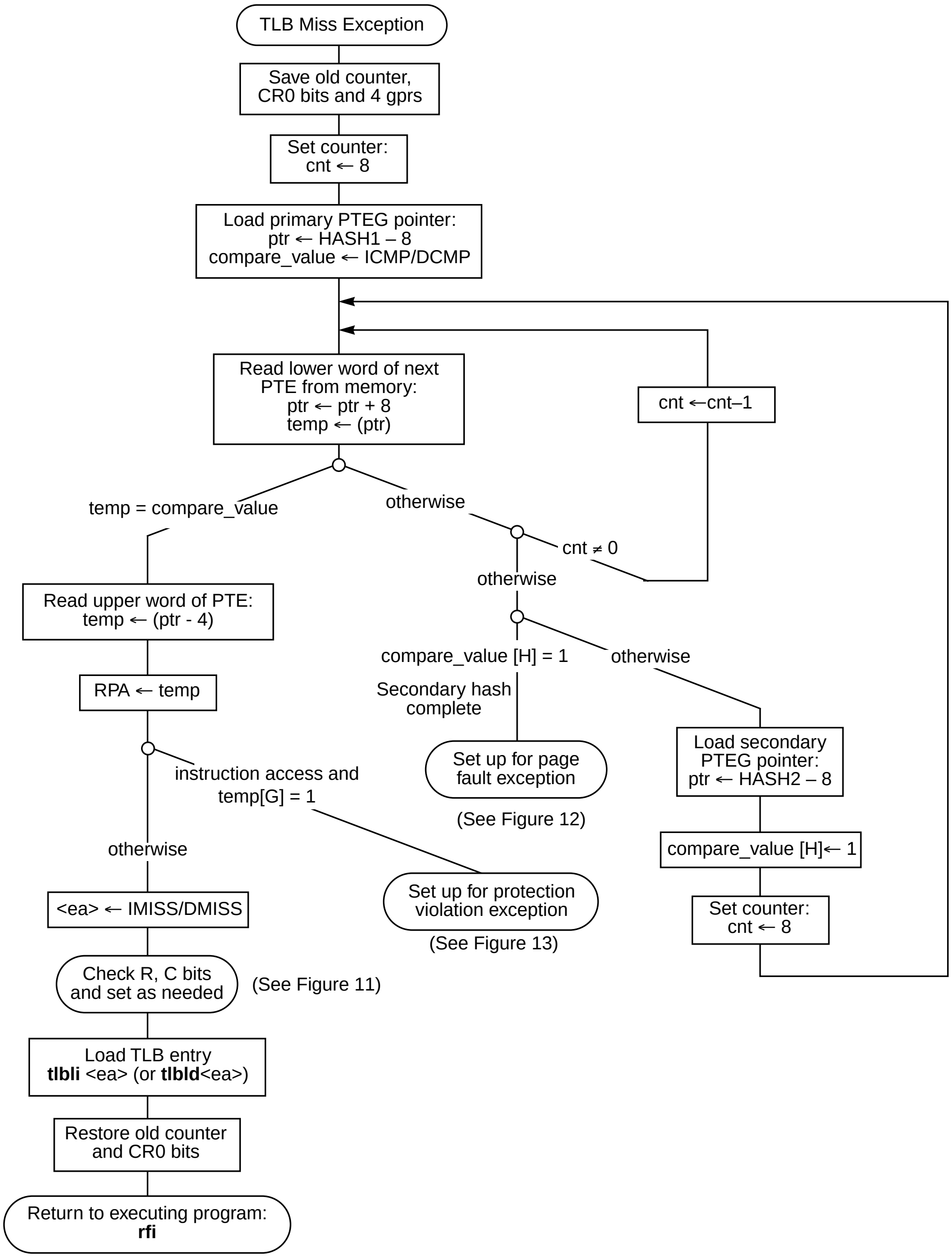

TLB Miss Exception
Save old counter, CR0 bits and 4 gprs
Set counter: cnt ← 8
Load primary PTEG pointer: ptr ← HASH1 – 8 compare_value ← ICMP/DCMP
Read lower word of next PTE from memory: ptr ← ptr + 8 temp ← (ptr)
cnt ←cnt–1
temp = compare_value
otherwise
cnt ≠ 0
otherwise
Read upper word of PTE: temp ← (ptr - 4)
compare_value [H] = 1
otherwise
RPA ← temp
Secondary hash complete
Load secondary PTEG pointer: ptr ← HASH2 – 8
instruction access and temp[G] = 1
Set up for page fault exception
(See Figure 12)
compare_value [H]← 1
otherwise
<ea> ← IMISS/DMISS
Set up for protection violation exception
(See Figure 13)
Set counter: cnt ← 8
Check R, C bits and set as needed
(See Figure 11)
Load TLB entry tlbli <ea> (or tlbld<ea>)
Restore old counter and CR0 bits
Return to executing program: rfi

Figure 10. Flow for Example Software Table Search Operation

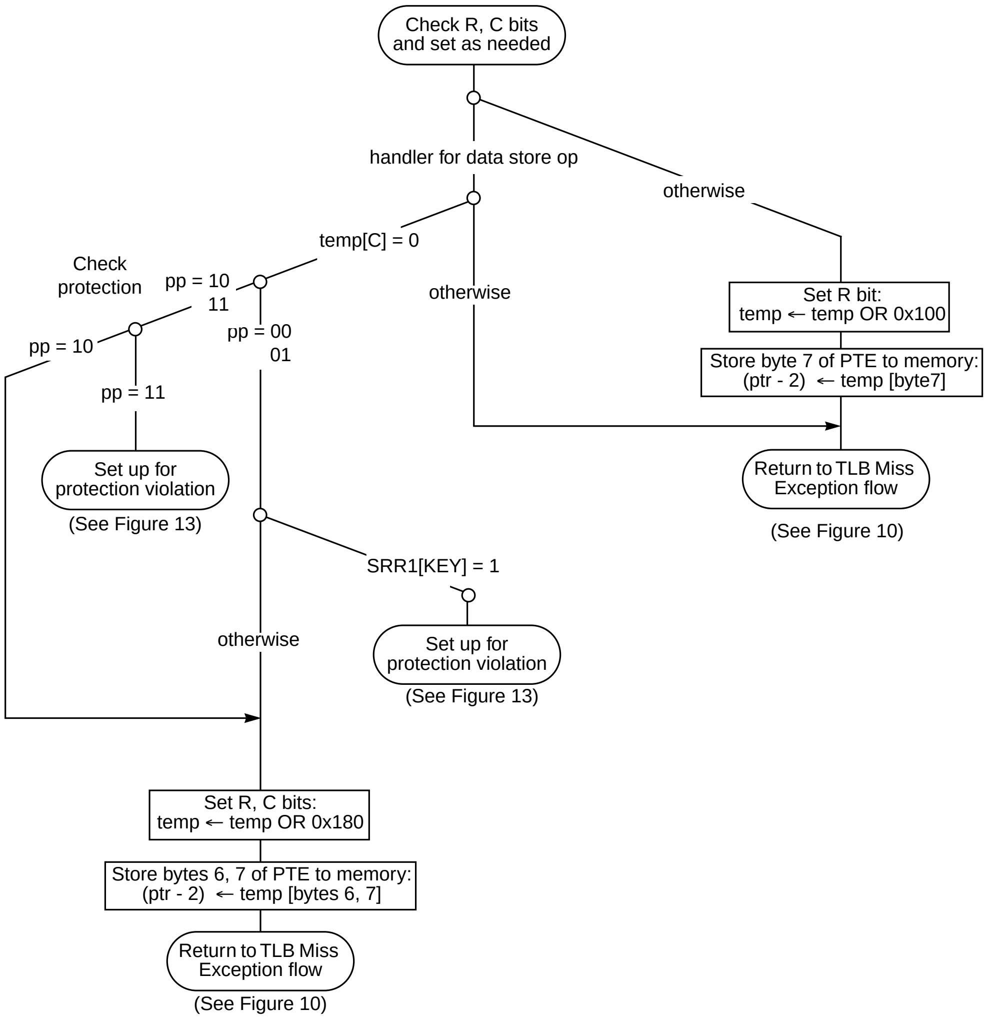

Check R, C bits and set as needed
handler for data store op
otherwise
temp[C] = 0
otherwise
Check protection
pp = 10
11
pp = 00
01
pp = 10
pp = 11
Set up for protection violation
(See Figure 13)
Set R bit: temp ← temp OR 0x100
Store byte 7 of PTE to memory: (ptr - 2) ← temp [byte7]
Return to TLB Miss Exception flow
(See Figure 10)
SRR1[KEY] = 1
otherwise
Set up for protection violation
(See Figure 13)
Set R, C bits: temp ← temp OR 0x180
Store bytes 6, 7 of PTE to memory: (ptr - 2) ← temp [bytes 6, 7]
Return to TLB Miss Exception flow
(See Figure 10)

Figure 11. Check and Set R and C Bit Flow

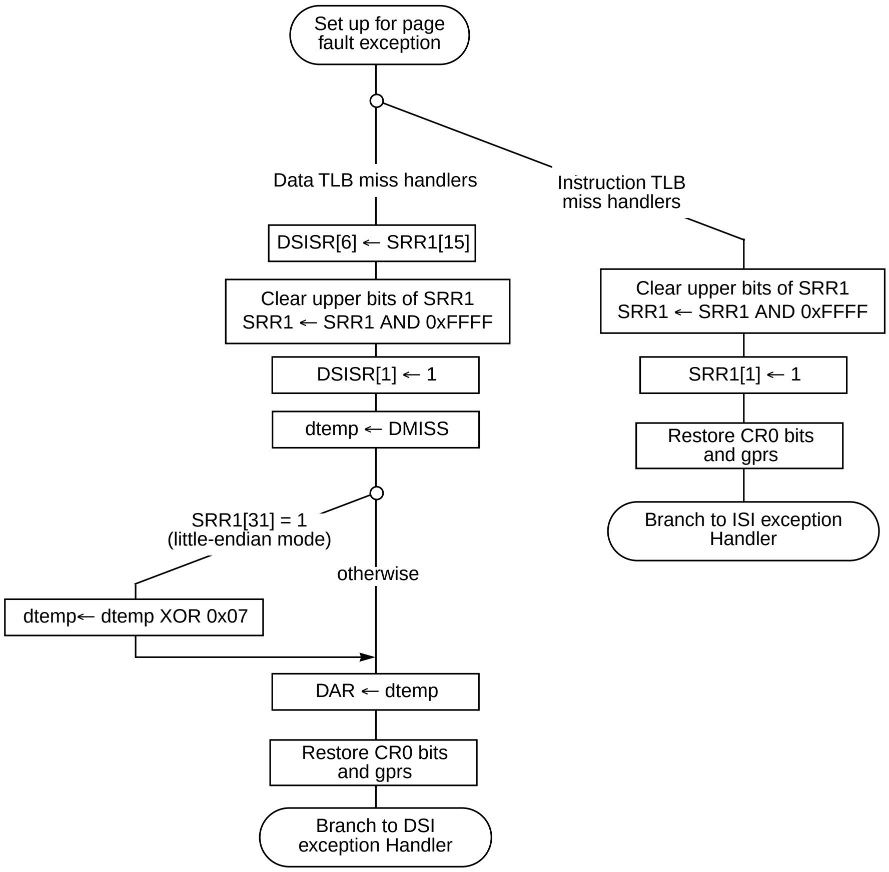


Figure 12. Page Fault Setup Flow

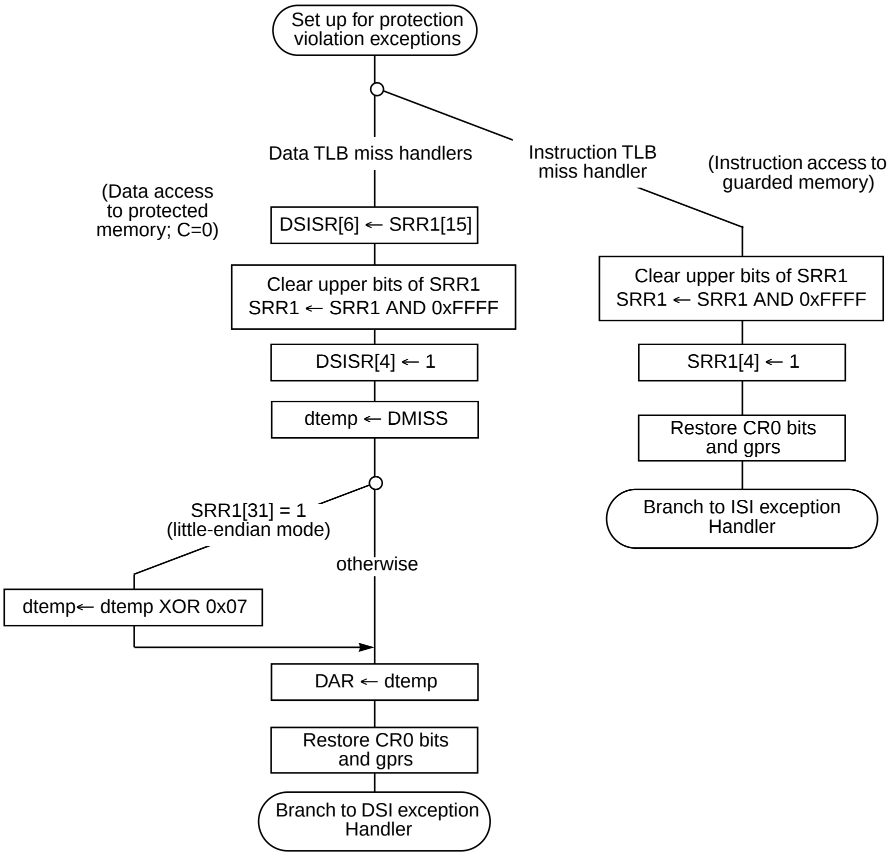

Set up for protection violation exceptions
Data TLB miss handlers
Instruction TLB miss handler
(Instruction access to guarded memory)
(Data access to protected memory; C=0)
DSISR[6] ← SRR1[15]
Clear upper bits of SRR1
SRR1 ← SRR1 AND 0xFFFF
DSISR[4] ← 1
dtemp ← DMISS
Clear upper bits of SRR1
SRR1 ← SRR1 AND 0xFFFF
SRR1[4] ← 1
Restore CR0 bits and gprs
Branch to ISI exception Handler
SRR1[31] = 1 (little-endian mode)
otherwise
dtemp← dtemp XOR 0x07
DAR ← dtemp
Restore CR0 bits and gprs
Branch to DSI exception Handler

Figure 13. Setup for Protection Violation Exceptions

5.3.2 Code for Example Exception Handlers

This section provides some assembly language examples that implement the flow diagrams described above. Note that although these routines fit into a few cache lines, they are supplied only as a functional example; they could be further optimized for faster performance.

```
# TLB software load for MPC755
#
# New Instructions:
#       tlbld           - write the dtlb with the pte in rpa reg
#       tlbli           - write the itlb with the pte in rpa reg
# New SPRs
#       dmiss           - address of dstream miss
#       imiss           - address of istream miss
#       hash1           - address primary hash PTEG address
#       hash2           - returns secondary hash PTEG address
#       iCmp            - returns the primary istream compare value
#       dCmp            - returns the primary dstream compare value
#       rpa             - the second word of pte used by tlblx
#
#
# there are three flows.
#   tlbDataMiss     - tlb miss on data load
#   tlbCeq0         - tlb miss on data store or store with tlb change bit == 0
#   tlbInstrMiss    - tlb miss on instruction fetch
#+
# place labels for rel branches
#-
#.machine PPC_755
# gpr r0..r3 are saved into SPRG4-7
.set    r0, 0
.set    r1, 1
.set    r2, 2
.set    r3, 3
.set    dMiss, 1010
.set    dCmp, 1011
.set    hash1, 1012
.set    hash2, 1013
.set    iMiss, 1014
.set    iCmp, 1015
.set    rpa, 1010
.set    c0, 0
.set    dar, 19
.set    dsisr, 18
.set    srr0, 26
.set    srr1, 27
.set    sprg4, 276
.set    sprg5, 277
.set    sprg6, 278
.set    sprg7, 279
.
.csect  tlbmiss[PR]
vec0:
.globl  vec0

.org    vec0+0x300
vec300:
.org    vec0+0x400
vec400:
#+
# Instruction TB miss flow
# Entry:
#       Vec = 1000
#       srr0    -> address of instruction that missed
#       srr1    -> 0:3=cr0 4=lru way bit 16:31 = saved MSR
#       iMiss   -> ea that missed
#       iCmp    -> the compare value for the va that missed
#       hash1   -> pointer to first hash pteg
#       hash2   -> pointer to second hash pteg
#
# Register usage:
#   Existing values of r0-r3 saved into sprg4-sprg7
#   r0-r3 used in the exception handler as follows
#       r0 is saved counter
#       r1 is junk
#       r2 is pointer to pteg
#       r3 is current compare value

.org    vec0+0x1000

tlbInstrMiss:
        mtspr   sprg4, r0       # save r0 into sprg4
        mtspr   sprg5, r1       # save r1 into sprg5
        mtspr   sprg6, r2       # save r2 into sprg6
        mtspr   sprg7, r3       # save r3 into sprg7
        mfspr   r2, hash1       # get first pointer
        addi    r1, 0, 8        # load 8 for counter
        mfctr   r0              # save counter
        mfspr   r3, iCmp        # get first compare value
        addi    r2, r2, -8      # pre dec the pointer
im0:    mtctr   r1              # load counter
im1:    lwzu    r1, 8(r2)       # get next pte
        cmp     c0, r1, r3      # see if found pte
        bdneq   im1             # dec count br if cmp ne and if count not zero
        bne     instrSecHash    # if not found set up second hash or exit
        l       r1, +4(r2)      # load tlb entry lower-word
        andi.   r3, r1, 8       # check G-bit
        bne     doISIp          # if guarded, take an ISI
        mtctr   r0              # restore counter
        mfspr   r0, iMiss       # get the miss address for the tlbli
        mfspr   r3, srr1        # get the saved cr0 bits
        mtcrf   0x80, r3        # restore CR0
        mtspr   rpa, r1         # set the pte
        ori     r1, r1, 0x100   # set reference bit
        srw     r1, r1, 8       # get byte 7 of pte
        tlbli   r0              # load the itlb
        stb     r1, +6(r2)      # update page table
        mfspr   r0, sprg4       # restore old value of r0
        mfspr   r1, sprg5       # restore old value of r1
        mfspr   r2, sprg6       # restore old value of r2
        mfspr   r3, sprg7       # restore old value of r3
        rfi                     # return to executing program
#+
# Register usage:
#       r0 is saved counter
#       r1 is junk
#       r2 is pointer to pteg
#       r3 is current compare value
#-
instrSecHash:
        andi.   r1, r3, 0x0040  # see if we have done second hash
        bne     doISI           # if so, go to ISI exception
        mfspr   r2, hash2       # get the second pointer
        ori     r3, r3, 0x0040  # change the compare value
        addi    r1, 0, 8        # load 8 for counter
        addi    r2, r2, -8      # pre dec for update on load
        b       im0             # try second hash
#+
# entry Not Found: synthesize an ISI exception
# guarded memory protection violation: synthesize an ISI exception
# Entry:
#       r0 is saved counter
#       r1 is junk
#       r2 is pointer to pteg
#       r3 is current compare value
#
doISIp:
        mfspr   r3, srr1        # get srr1
        andi.   r2,r3,0xFFFF    # clean upper srr1
        addis   r2, r2, 0x0800  # or in srr<4> = 1 to flag prot violation
        b       isi1:


doISI:
        mfspr   r3, srr1        # get srr1
        andi.   r2, r3, 0xFFFF  # clean srr1
        addis   r2, r2, 0x4000  # or in srr1<1> = 1 to flag pte not found
isi1    mtctr   r0              # restore counter
        mtspr   srr1, r2        # set srr1
        mtcrf   0x80, r3        # restore CR0
        mfspr   r0, sprg4       # restore old value of r0
        mfspr   r1, sprg5       # restore old value of r1
        mfspr   r2, sprg6       # restore old value of r2
        mfspr   r3, sprg7       # restore old value of r3
        b       vec400          # go to instr. access exception
#
#+
# Data TLB miss flow
# Entry:
#       Vec = 1100
#       srr0    -> address of instruction that caused data tlb miss
#       srr1    -> 0:3=cr0 4=lru way bit 5=1 if store 16:31 = saved MSR
#       dMiss   -> ea that missed
#       dCmp    -> the compare value for the va that missed
#       hash1   -> pointer to first hash pteg
#       hash2   -> pointer to second hash pteg
#
# Register usage:
#       r0 is saved counter
#       r1 is junk
#       r2 is pointer to pteg
#       r3 is current compare value
#-
.csect  tlbmiss[PR]
.org    vec0+0x1100

tlbDataMiss:
        mtspr   sprg4, r0       # save r0 into sprg4
        mtspr   sprg5, r1       # save r1 into sprg5
        mtspr   sprg6, r2       # save r2 into sprg6
        mtspr   sprg7, r3       # save r3 into sprg7
        mfspr   r2, hash1       # get first pointer
        addi    r1, 0, 8        # load 8 for counter
        mfctr   r0              # save counter
        mfspr   r3, dCmp        # get first compare value
        addi    r2, r2, -8      # pre dec the pointer
dm0:    mtctr   r1              # load counter
dm1:    lwzu    r1, 8(r2)       # get next pte
        cmp     c0, r1, r3      # see if found pte
        bdnzf   0, dm1          # dec count br if cmp ne and if count not zero
        bne     dataSecHash     # if not found set up second hash or exit
        l       r1, +4(r2)      # load tlb entry lower-word
        mtctr   r0              # restore counter
        mfspr   r0, dMiss       # get the miss address for the tlbld
        mfspr   r3, srr1        # get the saved cr0 bits
        mtcrf   0x80, r3        # restore CR0
        mtspr   rpa, r1         # set the pte
        ori     r1, r1, 0x100   # set reference bit
        srw     r1, r1, 8       # get byte 7 of pte
        tlbld   r0              # load the dtlb
        stb     r1, +6(r2)      # update page table
        mfspr   r0, sprg4       # restore old value of r0
        mfspr   r1, sprg5       # restore old value of r1
        mfspr   r2, sprg6       # restore old value of r2
        mfspr   r3, sprg7       # restore old value of r3
        rfi                     # return to executing program
#+
# Register usage:
#       r0 is saved counter
#       r1 is junk
#       r2 is pointer to pteg
#       r3 is current compare value
#-
dataSecHash:
        andi.   r1, r3, 0x0040  # see if we have done second hash
        bne     doDSI           # if so, go to DSI exception
        mfspr   r2, hash2       # get the second pointer
        ori     r3, r3, 0x0040  # change the compare value
        addi    r1, 0, 8        # load 8 for counter
        addi    r2, r2, -8      # pre dec for update on load
        b       dm0             # try second hash
#

#+
# C=0 in dtlb and dtlb miss on store flow
# Entry:
#       Vec = 1200
#       srr0    -> address of store that caused the exception
#       srr1    -> 0:3=cr0 4=lru way bit 5=1 16:31 = saved MSR
#       dMiss   -> ea that missed
#       dCmp    -> the compare value for the va that missed
#       hash1   -> pointer to first hash pteg
#       hash2   -> pointer to second hash pteg
#
# Register usage:
#       r0 is saved counter
#       r1 is junk
#       r2 is pointer to pteg
#       r3 is current compare value
#-

.csect  tlbmiss[PR]
.org    vec0+0x1200

tlbCeq0:
        mtspr   sprg4, r0       # save r0 into sprg4
        mtspr   sprg5, r1       # save r1 into sprg5
        mtspr   sprg6, r2       # save r2 into sprg6
        mtspr   sprg7, r3       # save r3 into sprg7
        mfspr   r2, hash1       # get first pointer
        addi    r1, 0, 8        # load 8 for counter
        mfctr   r0              # save counter
        mfspr   r3, dCmp        # get first compare value
        addi    r2, r2, -8      # pre dec the pointer
ceq0:   mtctr   r1              # load counter
ceq1:   lwzu    r1, 8(r2)       # get next pte
        cmp     c0, r1, r3      # see if found pte
        bdneq   ceq1            # dec count br if cmp ne and if count not zero
        bne     cEq0SecHash     # if not found set up second hash or exit
        l       r1, +4(r2)      # load tlb entry lower-word
        andi.   r3,r1,0x80      # check the C-bit
        beq     cEq0ChkProt     # if (C==0) go check protection modes
ceq2:   mtctr   r0              # restore counter
        mfspr   r0, dMiss       # get the miss address for the tlbld
        mfspr   r3, srr1        # get the saved cr0 bits
        mtcrf   0x80, r3        # restore CR0
        mtspr   rpa, r1         # set the pte
        tlbld   r0              # load the dtlb
        mfspr   r0, sprg4       # restore old value of r0
        mfspr   r1, sprg5       # restore old value of r1
        mfspr   r2, sprg6       # restore old value of r2
        mfspr   r3, sprg7       # restore old value of r3
        rfi                     # return to executing program
#+
# Register usage:
#       r0 is saved counter
#       r1 is junk
#       r2 is pointer to pteg
#       r3 is current compare value
#-
cEq0SecHash:
        andi.   r1, r3, 0x0040  # see if we have done second hash
        bne     doDSI           # if so, go to DSI exception
        mfspr   r2, hash2       # get the second pointer
        ori     r3, r3, 0x0040  # change the compare value
        addi    r1, 0, 8        # load 8 for counter
        addi    r2, r2, -8      # pre dec for update on load
        b       ceq0            # try second hash
#+
# entry found and PTE(c-bit==0):
# (check protection before setting PTE(c-bit)
# Register usage:
#       r0 is saved counter
#       r1 is PTE entry
#       r2 is pointer to pteg
#       r3 is trashed
#-
cEq0ChkProt:
        rlwinm. r3,r1,30,0,1    # test PP
        bge-    chk0            # if (PP==00 or PP==01) goto chk0:
        andi.   r3,r1,1         # test PP[0]
        beq+    chk2            # return if PP[0]==0
        b       doDSIp          # else DSIp

chk0:   mfspr   r3,srr1         # get old msr
        andis.  r3,r3,0x0008    # test the KEY bit (SRR0-bit 12)
        beq     chk2            # if (KEY==0) goto chk2:
        b       doDSIp          # else DSIp
chk2:   ori     r1, r1, 0x180   # set reference and change bit
        sth     r1, 6(r2)       # update page table
        b       ceq2            # and back we go
#
#+
# entry Not Found: synthesize a DSI exception
# Entry:
#       r0 is saved counter
#       r1 is junk
#       r2 is pointer to pteg
#       r3 is current compare value
#


doDSI:
        mfspr   r3, srr1        # get srr1
        rlwinm  r1,r3,9,6,6  # get srr1<flag> to bit 6 for load/store, zero rest
        addis   r1, r1, 0x4000  # or in dsisr<1> = 1 to flag pte not found
        b       dsi1:
doDSIp:
        mfspr   r3, srr1        # get srr1
        rlwinm  r1, r3,9,6,6 # get srr1<flag> to bit 6 for load/store, zero rest
        addis   r1, r1, 0x0800  # or in dsisr<4> = 1 to flag prot violation
dsi1:   mtctr   r0              # restore counter
        andi.   r2, r3, 0xFFFF  # clear upper bits of srr1
        mtspr   srr1, r2        # set srr1
        mtspr   dsisr, r1       # load the dsisr
        mfspr   r1, dMiss       # get miss address
        rlwinm. r2,r2,0,31,31   # test LE bit
        beq     dsi2:           # if little endian then:
        xor r1,r1,0x07          # de-mung the data address
```

```
dsi2:   mtspr   dar, r1         # put in dar
        mtcrf   0x80, r3        # restore CR0
        mfspr   r0, sprg4       # restore old value of r0
        mfspr   r1, sprg5       # restore old value of r1
        mfspr   r2, sprg6       # restore old value of r2
        mfspr   r3, sprg7       # restore old value of r3
        b       vec300          # branch to DSI exception
```

Part VI MPC755 Instruction Timing (Chapter 6)

The instruction timing of the MPC755 is identical to that of the MPC750 except for addition of the new **tlbli** and **tlbld** instructions.

Table 20 provides latencies for the new **tlbli** and **tlbld** instructions.

Table 20. TLB Load and Store Instruction Latencies

Primary Opcode	Extended Opcode	Mnemonic	Execution Unit	Clock Cycles
31	978	**tlbld**	LSU	2&
31	1010	**tlbli**	LSU	3&

Note: Cycle times marked with “&” require a variable number of cycles due to serialization.

Part VII MPC755 Signal Descriptions (Chapter 7)

This section describes two new signals that select the I/O voltages for the system bus (BVSEL) and the L2 interface (L2VSEL) as described in Table 21. Refer to the *MPC755 Hardware Specification* for more detailed information about these signals. All other MPC755 signals operate the same as the MPC750 signals.

Table 21. Voltage-Select Signal Descriptions

Signal	Comments
BVSEL	BVSEL and L2VSEL are assigned to two unused BGA positions on the MPC755’s 360-pin and MPC745’s 255-pin BGA footprint. Internal pullups are provided to default to MPC750-compatible I/O voltages if unconnected.
L2VSEL	

Part VIII MPC755 System Interface Operation (Chapter 8)

This section describes the MPC755 embedded processor bus interface and how its operation differs from the MPC750. It shows how the signals, defined in Chapter 7, “Signal Descriptions,” interact to perform address and data transfers and describes how the 32-bit bus mode is implemented on the MPC755.

8.1 MPC755 System Interface Overview

The system interface prioritizes requests for bus operations from the instruction and data caches, and performs bus operations in accordance with the protocol described in the *PowerPC Microprocessor Family: The Bus Interface for 32-Bit Microprocessors*. It includes address register queues, prioritization logic, and a bus control unit. The system interface latches snoop addresses for snooping in the data cache and in the address register queues, and for reservations controlled by the Load Word and Reserve Indexed (**lwarx**) and Store Word Conditional Indexed (**stwcx.**) instructions, and maintains the touch load address for the cache. The interface allows one level of pipelining; that is, with certain restrictions described later, there can be two outstanding transactions at any given time. Accesses are prioritized with load operations preceding store operations.

Instructions are automatically fetched from the memory system into the instruction unit where they are dispatched to the execution units at a peak rate of two instructions per clock. Conversely, load and store instructions explicitly specify the movement of operands to and from the integer and floating-point register files and the memory system.

When the MPC755 encounters an instruction or data access, it calculates the logical address and uses the low-order address bits to check for a hit in the on-chip, 32-Kbyte instruction and data caches. During cache lookup, the instruction and data memory management units (MMUs) use the higher-order address bits to calculate the virtual address, from which they calculate the physical address. The physical address bits are then compared with the corresponding cache tag bits to determine if a cache hit occurred in the L1 instruction or data cache. If the access misses in the corresponding cache, the physical address is used to access the L2 cache tags (if the L2 cache is enabled). If no match is found in the L2 cache tags, the physical address is used to access system memory.

In addition to the loads, stores, and instruction fetches, the MPC755 performs hardware table search operations following TLB misses; L2 cache cast-out operations when least-recently used cache lines are written to memory after a cache miss; and cache-line snoop push-out operations when a modified cache line experiences a snoop hit from another bus master.

Figure 1 shows the address path from the execution units and instruction fetcher through the translation logic to the caches and system interface logic.

The MPC755 uses separate address and data buses and a variety of control and status signals for performing reads and writes. The address bus is 32 bits and the data bus is 32 or 64 bits. The interface is synchronous—all MPC755 inputs are sampled, and all outputs are driven from the rising edge of the bus clock. The processor runs at a multiple of the system bus-clock speed. The MPC755 core operates at 1.9–2.1 volts, and the I/O signals operate at 1.8 or 3.3 volts.

8.2 Address Bus Pipelining

The MPC750 and MPC755 function identically in that the address bus pipelines an instruction transaction before previous data tenures complete for a data transaction. Conversely, the processor performs address bus pipelining for a data transaction following an instruction transaction. However, address bus pipelining does not occur for two consecutive instruction or two consecutive data transactions. Note that this behavior is not documented in the *MPC750 User’s Manual*.

8.3 Bus Clocking

Like the MPC750, the MPC755 requires a single system clock input (SYSCLK) used by the (PLL) circuit to generate a master clock for all of the CPU circuitry (including the bus interface circuitry) which is frequency- and phase-locked to the SYSCLK input. The master clock may be set to integer or half-clock multiples of the SYSCLK frequency. Refer to the *MPC755 Hardware Specification* for the ratios supported.

8.4 32-Bit Data Bus Mode

The MPC755 supports an optional 32-bit data bus mode in which the data bus high and corresponding parity signals (DH[0:31] and DP[0:3]) are used, and the data bus low and corresponding parity signals (DL[0:31]) and (DP[4:7]) are ignored. The following list summarizes the functionality of the 32-bit data bus mode on the MPC755:

- Data tenures of 1, 2, and 8 beats supported (1 to 4 bytes per beat).
- The address and transfer attribute information is unchanged from 64-bit mode.
- The $\overline{\text{TBST}}$ and TSIZ[0:2] signals must be reinterpreted for burst size.
- Data termination is the same for each data beat using $\overline{\text{TA}}$, $\overline{\text{DRTRY}}$, and $\overline{\text{TEA}}$.
- 32-bit mode configured at power-on (hard reset) through the $\overline{\text{TLBISYNC}}$ signal.

The 32-bit data bus mode operates the same as the 64-bit data bus mode with the exception of the byte lanes involved in the transfer and the number of data beats that are performed. Only byte lanes 0 through 3 are used, corresponding to the data bus signals DH[0:31] and DP[0:3]. Byte lanes 4 through 7 (corresponding to DL[0:31] and DP[4:7]) are never used in this mode. The unused data bus signals are not sampled by the processor during read operations, and they are driven low during write operations.

The number of data beats required for a data tenure in 32-bit data bus mode are one, two, or eight depending on the size of the transaction and the cache mode for the address. Data transactions of one or two data beats are performed for cache-inhibited load/store or write-through store operations. These transactions do not assert the $\overline{\text{TBST}}$ signal even though a two-beat burst may be performed (that is, the same $\overline{\text{TBST}}$ and TSIZ[0:2] encoding as in 64-bit data bus mode). Single-beat data transactions are performed for operations of size 4 bytes or less, and double-beat data transactions are performed for 8-byte operations only. (The processor only generates an 8-byte operation for a double-word aligned load or store-double operation to or from the floating-point registers.)

Data transactions of eight data beats are performed for burst operations that load into or cast out from the MPC755's internal caches. These transactions transfer 32 bytes similarly to 64-bit mode, and they assert the $\overline{\text{TBST}}$ signal and indicate a transfer size of two (TSIZ[0:2] = 010) similar to 64-bit data bus mode.

Otherwise, the same bus protocols apply for arbitration, transfer, and termination of the address and data tenures in 32-bit data bus mode as apply in 64-bit data bus mode. Late $\overline{\text{ARTRY}}$ cancellation of the data tenure applies on the bus clock cycle after the first data beat is acknowledged (after the first $\overline{\text{TA}}$) for word or smaller transactions, or on the bus clock cycle after the second data beat is acknowledged (after the second $\overline{\text{TA}}$) for double-word or burst operations (or coincident with the respective $\overline{\text{TA}}$ if no-$\overline{\text{DRTRY}}$ mode is selected).

8.4.1 Burst Ordering

For burst operations in 32-bit mode, a data block of 32-bytes (one cache line) is transferred in the same order as in 64-bit data bus mode with the exception that eight data beats are required to perform the transfer instead of four. For each double word of the block that is transferred, the upper word of the double word is transferred first on the data bus (on DH[0:31]), and then the lower word of the double word is transferred. Table 22 shows the burst order for each starting address.

Table 22. Burst Ordering

Data Transfer	For Double Word Starting Address:			
	A[27:28] = 00	A[27:28] = 01	A[27:28] = 10	A[27:28] = 11
1st Data Beat	DW0 - u	DW1 - u	DW2 - u	DW3 - u
2nd Data Beat	DW0 - l	DW1 - l	DW2 - l	DW3 - l
3rd Data Beat	DW1 - u	DW2 - u	DW3 - u	DW0 - u
4th Data Beat	DW1 - l	DW2 - l	DW3 - l	DW0 - l
5th Data Beat	DW2 - u	DW3 - u	DW0 - u	DW1 - u
6th Data Beat	DW2 - l	DW3 - l	DW0 - l	DW1 - l
7th Data Beat	DW3 - u	DW0 - u	DW1 - u	DW2 - u
8th Data Beat	DW3 - l	DW0 - l	DW1 - l	DW2 - l

Notes:

A[27:28] specifies the first double word of the 32-byte block being transferred; the remaining double words to transfer must wrap around the block.

A[29:31] are always 0b000 for burst transfers initiated by the MPC755.

"DWx" represents the double word that would be addressed by A[27:28] = "x" if a non-burst transfer were performed. "u" and "l" represent the upper word and lower word of the double word respectively.

Each data beat is terminated with one valid assertion of $\overline{\text{TA}}$ (without $\overline{\text{DRTRY}}$ cancellation).

8.4.2 Aligned Transfers

The aligned data transfer cases for 32-bit data bus mode are shown in Table 23. All of the transfers require a single data beat (if cache-inhibited or write-through) except for double-word cases that require two data beats. The double-word case is only generated by the processor for load or store-double operations to/from the floating-point registers. All cache-inhibited instruction fetches are performed as word operations.

Table 23. Aligned Data Transfers—32-Bit Data Bus Mode

Program Transfer Size	Bus TSIZ[0:2]	Bus A[29:31]	Data Bus Byte Lanes							
			DH0...		...DH31		DL0...		...DL31	
			B0	B1	B2	B3	B4	B5	B6	B7
Byte - 1 beat	0 0 1	0 0 0	A	—	—	—	x	x	x	x
Byte - 1 beat	0 0 1	0 0 1	—	A	—	—	x	x	x	x
Byte - 1 beat	0 0 1	0 1 0	—	—	A	—	x	x	x	x
Byte - 1 beat	0 0 1	0 1 1	—	—	—	A	x	x	x	x

Table 23. Aligned Data Transfers—32-Bit Data Bus Mode (Continued)

Program Transfer Size	Bus TSIZ[0:2]	Bus A[29:31]	Data Bus Byte Lanes							
			DH0...			...DH31	DL0...			...DL31
			B0	B1	B2	B3	B4	B5	B6	B7
Byte - 1 beat	0 0 1	1 0 0	A	—	—	—	x	x	x	x
Byte - 1 beat	0 0 1	1 0 1	—	A	—	—	x	x	x	x
Byte - 1 beat	0 0 1	1 1 0	—	—	A	—	x	x	x	x
Byte - 1 beat	0 0 1	1 1 1	—	—	—	A	x	x	x	x
Half-Word - 1 beat	0 1 0	0 0 0	A	A	—	—	x	x	x	x
Half-Word - 1 beat	0 1 0	0 1 0	—	—	A	A	x	x	x	x
Half-Word - 1 beat	0 1 0	1 0 0	A	A	—	—	x	x	x	x
Half-Word - 1 beat	0 1 0	1 1 0	—	—	A	A	x	x	x	x
Word - 1 beat	1 0 0	0 0 0	A	A	A	A	x	x	x	x
Word - 1 beat	1 0 0	1 0 0	A	A	A	A	x	x	x	x
Double Word - 1st beat	0 0 0	0 0 0	A	A	A	A	x	x	x	x
Double Word - 2nd beat			A	A	A	A	x	x	x	x

Notes:

“A” - byte lanes that are read or written during that bus transaction.

“—” - These lanes are ignored during read transactions and driven with undefined data during write transactions.

“x” byte lanes are not used in 32-bit data bus mode. They are not sampled by the MPC755 during reads and are driven low during writes.

8.4.3 Misaligned Data Transfers

Misaligned data transfer cases operate similarly in 32-bit data bus mode as in 64-bit data bus mode with the usual exception that only the DH[0:31] data bus is used. An example of a four-byte misaligned transfer starting at each possible byte address within a double word is shown in Table 24.

Table 24. Misaligned Data Transfers Example—32-Bit Data Bus Mode

Program Size of Word (4 bytes)	Bus TSIZ[0:2]	Bus A[29:31]	Data Bus Byte Lanes							
			DH0...			...DH31	DL0...			...DL31
			B0	B1	B2	B3	B4	B5	B6	B7
Aligned	1 0 0	0 0 0	A	A	A	A	x	x	x	x
Misaligned - 1st access	0 1 1	0 0 1	—	A	A	A	x	x	x	x
2nd Access	0 0 1	1 0 0	A	—	—	—	x	x	x	x
Misaligned - 1st access	0 1 0	0 1 0	—	—	A	A	x	x	x	x
2nd Access	0 1 0	1 0 0	A	A	—	—	x	x	x	x
Misaligned - 1st access	0 0 1	0 1 1	—	—	—	A	x	x	x	x
2nd Access	0 1 1	1 0 0	A	A	A	—	x	x	x	x

Table 24. Misaligned Data Transfers Example—32-Bit Data Bus Mode (Continued)

Program Size of Word (4 bytes)	Bus TSIZ[0:2]	Bus A[29:31]	Data Bus Byte Lanes							
			DH0...			...DH31	DL0...			...DL31
			B0	B1	B2	B3	B4	B5	B6	B7
Aligned	1 0 0	1 0 0	A	A	A	A	x	x	x	x
Misaligned - 1st access	0 1 1	1 0 1	—	A	A	A	x	x	x	x
2nd Access	0 0 1	0 0 0	A	—	—	—	x	x	x	x
Misaligned - 1st access	0 1 0	1 1 0	—	—	A	A	x	x	x	x
2nd Access	0 1 0	0 0 0	A	A	—	—	x	x	x	x
Misaligned - 1st access	0 0 1	1 1 1	—	—	—	A	x	x	x	x
2nd Access	0 1 1	0 0 0	A	A	A	—	x	x	x	x

Notes:
"A" - byte lane read in
"x" - ignored byte lane (does not need to be valid)

8.4.4 Selecting D32 Mode

The processor selects 64-bit or 32-bit data bus mode at power-up by sampling the state of the $\overline{\text{TLBISYNC}}$ signal at the negation of $\overline{\text{HRESET}}$ (coming out of hard reset). If the $\overline{\text{TLBISYNC}}$ signal is high (negated) at the negation of $\overline{\text{HRESET}}$, 64-bit data mode is selected. If $\overline{\text{TLBISYNC}}$ is low (asserted), 32-bit data mode is used.

For 32-bit systems not using the $\overline{\text{TLBISYNC}}$ signal, $\overline{\text{TLBISYNC}}$ can be connected to $\overline{\text{HRESET}}$ directly. Otherwise, it can be connected to a pull-up resistor to select 64-bit mode. For systems using the $\overline{\text{TLBISYNC}}$ input function, the state of $\overline{\text{HRESET}}$ must be logically combined in the $\overline{\text{TLBISYNC}}$ generation path to select the desired mode.

8.4.5 Signal Relationships

The signal relationships for 32-bit mode are the same as 64-bit mode. Figure 14 and Figure 15 show an example of an 8-beat burst transaction and a 2-beat burst transaction with $\overline{\text{DRTRY}}$, respectively.

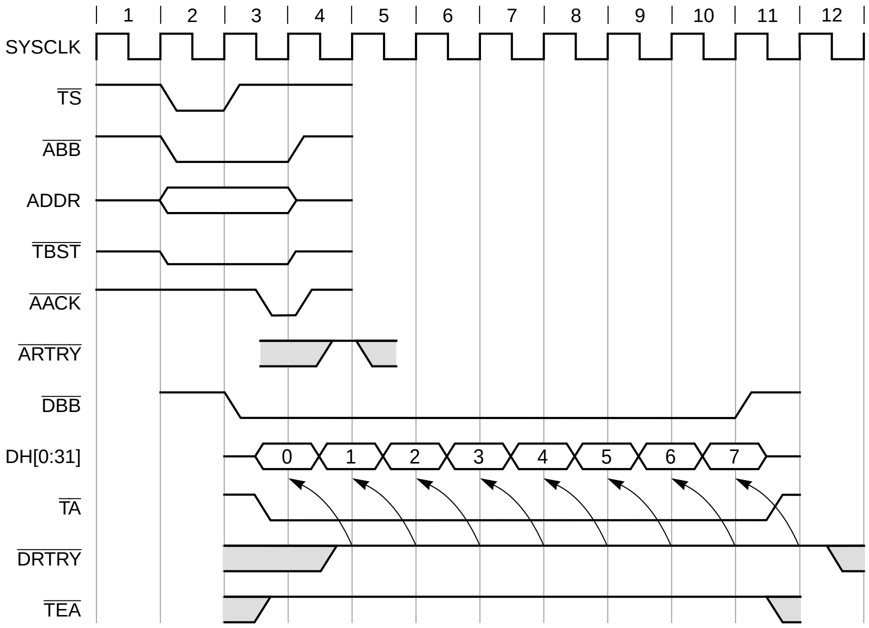


Figure 14. 32-Bit Data Bus Mode—8-Beat Burst (No Retry Conditions)

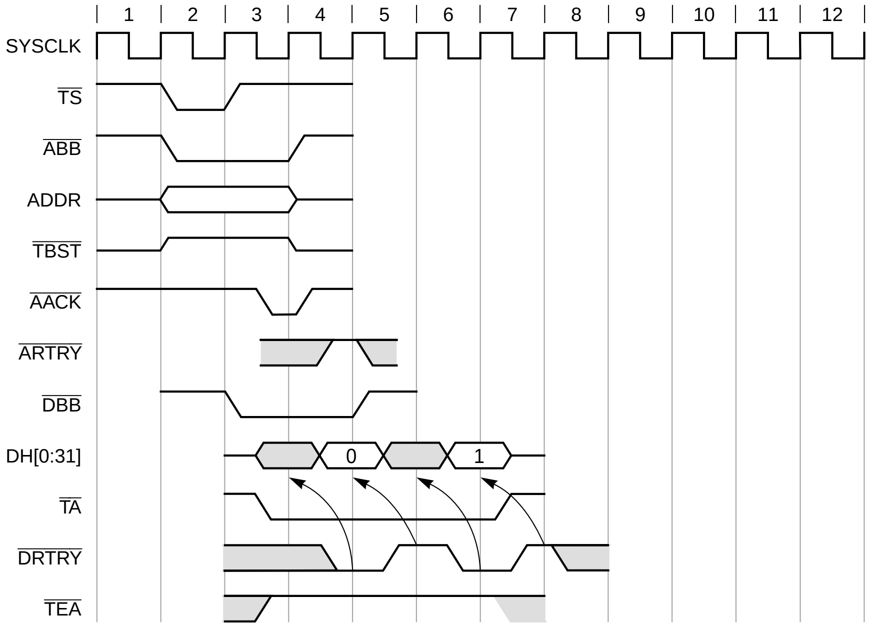


Figure 15. 32-Bit Data Bus Mode—2-Beat Burst (with $\overline{\text{DRTRY}}$)

Part IX MPC755 L2 Cache Interface Operation (Chapter 9)

This section describes the L2 cache interface operation of the MPC755, and how it differs from the MPC750.

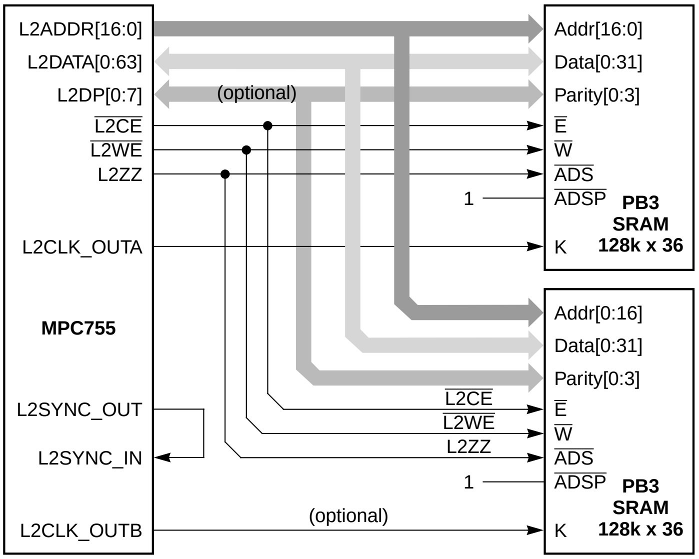
9.1 MPC755 L2 Cache Interface Overview

The MPC755's L2 cache is implemented with an on-chip, two-way set-associative tag memory, and with external synchronous SRAMs for data storage, similar to the MPC750. The external SRAMs are accessed through a dedicated L2 cache port which supports a single bank of up to 1 Mbyte of synchronous SRAMs. The L2 cache normally operates in copyback mode and supports system cache coherency through snooping. The differences from the MPC750 L2 cache interface are summarized as follows:

- Support for 4-1-1-1 PB3 synchronous burst-only SRAMS
- Additional control of the L2 interface during low-power operation
- Additional information about (and control of) the L2 DLL circuitry
- A new instruction-only mode
- Private memory capability for half or all of the L2 SRAM
- More flexible control of the L2 parity signals by allowing data or data and address parity

In addition to including the MPC755-specific information, this section supersedes Chapter 9, "L2 Cache Interface," in the *MPC750 User's Manual*.

Figure 16 shows a typical connection from the MPC755 processor L2 interface to a bank of PB3 SRAMS. See the *MC750 User's Manual* for typical connections to other SRAM technologies. Note that the signals for the L2 interface on the MPC755 are the same as those used for the MPC750.



Notes:

1. For a 1 Mbyte L2, use address bits 0–16 (bit 0 is LSB).
2. For a 512 Kbyte L2, use address bits 0–15 (bit 0 is LSB).
3. For a 256 Kbyte L2, use address bits 0–14 (bit 0 is LSB).
4. External clock routing should ensure that the rising edge of the L2 clock is coincident at the K input of all SRAMs and at the L2SYNC_IN input of the MPC755. The clock ‘A’ network only could be used, or the clock ‘B’ network could also be used depending on loading, frequency, and number of SRAMS.
5. No pull-up resistors are normally required for the L2 interface.
6. The MPC755 supports only one bank of SRAMS.
7. For high-speed operation, no more than two loads should be presented on each L2 interface signal.

Figure 16. Typical Synchronous 1–Mbyte L2 Cache System Using PB3 SRAM

9.1.1 L2 Cache Organization

The MPC750’s L2 cache interface is implemented with an on-chip, two-way set-associative tag memory with 4096 tags per way, and a dedicated interface with support for up to 1 Mbyte of external synchronous SRAM for data storage. The tags are sectored to support either two cache blocks per tag entry (two sectors, 64 bytes), or four cache blocks per tag entry (four sectors, 128 bytes) depending on the L2 cache size. If the L2 cache is configured for 256 Kbytes or 512 Kbytes of external SRAM, the tags are configured for two sectors per L2 cache block. The L2 tags are configured for four sectors per L2 cache block when 1 Mbyte of external SRAM is used. Each sector (32-byte L1 cache block) in the L2 cache has its own valid and modified bits and other status bits that implement the MEI cache coherency protocol.

Table 25 lists the data RAM organizations for the various L2 cache sizes. Table 25 also indicates typical SRAM sizes that might be used to construct such a cache.

Table 25. L2 Cache Sizes and Data RAM Organizations

L2 Cache Size	L2 Data Bus Size	L2 Data RAM Organization	Example SRAM Sizes
256 Kbytes	64/72 bit	32 Kbytes x 64/72	(2) 32 Kbytes x 32/36
512 Kbytes	64/72 bit	64 Kbytes x 64/72	(2) 64 Kbytes x 32/36
1 Mbyte	64/72 bit	128 Kbytes x 64/72	(2) 128 Kbytes x 32/36

Note:

The MPC755 supports only one bank of SRAMs.

For very high speed operation, no more than two SRAM devices should be used.

9.1.2 L2 Cache Control

The L2 cache control register (L2CR) allows control of L2 cache configuration and timing, byte-level data parity generation and checking, global invalidation of L2 contents, write-through operation, and L2 test support. The L2 cache interface provides two clock outputs that allow the clock inputs of the SRAMs to be driven at select frequency divisions of the processor core frequency. See the *MPC755 Hardware Specifications* for details about the specific frequency ratios supported. For more details about the L2CR, see Section 9.4.1, “L2 Cache Control Register (L2CR).”

9.1.3 L2 Private Memory

A portion, or all, of the L2 cache can alternately be used as a private SRAM. In this way, a portion of the physical address space can be mapped into a portion of the L2 SRAM. This functionality is described in Section 9.2.2, “L2 Private Memory Operation.” When private SRAM is used and the upper bits of the physical address match the L2PM[PMBA] field, the data is written or read from the private space of the L2 SRAM instead of external memory. Note that all of the SRAM can be designated as private, or for 512 Kbytes or 1 Mbyte SRAM, half can be designated as private and half as L2 cache. See Table 28 for all the supported combinations. Also, see Section 9.6.5, “Cache Control Instructions and Effect on Private Memory Operation,” for information on the operation of cache control instructions with respect to private memory space.

9.2 L2 Interface Operation

This section describes the general operation of both the L2 cache and the private memory capabilities of the L2 interface.

9.2.1 L2 Cache Operation

The MPC755’s L2 cache is a combined instruction and data cache that receives memory requests from both L1 instruction and data caches independently. The L1 requests are generally the result of instruction fetch misses, data load or store misses, write-through operations, or cache management instructions. Each L1 request generates an address lookup in the L2 tags. If a hit occurs, the instructions or data are forwarded to the L1 cache. A miss in the L2 tags causes the L1 request to be forwarded to the 60x bus interface. The cache block received from the bus is forwarded to the L1 cache immediately, and is also loaded into the L2 cache with the tag marked valid and unmodified. If the cache block loaded into the L2 causes a new tag entry to be allocated and the current tag entry is marked valid modified, the modified sectors of the tag to be replaced are cast out from the L2 cache to the 60x bus.

See Section 9.6.4, “Other Cache Control Instructions and Effect on L2 Cache,” for more information on the operation of cache control operations on the L2 cache.

9.2.1.1 L2 Cache Access Priorities

At any given time the L1 instruction cache may have one instruction fetch request, and the L1 data cache may have one load and two stores requesting L2 cache access. The L2 cache also services snoop requests from the 60x bus. When there are multiple pending requests to the L2 cache, snoop requests have highest priority, followed by data load and store requests (serviced on a first-in, first-out basis). Instruction fetch requests have the lowest priority in accessing the L2 cache when there are multiple accesses pending.

If read requests from both the L1 instruction and data caches are pending, the L2 cache can perform a hit-under-miss operations and supplies the available instruction or data while a bus transaction for the previous L2 cache miss is performed. The L2 cache does not support miss-under-miss, and the second instruction fetch or data load stalls until the bus operation resulting from the first L2 miss completes.

9.2.1.2 L2 Cache Services

All requests to the L2 cache that are marked cacheable (even if the respective L1 cache is disabled or locked) cause a tag lookup and will be serviced if the instructions or data are in the L2 cache. Burst requests from the L1 caches and single-beat read requests that hit in the L2 cache are forwarded the instructions or data, and the L2 LRU bit for that tag is updated. Burst writes from the L1 data cache due to a castout or replacement copyback are written only to the L2 cache, and the L2 cache sector is marked modified. Designers should note that during burst transfers into and out of the L2 cache SRAM array, an address is generated by the MPC755 for each data beat.

If the L2 cache is configured as write-through, the L2 sector is marked unmodified, and the write is forwarded to the 60x bus. If the L1 castout requires a new L2 tag entry to be allocated and the current tag is marked modified, any modified sectors of the tag to be replaced are cast out of the L2 cache to the 60x bus.

Single-beat reads that miss in the L2 cache do not cause any state changes in the L2 cache and are forwarded on the 60x bus interface. Cacheable single-beat store requests marked copyback that hit in the L2 are allowed to update the L2 cache sector, but do not cause L2 cache sector allocation or deallocation. Cacheable, single-beat store requests that miss in the L2 are forwarded to the 60x bus. Single-beat store requests marked write-through (through address translation or through the configuration of L2CR[L2WT]) are written to the L2 cache if they hit and are written to the 60x bus independent of the L2 hit/miss status. If the store hits in the L2 cache, the modified/unmodified status of the tag remains unchanged.

9.2.1.3 L2 Cache Coherency and WIMG Bits

Different from the MPC750, a request to the L2 cache on the MPC755 that is marked cache-inhibited by address translation (through either the MMU or by default WIMG configuration) will hit in the L2 cache if it has been previously loaded (and is still valid), causing a paradox condition. However, misses for cache-inhibited accesses do not cause a new entry to be allocated and do not cause any L2 cache tag state change.

9.2.1.4 Single-Beat Accesses to L2 Interface

The processor performs single-beat read and write accesses when the L1 instruction and/or data caches are disabled, and when the WIMG bit settings indicate that an area of memory is cache-inhibited (this case not forwarded to the L2 interface). Additionally, single-beat writes occur to the L2 interface when that area of memory is designated as write-through. PB2 SRAMs naturally support single-beat read and write accesses. However, the L2 interface requires 64-bit accesses to the SRAM. Therefore, for single-beat writes, the MPC750 and MPC755 automatically perform a read-modify-write operation in order to write the complete 64-bits to the L2.

PB3 SRAMs support bursting accesses only. Thus, for PB3 SRAMs, the L2 interface always automatically performs a burst read for a complete cache line from the SRAM. If a single-beat read was requested, then the appropriate double word is forwarded to the L1. Write accesses to PB3 SRAMs also require burst accesses. Thus for a single-beat write, the L2 interface automatically performs a burst read-modify-write in order to perform the complete write burst.

9.2.2 L2 Private Memory Operation

The L2 interface of the MPC755 can also be used as a low-latency, high-bandwidth private memory space. The private memory space is not snooped and is therefore not coherent with other processors in a system. The private space can contain instructions and data and its contents can be cached in the L1 instruction and data caches provided the accesses are marked cacheable.

The private memory receives requests from both the L1 instruction cache and the L1 data cache independently. The L1 requests are generally the result of instruction misses, data load or store misses, L1 data cache castouts, write-through operations, or cache management instructions. For all cacheable accesses, the L1 requests are looked-up in the L2 tags and compared with the corresponding PMBA bits of the L2PM. If a match occurs with L2PM[PMBA], the result of the L2 tag lookup is ignored and the request is forwarded to the external L2 SRAM interface as a private memory access. All transactions that read or write data, except those caused by the **eciwx** and **ecowx** instructions, are allowed to hit in the private memory space, regardless of the WIMG memory/cache attribute bits.

Transactions caused by the **icbi**, **sync**, **tlbie**, **tlbsync**, **eieio**, **eciwx**, and **ecowx** instructions never hit in the private memory space and are forwarded to the system interface. Accesses caused by the **dcbi** instruction that hit in the private memory space are discarded (after invalidating the L1 data cache). The private memory space does not have coherency state information. When the L1 data cache is reloaded for a cacheable load or store, the state will be exclusive or modified, respectively.

Generally, the private memory operates according to the following:

- Arbitration is shared with the L2 cache and thus uses the same priorities.
- All requests to the L2 interface that are marked cache-inhibited by address translation (WIMG bits) are allowed to hit in the private space. Cache-inhibited stores write the appropriate data to the L2 interface.
- Requests to the L2 interface that are marked cacheable by address translation (even if the respective L1 cache is locked) are serviced by the L2 interface if they map to the private memory space.
- Burst read and single-beat read requests from the L1 instruction or data caches that map to the private memory space are forwarded data from the L2 SRAMs designated as private memory.
- Burst read requests from the L1 instruction or data caches that do not map to private memory space (and miss in the L2 cache, if enabled) initiate a burst read operation from the system interface for the cache line that missed. The cache line received from the bus is forwarded to the appropriate L1 cache (and the L2 cache, if enabled).
- Normal burst writes from the L1 data cache due to castouts (also referred to as replacement copybacks) that map to the private memory space are written to the external SRAMs designated as private memory regardless of the L2CR[L2IO] setting. Burst writes that don’t map to the private memory space are allocated in the L2 cache (if enabled).

Note that software-generated single-beat reads and writes directed to the private memory SRAMs are handled in the same way as described for the SRAMs as L2 cache, and read-modify-write transactions are performed automatically by the L2 controller as needed as described in Section 9.2.1.4, “Single-Beat Accesses to L2 Interface.”

See Section 9.6.4, “Other Cache Control Instructions and Effect on L2 Cache,” for more information on the operation of cache control operations on the L2 cache. However, the following apply to the private memory space:

- Cacheable **stwcx.** operations are handled by the L1 data cache similarly to normal cacheable stores. The L2 interface does not treat **stwcx.** differently than a normal cacheable store. Cache-inhibited **stwcx.** accesses that hit in the private memory space write the appropriate data to the L2 interface and are not forwarded to the system interface.
- **dcbz** operations that hit in the private memory space do not affect the data in the external SRAMs. They are handled entirely by the L1.
- **dcbf** operations are issued to the L2 interface after being processed by the L1 data cache. If a **dcbf** that hits in L1 data cache and requires a line push hits in the private memory space, the cache line is written to the L2 interface. **dcbf** operations that hit in the private memory space are never forwarded to the system interface.
- **dcbst** instructions are issued to the L2 cache after being processed by the L1 data cache. If a **dcbst** that hits in the L1 data cache and requires a line push hits in the private memory space, the cache line is written to the external SRAMs. **dcbst** operations that hit in the private memory space are never forwarded to the system interface.

- **dcbi** instructions that hit in the private memory space are discarded and are never forwarded to the system interface.
- **icbi** instructions never affect the L2 interface and are just passed to the system interface for further processing.
- **sync**, **eieio**, **eciwx**, **ecowx**, **tlbie**, and **tlbsync** instructions pass though the L2 interface and are forwarded to the system interface for further processing.

Note that L2 cache-related performance monitor events may not produce expected results when L2 private memory is enabled. Specifically, hits to the private memory are treated as L2 cache misses by the performance monitor. No new performance monitor events have been added to specifically support the L2 private memory.

9.3 L2 Clocking

The MPC755 generates the clock for the external L2 synchronous data RAMs in the same way as the MPC750. The clock frequency for the RAMs is divided down from the core clock frequency of the MPC755. The divided-down clock is then phase-adjusted by an on-chip delay-lock loop (DLL) circuit, sent out from the MPC755 to the external RAMs, and then returned as an input to the DLL so that the rising-edge of the clock as seen at the external RAMs can be aligned to the clocking of the internal latches in the MPC755’s L2 bus interface.

The core-to-L2 frequency divisor for the L2 PLL is selected through the L2CLK bits of the L2CR register. Generally, the divisor must be chosen according to the frequency supported by the external RAMs, the internal core operating frequency, and the phase adjustment range that the L2 DLL supports. The L2 RAM frequency can be divided down from the core operating frequency as described in the *MPC755 Hardware Specification*. Additional supported frequency ratios for the MPC755 are also highlighted in the hardware specification.

9.4 L2 Registers

This section describes the cache configuration bits in the L2 cache control register (L2CR) and the L2 cache private memory control register (L2PM).

9.4.1 L2 Cache Control Register (L2CR)

The L2 cache control register of the MPC755 is a read/write, supervisor-level, implementation-specific SPR used to configure and operate the L2 cache, and it is slightly different from the L2CR of the MPC750. The differences are summarized as follows:

- New encoding for L2RAM field defined for PB3 SRAM support
- More output hold options defined for L2OH field
- New L2CR bit for instruction-only mode—L2IO
- New L2CR fields defined for low-power operation and DLL control—L2CS, L2DRO, and L2CTR

The L2CR register can be accessed with the **mtspr** and **mfspr** instructions using SPR 1017 (decimal). Note that all bits of L2CR are cleared by a hard reset and on power-on reset. Figure 17 shows the bits of the L2CR.

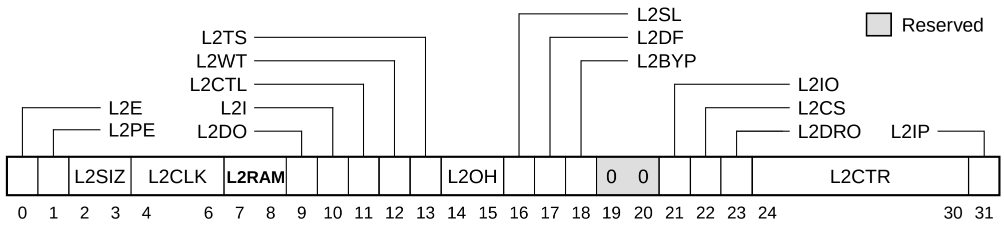


Figure 17. L2 Cache Control Register (L2CR)

The L2CR bits for the MPC755 are described in Table 26.

Table 26. L2 Cache Control Register

Bit	Name	Description
0	L2E	L2 enable. Enables L2 cache operation (including snooping) starting with the next transaction the L2 cache unit receives. Before enabling the L2 cache, the L2 clock must be configured through L2CR[2CLK], and the L2 DLL must stabilize (see the MPC755 Hardware Specifications) and all other L2CR bits must be set appropriately. The L2 cache may need to be globally invalidated.
1	L2PE	L2 data parity generation and checking enable. Enables parity generation and checking for the L2 data RAM interface. When disabled, generated parity is always zeros. Note that the L2 interface always generates and drives parity on the L2DP[0:7] signals for writes to the SRAM array.
2–3	L2SIZ	L2 size. Should be set according to the size of the L2 data RAMs used. A 256-Kbyte L2 cache requires a data RAM configuration of 32 Kbytes x 64 bits; a 512-Kbyte L2 cache requires a configuration of 64 Kbyte x 64 bits; a 1-Mbyte L2 cache requires a configuration of 128 Kbytes x 64 bits. 00 Reserved 01 256 Kbyte 10 512 Kbyte 11 1 Mbyte
4–6	L2CLK	L2 clock ratio (core-to-L2 frequency divider). Specifies the clock divider ratio between the core clock frequency and the L2 data RAM interface. When these bits are cleared, the L2 clock is stopped and the on-chip DLL for the L2 interface is disabled. For non-zero values, the processor generates the L2 clock and the on-chip DLL is enabled. After the L2 clock ratio is chosen, the DLL must stabilize before the L2 interface can be enabled (see the MPC755 Hardware Specifications). The resulting L2 clock frequency cannot be slower than the clock frequency of the 60x bus interface. 000 L2 clock and DLL disabled 001 ÷1 010 ÷1.5 011 Reserved 100 ÷2 101 ÷2.5 110 ÷3 111 Reserved

Table 26. L2 Cache Control Register (Continued)

Bit	Name	Description
7–8	L2RAM	L2 RAM type—Configures the L2 interface for the type of synchronous SRAMs used: • Flow-through (register-buffer) synchronous burst SRAMs that clock addresses in and flow data out • Pipelined (register-register) PB2 synchronous burst SRAMs that clock addresses in and clock data out (with 3-1-1-1 access times) • Pipelined (register-register) PB3 synchronous burst SRAMs (with 4-1-1-1 access times) • Late-write synchronous SRAMs, for which the MPC755 requires a pipelined (register-register) configuration. Late-write RAMs require write data to be valid on the cycle after $\overline{\text{WE}}$ is asserted rather than on the same cycle as the write enable (as required with traditional burst RAMs). For the PB2 burst RAM selection, the MPC755 does not burst data into the L2 cache; it generates an address for each access. However, for the PB3 burst RAM selection, the MPC755 does burst data into the L2 cache. If the SRAMs or part of the SRAM is configured as an L2 cache, the L1 caches should be enabled for data to be efficiently loaded into the L2 cache for all types of SRAMs; otherwise, significant latencies are incurred. If all the L2 SRAM cache is configured as private memory, disabled L1 instruction and data caches do not affect the L2 latencies. Pipelined SRAMs may be used for all L2 clock modes. Note that flow-through SRAMs can be used only for L2 clock modes that are divide-by-2 or slower (divide-by-1 and divide-by-1.5 not allowed). 00 Flow-through (register-buffer) synchronous burst SRAM 01 Pipelined (register-register) PB3 synchronous burst SRAM 10 Pipelined (register-register) PB2 synchronous burst SRAM 11 Pipelined (register-register) synchronous late-write SRAM
9	L2DO	L2 data-only. Setting this bit enables data-only operation in the L2 cache. For this operation, instruction transactions from the L1 instruction cache already cached in the L2 cache can hit in the L2, but new instruction transactions from the L1 instruction cache are treated as cache-inhibited (bypass L2 cache, no L2 checking done). When both L2DO and L2IO are set, the L2 cache is effectively locked (cache misses do not cause new entries to be allocated but write hits use the L2).
10	L2I	L2 global invalidate. Setting L2I invalidates the L2 cache globally by clearing the L2 bits including status bits. This bit must not be set while the L2 cache is enabled.
11	L2CTL	L2 RAM control (ZZ enable). Setting L2CTL enables the automatic operation of the L2ZZ (low-power mode) signal for cache RAMs that support the ZZ function (PB2 RAMs). If L2CTL is set, L2ZZ asserts automatically when the MPC755 enters nap or sleep mode and negates automatically when the MPC755 exits nap or sleep mode. The use of this bit is not recommended for future compatibility. This bit should not be set when the MPC755 is in nap mode and snooping is to be performed through the negation of $\overline{\text{QACK}}$. Additionally, it should not be set when using PB3 SRAMs.
12	L2WT	L2 write-through. Setting L2WT selects write-through mode (rather than the default write-back mode) so all writes to the L2 cache also write through to the 60x bus. For these writes, the L2 cache entry is always marked as exclusive rather than modified. This bit must never be set after the L2 cache has been enabled because previously-modified lines could get re-marked as exclusive during normal operation.
13	L2TS	L2 test support. Setting L2TS causes cache block pushes from the L1 data cache that result from **dcbf** and **dcbst** instructions to be written only into the L2 cache and marked valid, rather than being written only to the 60x bus and marked invalid in the L2 cache in case of a hit. This bit allows a **dcbz**/**dcbf** instruction sequence to be used with the L1 cache enabled to easily initialize the L2 cache with any address and data information. This bit also keeps **dcbz** instructions from being broadcast on the 60x bus and single-beat cacheable store misses in the L2 from being written to the 60x bus.

Table 26. L2 Cache Control Register (Continued)

Bit	Name	Description
14–15	L2OH	L2 output hold. These bits configure output hold time for address, data, and control signals driven by the MPC755 to the L2 data RAMs. They should generally be set according to the SRAM's input hold time requirements, for which late-write SRAMs usually differ from flow-through or burst SRAMs. See the MPC755 Hardware Specification for the actual recommended values. 00 Least hold time 01 More hold time 10 Even more hold time 11 Most output hold time
16	L2SL	L2 DLL slow. Setting L2SL increases the delay of each tap of the DLL delay line. It is intended to increase the delay through the DLL to accommodate slower L2 RAM bus frequencies. Generally, L2SL should be set if the L2 RAM interface is operated below 100 MHz.
17	L2DF	L2 differential clock. Setting L2DF configures the two clock-out signals (L2CLK_OUTA and L2CLK_OUTB) of the L2 interface to operate as one differential clock. In this mode, the B clock is driven as the logical complement of the A clock. This mode supports the differential clock requirements of late-write SRAMs. Generally, this bit should be set when late-write SRAMs are used.
18	L2BYP	L2 DLL bypass. The DLL unit receives three input clocks: • A square-wave clock from the PLL unit to phase adjust and export • A non-square-wave clock for the internal phase reference • A feedback clock (L2SYNC_IN) for the external phase reference. Setting L2BYP causes the non-square wave clock (#2) to be used for both phase adjust and phase reference (#1 and #2), thus bypassing the square wave clock from the PLL. (Note that clock #2 is the actual clock used by the registers of the L2 interface circuitry.) L2BYP is intended for use when the PLL is being bypassed. If the PLL is being bypassed, the DLL must be operated in 1:1 mode and SYSCLK must be fast enough for the DLL to support.
19–20	—	Reserved. These bits are implemented but not used; keep at 0 for future compatibility.
21	L2IO	L2 instruction-only. Setting this bit enables instruction-only operation in the L2 cache. For this operation, data transactions from the L1 data cache already cached in the L2 cache can hit in the L2 (including writes), but new data transactions (transactions that miss in the L2) from the L1 data cache are treated as cache-inhibited (bypass L2 cache, no L2 checking done). When both L2DO and L2IO are set, the L2 cache is effectively locked (cache misses do not cause new entries to be allocated but write hits use the L2). Note that this bit can be programmed dynamically.
22	L2CS	L2 clock stop. Setting this bit causes the L2 clocks to the SRAMs to automatically stop whenever the MPC755 enters nap or sleep modes, and automatically restart when exiting those modes (including for snooping during nap mode). It operates by asynchronously gating off the L2CLK_OUT[A:B] signals while in nap or sleep mode. The L2 SYNC_OUT/SYNC_IN path remains in operation, keeping the DLL synchronized. This bit is provided as a power-saving alternative to the L2CTL bit and its corresponding ZZ pin, which may not be useful for dynamic stopping/restarting of the L2 interface from nap and sleep modes due to the relatively long recovery time from ZZ negation that many SRAM vendors require.
23	L2DRO	L2 DLL rollover. Setting this bit enables a potential rollover (or actual rollover) condition of the DLL to cause a checkstop for the processor. A potential rollover condition occurs when the DLL is selecting the last tap of the delay line, and thus may risk rolling over to the first tap with one adjustment while in the process of keeping synchronized. Such a condition is improper operation for the DLL, and, while this condition is not expected, it allows detection for added security. This bit should be set when the DLL is first enabled (set with the L2CLK bits) to detect rollover during initial synchronization. It could also be set when the L2 cache is enabled (with L2E bit) after the DLL has achieved its initial lock.

Table 26. L2 Cache Control Register (Continued)

Bit	Name	Description
24–30	L2CTR	L2 DLL counter (read-only). These bits indicate the current value of the DLL counter (0 to 127). They are asynchronously read when the L2CR is read, and as such, should be read at least twice with the same value in case the value is asynchronously caught in transition. These bits are intended to provide observability of where in the 128-bit delay chain the DLL is at any given time. Generally, the DLL operation should be considered at risk if it is found to be within a couple of taps of its beginning or end point (tap 0 or tap 128).
31	L2IP	L2 global invalidate in progress (read only). This read-only bit indicates whether an L2 global invalidate is occurring. It should be monitored after an L2 global invalidate has been initiated by the L2I bit to determine when it has completed.

9.4.2 L2 Private Memory Control Register (L2PM)

The L2 private memory control register is a new register in the MPC755 that allows a portion of the physical address space to be mapped into a portion of the L2 SRAM. It is a read/write, supervisor-level, implementation-specific register (SPR) which is accessed with the **mtspr** and **mfspr** instructions using SPR 1016 (decimal). Note that all bits of L2PM are cleared by a hard reset or power-on reset. Figure 18 shows the bits of the L2PM.

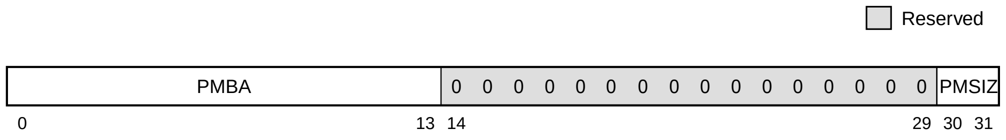


Figure 18. L2 Private Memory Control Register (L2PM)

The L2PM bits are described in Table 27.

Table 27. L2PM Bit Settings

Bit	Name	Description
0–13	PMBA	Private memory base address. If the upper bits of the physical address match the PMBA, the data is written or read from the private memory space of the L2 SRAM instead of external memory. 0–11 for 1 Mbyte 0–12 for 512 Kbytes 0–13 for 256 Kbytes
14–29	—	Reserved
30–31	PMSIZ	Private memory size. These bits along with the L2SIZ bits of the L2CR determine the amount of the L2 cache that is used as private memory space. See Table 28 for the L2 SRAM configurations. 00 = Private memory disabled 01 = 256 Kbytes 10 = 512 Kbytes 11 = 1 Mbyte

Table 28 describes the combinations possible (and the required bit settings) for using some or all of the L2 SRAM as private memory.

Table 28. L2 SRAM Configuration

Total L2 SRAM	Configured only as L2 Cache	Configured as 1/2 L2 Cache and 1/2 Private Memory	Configured only as Private Memory

256KB	L2E = 1
	L2SIZ = 01 (256 Kbytes)
	PMSIZ = 00 (disabled)
		not available	L2E = 0
			L2SIZ = don’t care
			PMSIZ = 01 (256 Kbytes)

512KB	L2E = 1
	L2SIZ = 10 (512 Kbytes)
	PMSIZ = 00 (disabled)
		L2E =1
		L2SIZ = 01 (256 Kbytes)
		PMSIZ = 01 (256 Kbytes)
			L2E = 0
			L2SIZ = don’t care
			PMSIZ = 10 (512 Kbytes)

1M	L2E = 1
	L2SIZ = 11 (1 Mbyte)
	PMSIZ = 00 (disabled)
		L2E =1
		L2SIZ = 10 (512 Kbytes)
		PMSIZ = 10 (512 Kbytes)
			L2E = 0
			L2SIZ = don’t care
			PMSIZ = 11 (1 Mbyte)

9.5 L2 Address and Data Parity Signals

The L2 parity signals (L2DP[0:7]) can be generated and checked by setting the L2PE bit in the L2CR. The parity bits are generated and checked using the corresponding L2DATA signals, and represent odd parity. If the L2AP_EN bit in HID2 is also set, the L2ADDR signals are also included in the parity generation and checking (again, representing odd parity) on the MPC755. Table 29 lists the association between L2DP[0:7] signals and the L2DATA and L2ADDR signals.

Table 29. L2 Data Parity Signal Associations

Signal	L2AP_EN = 0
	L2PE = 1
		L2AP_EN = 1
		L2PE = 1

L2DP0	L2DATA[0:7]	L2DATA[0:7], L2ADDR[0:2]
L2DP1	L2DATA[8:15]	L2DATA[8:15], L2ADDR[3:4]
L2DP2	L2DATA[16:23]	L2DATA[16:23], L2ADDR[5:6]
L2DP3	L2DATA[24:31]	L2DATA[24:31], L2ADDR[7:8]
L2DP4	L2DATA[32:39]	L2DATA[32:39], L2ADDR[9:10]
L2DP5	L2DATA[40:47]	L2DATA[40:47], L2ADDR[11:12]
L2DP6	L2DATA[48:55]	L2DATA[48:55], L2ADDR[13:14]
L2DP7	L2DATA[56:63]	L2DATA[56:63], L2ADDR[15:16]

9.6 L2 Cache Programming Considerations

This section describes some of the programming considerations for controlling the L2 cache and the effect of other cache control instructions on the L2 cache.

9.6.1 Enabling and Disabling the L2 Cache

Following a power-on or hard reset, the L2 cache and the L2 DLL are disabled initially. Before enabling the L2 cache, the L2 DLL must first be configured through the L2CR register, and the DLL must be allowed sufficient time (see the *MPC755 Hardware Specifications*) to achieve phase lock. Before enabling the L2 cache, other configuration parameters must be set in the L2CR, and the L2 tags must be globally invalidated. The L2 cache should be initialized during system start-up.

The sequence for initializing the L2 cache is as follows:

- Power-on reset (automatically performed by the assertion of $\overline{\text{HRESET}}$ signal).
- Disable L2 cache by clearing L2CR[L2E].
- Set the L2CR[L2CLK] bits to the desired clock divider setting. Setting a non-zero value automatically enables the DLL. All other L2 cache configuration bits should be set to properly configure the L2 cache interface for the SRAM type, size, and interface timing required.
- Wait for the L2 DLL to achieve phase lock. This can be timed by setting the decrementer for a time period equal to 640 L2 clocks, or by performing an L2 global invalidate.
- Perform an L2 global invalidate. The global invalidate could be performed before enabling the DLL, or in parallel with waiting for the DLL to stabilize. Refer to Section 9.6.2, "L2 Cache Global Invalidation," for more information about L2 cache global invalidation. Note that a global invalidate always takes much longer than it takes for the DLL to stabilize.
- After the DLL stabilizes, an L2 global invalidate has been performed, and the other L2 configuration bits have been set, enable the L2 cache for normal operation by setting the L2CR[L2E] bit to 1.

Note that if the L1 data cache is disabled and the L2 cache is enabled, hits in the L2 work correctly and update the L2. However, no new entries are allocated into the L2 because when the L1 data cache is disabled, the processor only performs single-beat accesses. Thus, these accesses all propagate to the 60x bus interface (the L2 only stores and allocates entries for burst accesses).

Before the L2 cache is disabled it must be flushed to prevent coherency problems. Note that the cache management instructions **dcbf**, **dcbst**, and **dcbi** do not affect the L1 data cache or L2 cache when they are disabled.

9.6.2 L2 Cache Global Invalidation

The L2 cache supports a global invalidation function in which all bits of the L2 tags (tag data bits, tag status bits, and LRU bit) are cleared. It is performed by an on-chip hardware state machine that sequentially cycles through the L2 tags. The global invalidation function is controlled through L2CR[L2I], and it must be performed only while the L2 cache is disabled. The MPC755 can continue operation during a global invalidation provided the L2 cache has been properly disabled before the global invalidation operation starts. Note that the MPC755 must be operating at full power (low power modes disabled) in order to perform L2 cache invalidation.

The sequence for performing a global invalidation of the L2 cache is as follows:

- Clear HID0[DPM] bit to zero. Dynamic power management must be disabled.
- Execute a **sync** instruction to finish any pending store operations in the load/store unit, disable the L2 cache by clearing L2CR[L2E], and execute an additional **sync** instruction after disabling the L2 cache to ensure that any pending operations in the L2 cache unit have completed.
- Initiate the global invalidation operation by setting the L2CR[L2I] bit to 1.
- Monitor the L2CR[L2IP] bit to determine when the global invalidation operation is completed (indicated by the clearing of L2CR[L2IP]). The global invalidation requires approximately 32K core clock cycles to complete.
- After detecting the clearing of L2CR[L2IP], clear L2CR[L2I] and re-enable the L2 cache for normal operation by setting L2CR[L2E]. Also, dynamic power management can be enabled at this time.

9.6.3 L2 Cache Flushing

L1 cache-block-push operations generated by the execution of **dcbf** and **dcbst** instructions write through to the 60x bus interface and invalidate the L2 cache sector if they hit. The execution of **dcbf** and **dcbst** instructions that do not cause a cache-block-push from the L1 cache are forwarded to the L2 cache to perform a sector invalidation and/or push from the L2 cache to the 60x bus as required. If the **dcbf** and **dcbst** instructions do not cause a sector push from the L2 cache, they are forwarded to the 60x bus interface for address-only broadcast if HID0[ABE] is set to 1.

9.6.4 Other Cache Control Instructions and Effect on L2 Cache

The execution of the **stwcx.** instruction results in single-beat writes from the L1 data cache. These single-beat writes are processed by the L2 cache according to hit/miss status, L1 and L2 write-through configuration, and reservation-active status. If the address associated with the **stwcx.** instruction misses in the L2 cache or if the reservation is no longer active, the **stwcx.** instruction bypasses the L2 cache and is forwarded to the 60x bus interface. If the **stwcx.** hits in the L2 cache and the reservation is still active, one of the following actions occurs:

- If the **stwcx.** hits a modified sector in the L2 cache (independent of write-through status), or if the **stwcx.** hits both the L1 and L2 caches in copy-back mode, the **stwcx.** is written to the L2 and the reservation completes.
- If the **stwcx.** hits an unmodified sector in the L2 cache, and either the L1 or L2 is in write-through mode, the **stwcx.** is forwarded to the 60x bus interface and the sector hit in the L2 cache is invalidated.

The **dcbi** instruction is always forwarded to the L2 cache and causes a segment invalidation if a hit occurs. The **dcbi** instruction is also forwarded to the 60x bus interface for broadcast if HID0[ABE] is set to 1. The **icbi** instruction invalidates only L1 cache blocks and is never forwarded to the L2 cache. Any **dcbz** instructions marked global do not affect the L2 cache state. If a **dcbz** instruction hits in the L1 and L2 caches, the L1 data cache block is cleared and the **dcbz** instruction completes. If a **dcbz** instruction misses in the L2 cache, it is forwarded to the 60x bus interface for broadcast. Any **dcbz** instructions that are marked nonglobal act only on the L1 data cache. Note that the **dcbz** instruction on the MPC755 must be preceded by a **dcbf** instruction to that address.

The **sync** and **eieio** instructions bypass the L2 cache and are forwarded to the 60x bus.

9.6.5 Cache Control Instructions and Effect on Private Memory Operation

When private memory is used as all or part of the L2 interface, cache control instructions function as follows:

- Cacheable **stwcx.** operations are handled by the L1 data cache similarly to normal cacheable stores. The L2 interface does not treat **stwcx.** differently than a normal cacheable store. Cache-inhibited **stwcx.** accesses that hit in the private memory space write the appropriate data to the L2 interface and are not forwarded to the system interface.
- **dcbz** operations that hit in the private memory space do not affect the data in the external SRAMs. They are handled entirely by the L1.
- **dcbf** operations are issued to the L2 interface after being processed by the L1 data cache. If a **dcbf** that hits in L1 data cache and requires a line push hits in the private memory space, the cache line is written to the L2 interface. **dcbf** operations that hit in the private memory space are never forwarded to the system interface.
- **dcbst** instructions are issued to the L2 cache after being processed by the L1 data cache. If a **dcbst** that hits in the L1 data cache and requires a line push hits in the private memory space, the cache line is written to the external SRAMs. **dcbst** operations that hit in the private memory space are never forwarded to the system interface.
- **dcbi** instructions that hit in the private memory space are discarded and are never forwarded to the system interface.
- **icbi** instructions never affect the L2 interface and are just passed to the system interface for further processing.
- **sync**, **eieio**, **eciwx**, **ecowx**, **tlbie**, and **tlbsync** instructions pass though the L2 interface and are forwarded to the system interface for further processing.

9.6.6 L2 Cache Testing

Several features are provided to facilitate testing of the L2 cache. The *MPC750 User's Manual* supplied some incorrect recommended procedures for testing the L2 cache. This section contains a corrected L2 cache test description that applies for both the MPC750 and the MPC755.

A typical test for verifying the proper operation of the MPC755's L2 cache memory (external SRAM and tag) performs the following steps:

1. Initialize the test sequence by disabling address translation to invoke the default WIMG setting of 0b0011.
2. Set the L2CR[L2DO] and L2CR[L2TS] bits and perform a global invalidation of the L1 data cache and the L2 cache. The L1 instruction cache can remain enabled to improve execution efficiency.

After initialization of the test sequence is complete, the L2 cache external SRAM may be tested using the following procedure:

1. Enable the L2 cache and the L1 data cache. Caches should have been invalidated during the initialization step.
2. Execute a series of **dcbz**, **stw**, and **dcbf** instructions to initialize the cache with a sequential range of addresses and with cache data consisting of zeroes.
3. Disable the L1 data cache.
4. Initialize the performance monitor counters to zero, and enable counting of L2 hits in the appropriate MMCR register. Refer to Chapter 11, “Performance Monitor,” of the *MPC750 User’s Manual* for complete details on using the performance monitors.
5. Perform a series of single-beat load and store operations using a variety of non-zero bit patterns to test for stuck bits and pattern sensitivities in the L2 cache SRAM. These loads and stores should be in the range of addresses used to initialize the caches in step 2 so that each access will hit in the L2 cache.
6. Disable the performance monitor counters, and read the value for the L2 cache hits. Verify that this result matches the accesses performed by the test routine.

A complete L2 cache test should test the tag memory as well as the SRAMs. Each bit of tag memory should be tested by loading the cache tags with data consisting of all 0's in one way of the cache and all 1's in the other way. Then, a series of accesses should be performed, walking a one or zero through the upper address bits to test for stuck bits and pattern sensitivities in the tag.

The number of tag bits used by the cache depends on the size of the cache. On the MPC750 and the MPC755, a 256-Kbyte cache uses 15 tag bits, a 512-Kbyte cache uses 14 tag bits, and a 1-Mbyte cache uses 13 tag bits.

For example, to test all the tag bits of a 512-Kbyte cache, a test program needs to do the following:

- Initialize the test sequence by disabling address translation to invoke the default WIMG bit settings of 0b0011.
- Set the L2CR[L2DO] and L2CR[L2TS] bits and perform an invalidation of the L1 data cache and the L2 cache. The L1 instruction cache may remain enabled for efficiency.
- Enable the L2 cache and the L1 data cache.

- Perform a series of **dcbz**, **stw**, and **dcbf** operations to fill the cache with unique data. Fill way 0 of the tag with data consisting of all zeroes, and fill way 1 of the tag with data consisting of all ones. The following pseudocode illustrates this procedure:

```
cache_size = (512 * 1024)       // 512 Kbyte
cache_line_size = 32            // 32 byte cache line size for 750
tag_bits = 14                   // for 512 Kbyte cache
r10 = 0x00000000                // all zeroes in upper tag_bits bits
r11 = 0xFFFc0000                // all ones in upper tag_bits bits
r12 = 0                         // index

for(i = 0; i < (1/2 cache_size / cache_line_size); i++)
{
    dcbz    r10,r12             // zero out line in L1 dcache
    add     r13,r10,r12         // create unique data
    stwx    r13,r10,r12         // store unique data in newly
                                // allocated L1 cache entry
    dcbf    r10,r12             // push data to L2 cache WAY 0
    dcbz    r11,r12             // zero out line in L1 dcache
    add     r13,r11,r12         // create unique data
    stwx    r13,r11,r12         // store unique data in newly
                                // allocated L1 cache entry
    dcbf    r11,r12             // push data to L2 cache WAY 1
    r12 += cache_line_size      // go to next cache line and repeat
}
```

- Disable the L1 data cache.
- Read back the data just written and verify its correctness. Use the performance monitors to count load hits in the L2 to verify that the data came from the L2. The number of hits should equal the number of loads.
- Attempt a series of loads from the cache with addresses that should not be in the tag by walking a one through the upper tag bits:

```
    r15 = 0x80000000            // address with a 1 in the top bit
    for(i = 0; i < tag_bits; i++)
{
initialize/enable the performance monitor counters to count load hits
    r12 = 0x00000000            // index
    for(j = 0; j < (1/2 cache_size / cache_line_size); j++)
{
    lwzx    r13,r15,r12         // attempt to load data
    r12 += cache_line_size      // go to the next cache line
}
disable the performance monitors, check to ensure that there were no hits
    r15 = r15 >> 1              // shift the one bit
                                // to the right for the next iteration
}
```

- Then perform a similar series of loads, this time by walking a zero through a series of addresses with ones in the upper tag bits. The first iteration of the inner loop above uses the start address 0x7FFC_0000, the second iteration uses the start address 0xBFFC_0000, the third 0xDFF_C000, and so on for each tag bit for the case of a 512-Kbyte cache. If there are any load hits at any point in the loop, there is a faulty tag in the cache.
- Repeat the entire process, this time with all ones in the way 0 tag entries, and all zeroes in the way 1 tag entries. (r10 = 0xFFFC_0000 and r11 = 0x0000_0000 in the pseudocode for the fourth step above for a 512-Kbyte cache.)

Caution: For these L2 cache tests, instruction translation is disabled and the L1 instruction cache is enabled. This means that WIMG defaults to 0b0011. Even though the L2 cache is in data-only mode, if an address in the L2 matches an instruction access, the L2 will hit and provide data for that access.

Therefore, cache test programs should avoid loading the L2 with address ranges that match the memory location of the test code. Otherwise, instruction accesses will hit on test data and cause random program behavior. For the test procedure described here, the test program should be located outside the address ranges 0x0000_0000 + cache_size and 0xFFFF_FFFF - cache_size.

The entire L2 cache may be tested by clearing L2CR[L2DO] and L2CR[L2TS], restoring the L1 and L2 caches to their normal operational state, and executing a comprehensive test program designed to exercise all the caches. The test program should include operations that cause L2 hit, reload, and castout activity that can be subsequently verified through the performance monitor.

Most of the tests described in this section only use the performance monitors to verify the number of cache hits that occurred during the test. While the performance monitors also provide facilities for counting L2 cache misses, this facility is only useful for counting L2 cache misses that cause burst reads to memory to occur. With the L1 data cache disabled and the L2CR[L2TS] bit set, all accesses are single-beat and therefore are not counted by the MPC750's performance monitor as L2 cache misses. The performance monitors can only be used to count misses when the L1 cache is enabled.

9.7 L2 Cache SRAM Timing Examples

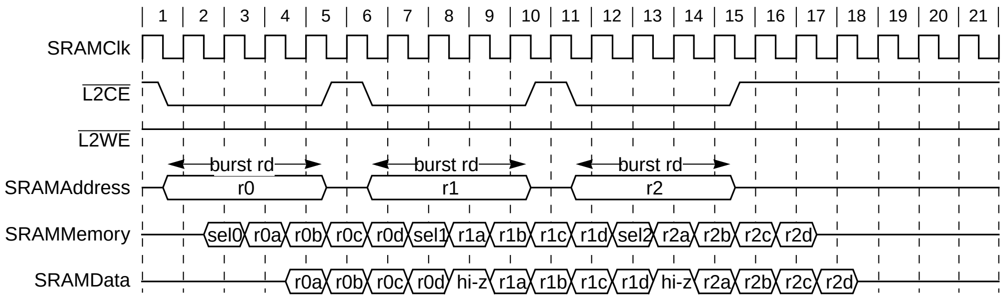
The *MPC750 User’s Manual* describes the signal timing for the three types of SRAM (flow-through burst SRAM, pipelined burst SRAM, and late-write SRAM) supported by the MPC750’s L2 cache interface. This section provides example timing diagrams for the new PB3 synchronous burst SRAMS supported by the MPC755. The timing diagrams illustrate the best case logical (ideal, not AC-timing accurate) interface operations. For proper interface operation, the designer must select SRAMs that support the signal sequencing illustrated in the timing diagrams. Note that the PB3 SRAMs operate differently from the PB2 SRAMS, and require a different configuration setting in L2CR.

PB3 SRAMs provide the efficiencies of the late-write SRAMs, but operate more like traditional PB2 SRAMs (that is, there is no internal write queue). They may be available at speeds comparable to late-write SRAMs, but closer to PB2 prices. They achieve their speed/price benefits by staging the initial internal array access over two clock cycles, thereby requiring an additional wait state for the first read data beat.

9.7.1 Pipelined PB3 Burst SRAM

Pipelined burst SRAMs operate at higher frequencies than flow-through burst SRAMs by clocking the read data from the memory array into a buffer before driving the data onto the data bus. This causes initial read accesses by the pipelined burst SRAMs to occur one cycle later than flow-through burst SRAMs, but the L2 bus frequencies supported can be higher. Note that the MPC750’s L2 cache interface requires the use of single-cycle deselect pipelined burst SRAMs for proper operation. Some PB3 SRAM devices have strobes with data latches that allow for very late clocking. The MPC755 doesn't support this feature. The MPC755 supports strobeless use of the PB3 devices and all timing (including setup times) must meet the specifications described in the *MPC755 Hardware Specifications.*

Figure 19 shows a burst read-read-read memory access sequence when the L2 cache interface is configured with PB3 burst SRAMs.


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
SRAMClk
L2CE
L2WE
burst rd
SRAMAddress
r0
r1
r2
SRAMMemory
sel0 r0a r0b r0c r0d sel1 r1a r1b r1c r1d sel2 r2a r2b r2c r2d
SRAMData
r0a r0b r0c r0d hi-z r1a r1b r1c r1d hi-z r2a r2b r2c r2d

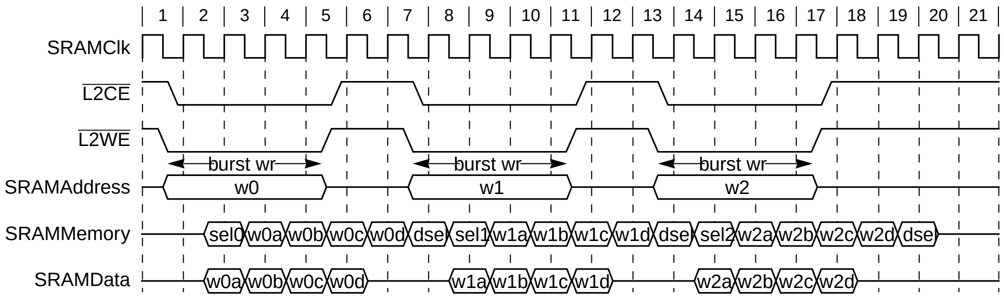
Notes:

For PB3, L2ZZ is reused as $\overline{\text{L2ADS}}$ and asserts during the 1st clock only of each $\overline{\text{L2CE}}$ assertion.

For PB3, the internal array access requires 1 cycle to row select, 1 cycle for each column select of burst (a–d), and 1 cycle to deselect if write.

Figure 19. Burst Read-Read-Read L2 Cache Access (Pipelined)

Figure 20 shows a burst write-write-write memory access sequence when the L2 cache interface is configured with PB3 burst SRAMs.


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
SRAMClk
L2CE
L2WE
burst wr
SRAMAddress
w0
w1
w2
SRAMMemory
sel0 w0a w0b w0c w0d dse sel1 w1a w1b w1c w1d dse sel2 w2a w2b w2c w2d dse
SRAMData
w0a w0b w0c w0d w1a w1b w1c w1d w2a w2b w2c w2d

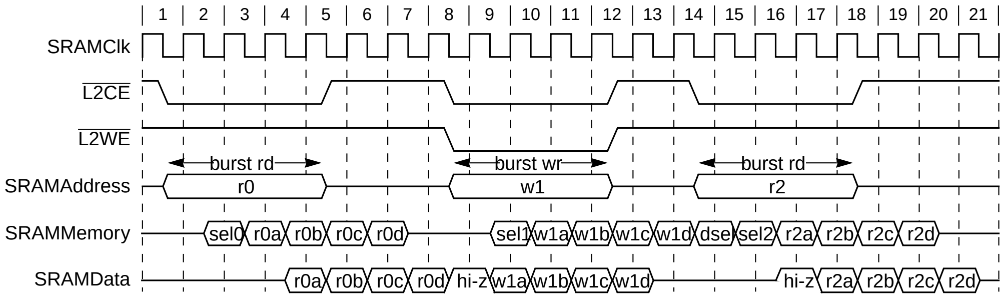
Notes:

For PB3, L2ZZ is reused as $\overline{\text{L2ADS}}$ and asserts during the 1st clock only of each $\overline{\text{L2CE}}$ assertion.

For PB3, the internal array access requires 1 cycle to row select, 1 cycle for each column select of burst (a–d), and 1 cycle to deselect if write.

Figure 20. Burst Write-Write-Write L2 Cache Access (Pipelined)

Figure 21 shows a burst read-write-read memory access sequence when the L2 cache interface is configured with PB3 burst SRAMs.



Notes:

For PB3, L2ZZ is reused as $\overline{\text{L2ADS}}$ and asserts during the 1st clock only of each $\overline{\text{L2CE}}$ assertion.

For PB3, the internal array access requires 1 cycle to row select, 1 cycle for each column select of burst (a–d), and 1 cycle to deselect if write.

Figure 21. Burst Read-Write-Read L2 Cache Access (Pipelined)

9.8 Private Memory SRAM Timing

The timing for private memory SRAM is the same as the L2 cache timing described in Section 9.7, “L2 Cache SRAM Timing Examples.”

Part X Power and Thermal Management (Chapter 10)

The power and thermal management of the MPC755 functions the same as that of the MPC750, and is completely described in the *MPC750 User’s Manual* except for the restriction on global L2 cache invalidation described in Section 9.6.2, “L2 Cache Global Invalidation.” Additionally, for both the MPC750 and MPC755, no combination of the thermal assist unit, the decrementer register, and the performance monitor can be used at any one time. If exceptions for any two of these functional blocks are enabled together, multiple exceptions caused by any of these three blocks cause unpredictable results.

Part XI Performance Monitor (Chapter 11)

The performance monitor of the MPC755 functions the same as that of the MPC750, and is completely described in the *MPC750 User’s Manual*, except that for both the MPC750 and MPC755, no combination of the thermal assist unit, the decrementer register, and the performance monitor can be used at any one time. If exceptions for any two of these functional blocks are enabled together, multiple exceptions caused by any of these three blocks cause unpredictable results.

INDEX

INDEX

INDEX

INDEX

INDEX

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

MOTOROLA