# IBM PowerPC 750GX and 750GL RISC Microprocessor

## User’s Manual

## Version 1.2

March 27, 2006

Printed in the United States of America March 2006.

The following are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

IBM
IBM Logo
POWER
PowerPC
PowerPC 750
PowerPC Architecture
PowerPC Logo

IEEE is a registered trademark in the United States, owned by the Institute of Electrical and Electronics Engineers.

Other company, product, and service names may be trademarks or service marks of others.

All information contained in this document is subject to change without notice. The products described in this document are NOT intended for use in applications such as implantation, life support, or other hazardous uses where malfunction could result in death, bodily injury, or catastrophic property damage. The information contained in this document does not affect or change IBM product specifications or warranties. Nothing in this document shall operate as an express or implied license or indemnity under the intellectual property rights of IBM or third parties. All information contained in this document was obtained in specific environments, and is presented as an illustration. The results obtained in other operating environments may vary.

THE INFORMATION CONTAINED IN THIS DOCUMENT IS PROVIDED ON AN "AS IS" BASIS. In no event will IBM be liable for damages arising directly or indirectly from any use of the information contained in this document.

IBM Microelectronics Division
2070 Route 52, Bldg. 330
Hopewell Junction, NY 12533-6351

The IBM home page can be found at **ibm.com**

The IBM Microelectronics Division home page can be found at **ibm.com**/chips

gx_title.fm.(1.2)
March 27, 2006

# List of Figures

# List of Tables

# About This Manual

This user's manual defines the functionality of the PowerPC® 750GX and 750GL RISC microprocessors. It describes features of the 750GX and 750GL that are not defined by the architecture. This book is intended as a companion to the *PowerPC Microprocessor Family: The Programming Environments* (referred to as *The Programming Environments Manual*).

**Note:** Soft copies of the latest version of this manual and documents referred to in this manual that are produced by IBM can be accessed on the world wide web as follows: http://www-3.ibm.com/chips/techlib.

**Note:** All information contained in this document referring to the PowerPC 750GX RISC Microprocessor also pertains to the IBM PowerPC 750GL RISC Microprocessor.

## Who Should Read This Manual

This manual is intended for system software developers, hardware developers, and applications programmers designing products for the 750GX. Readers should understand operating systems, microprocessor system design, basic principles of RISC processing, and details of the PowerPC Architecture™.

## Related Publications

### *PowerPC Architecture*

- May, Cathy, et. al., eds. *The PowerPC Architecture: A Specification for a New Family of RISC Processors*, Second Edition. San Francisco, CA: Morgan-Kaufmann, 1994.
- McClanahan, Kip. *PowerPC Programming for Intel Programmers*. Foster City, CA: Hungry Minds, 1995.
- Shanley, Tom. *PowerPC System Architecture,* Second Edition. Richardson, TX: Addison-Wesley, 1995.

### *PowerPC Microprocessor Documentation*

The latest version of this manual, errata, and other IBM documents referred to in this manual can be found at: http://www.ibm.com/chips/techlib.

- *PowerPC 750GX RISC Microprocessor Datasheet*. Provides data about bus timing, signal behavior, electrical and thermal characteristics, and other design considerations for each PowerPC implementation.
- *PowerPC Microprocessor Family: The Programming Environments Manual* (G522-0290-01). Provides information about resources defined by the PowerPC Architecture that are common to PowerPC processors.
- *Implementation Variances Relative to Rev. 1 of The Programming Environments Manual.*
- *PowerPC Microprocessor Family: The Programmer's Pocket Reference Guide* (SA14-2093-00). This foldout card provides an overview of the PowerPC registers, instructions, and exceptions for 32-bit implementations.
- *PowerPC Microprocessor Family: The Programmer's Reference Guide* (MPRPPCPRG-01). Includes the register summary, memory control model, exception vectors, and the PowerPC instruction set.
- Application notes. These short documents contain information about specific design issues useful to programmers and engineers working with PowerPC processors.

# Conventions Used in This Manual

### *Notational Conventions*

**mnemonics** Instruction mnemonics are shown in lowercase bold.

*italics* Italics indicate variable command parameters. For example: **bcctr***x*. Book titles in text are set in italics.

0x0 Prefix to denote a hexadecimal number.

0b0 Prefix to denote a binary number.

**crf**D Instruction syntax used to identify a destination Condition Register (CR) field.

**r**A, **r**B Instruction syntax used to identify a source General Purpose Register (GPR).

rD Instruction syntax used to identify a destination GPR.

**fr**A, **fr**B, **fr**C Instruction syntax used to identify a source Floating Point Register (FPR).

**fr**D Instruction syntax used to identify a destination FPR.

REG[FIELD] Abbreviations or acronyms for registers are shown in uppercase text. Specific bits, fields, or ranges appear in brackets. For example, MSR[LE] refers to the little-endian mode enable bit in the Machine State Register.

x In certain contexts, such as a signal encoding, this indicates a don’t care.

*n* Used to express an undefined numerical value.

¬ NOT logical operator.

& AND logical operator.

| OR logical operator.

$\boxed{0\ 0\ 0\ 0}$ Indicates reserved bits or bit fields in a register. Although these bits can be written to as either ones or zeros, they are always read as zeros.

### *Terminology Conventions*

The following table describes terminology conventions used in this manual and the equivalent terminology used in the PowerPC Architecture specification.

| PowerPC Architecture Specification | 750GX User’s Manual |
|---|---|
| Data-storage interrupt (DSI) | DSI exception |
| Extended mnemonics | Simplified mnemonics |
| Fixed-point unit (FXU) | Integer unit (IU) |
| Instruction storage interrupt (ISI) | ISI exception |
| Interrupt | Exception |
| Privileged mode (or privileged state) | Supervisor-level privilege |
| Problem mode (or problem state) | User-level privilege |
| Real address | Physical address |
| Relocation | Translation |
| Storage (locations) | Memory |
| Storage (the act of) | Access |
| Store in | Write back |
| Store through | Write through |

### *Instruction Field Conventions*

The following table describes instruction field conventions used in this manual and the equivalent conventions from the PowerPC Architecture specification.

| PowerPC Architecture Specification | 750GX User’s Manual |
|---|---|
| BA, BB, BT | **crb**A, **crb**B, **crb**D (respectively) |
| BF, BFA | **crf**D, **crf**S (respectively) |
| D | d |
| DS | ds |
| FLM | FM |
| FRA, FRB, FRC, FRT, FRS | **fr**A, **fr**B, **fr**C, **fr**D, **fr**S (respectively) |
| FXM | CRM |
| RA, RB, RT, RS | **r**A, **r**B, **r**D, **r**S (respectively) |
| SI | SIMM |
| U | IMM |
| UI | UIMM |
| /, //, /// | 0...0 (shaded) |

## Using This Manual with the Programming Environments Manual

Because the PowerPC Architecture is designed to be flexible to support a broad range of processors, the *PowerPC Microprocessor Family: The Programming Environments Manual* provides a general description of features that are common to PowerPC processors and indicates those features that are optional or that might be implemented differently in the design of each processor.

This document and *The Programming Environments Manual* describe three levels, or programming environments, of the PowerPC Architecture:

- PowerPC user instruction set architecture (UISA)—The UISA defines the level of the architecture to which user-level software should conform. The UISA defines the base user-level instruction set, user-level registers, data types, memory conventions, and the memory and programming models seen by application programmers.
- PowerPC virtual environment architecture (VEA)—The VEA, which is the smallest component of the PowerPC Architecture, defines additional user-level functionality that falls outside typical user-level software requirements. The VEA describes the memory model for an environment in which multiple processors or other devices can access external memory and defines aspects of the cache model and cache-control instructions from a user-level perspective. The resources defined by the VEA are particularly useful for optimizing memory accesses and for managing resources in an environment in which other processors and other devices can access external memory.

  Implementations that conform to the PowerPC VEA also conform to the PowerPC UISA, but might not necessarily adhere to the OEA.
- PowerPC operating environment architecture (OEA)—The OEA defines supervisor-level resources typically required by an operating system. The OEA defines the PowerPC memory-management model, supervisor-level registers, and the exception model.

  Implementations that conform to the PowerPC OEA also conform to the PowerPC UISA and VEA.

Some resources are defined more generally at one level in the architecture and more specifically at another. For example, conditions that cause a floating-point exception are defined by the UISA, while the exception mechanism itself is defined by the OEA.

Because it is important to distinguish between the levels of the architecture in order to ensure compatibility across multiple platforms, those distinctions are shown clearly throughout this book.

For ease in reference, the arrangement of topics in this book follows that of *The Programming Environments Manual*. Topics build upon one another, beginning with a description and complete summary of 750GX-specific registers and instructions and progressing to more specialized topics such as 750GX-specific details regarding the cache, exception, and memory-management models. Therefore, chapters can include information from multiple levels of the architecture. (For example, the discussion of the cache model uses information from both the VEA and the OEA.)

*The PowerPC Architecture: A Specification for a New Family of RISC Processors* defines the architecture from the perspective of the three programming environments and remains the defining document for the PowerPC Architecture. For information about PowerPC documentation, see *Related Publications* on page 19.

# 1. PowerPC 750GX Overview

The IBM PowerPC 750GX reduced instruction set computer (RISC) Microprocessor is an implementation of the PowerPC Architecture™ with enhancements based on the IBM PowerPC 750™, 750CXe, and 750FX RISC microprocessor designs. This chapter provides an overview of the PowerPC 750GX microprocessor features, including a block diagram that shows the major functional components. It also describes how the 750GX implementation complies with the PowerPC Architecture definition.

**Note:** In this document, the IBM PowerPC 750GX RISC Microprocessor is abbreviated as 750GX or 750GX RISC Microprocessor.

## 1.1 750GX Microprocessor Overview

The 750GX is a 32-bit implementation of the PowerPC Architecture in a 0.13 micron CMOS technology with six levels of copper interconnect. The 750GX is designed for high performance and low power consumption. It provides a superset of functionality to the PowerPC 750 processor, including a complete 60x bus interface, and enhancements such as an integrated 1-MB L2 cache.

750GX implements the 32-bit portion of the PowerPC Architecture, which provides 32-bit effective addresses, integer data types of 8, 16, and 32 bits, and floating-point data types of single and double-precision. 750GX is a superscalar processor that can complete two instructions simultaneously.

It incorporates the following six execution units:

- Floating-point unit (FPU)
- Branch processing unit (BPU)
- System register unit (SRU)
- Load/store unit (LSU)
- Two integer units (IUs): IU1 executes all integer instructions. IU2 executes all integer instructions except multiply and divide instructions.

The ability to execute several instructions in parallel and the use of simple instructions with rapid execution times yield high efficiency and throughput for 750GX-based systems. Most integer instructions execute in one clock cycle. The FPU is pipelined; it breaks the tasks it performs into subtasks, and then executes in three successive stages. Typically, a floating-point instruction can occupy only one of the three stages at a time, freeing the previous stage to work on the next floating-point instruction. Thus, three single-precision floating-point instructions can be in the FPU execute stage at a time. Double-precision add instructions have a 3-cycle latency; double-precision multiply and multiply/add instructions have a 4-cycle latency.

*Figure 1-1, 750GX Microprocessor Block Diagram,* on page 25 shows 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.

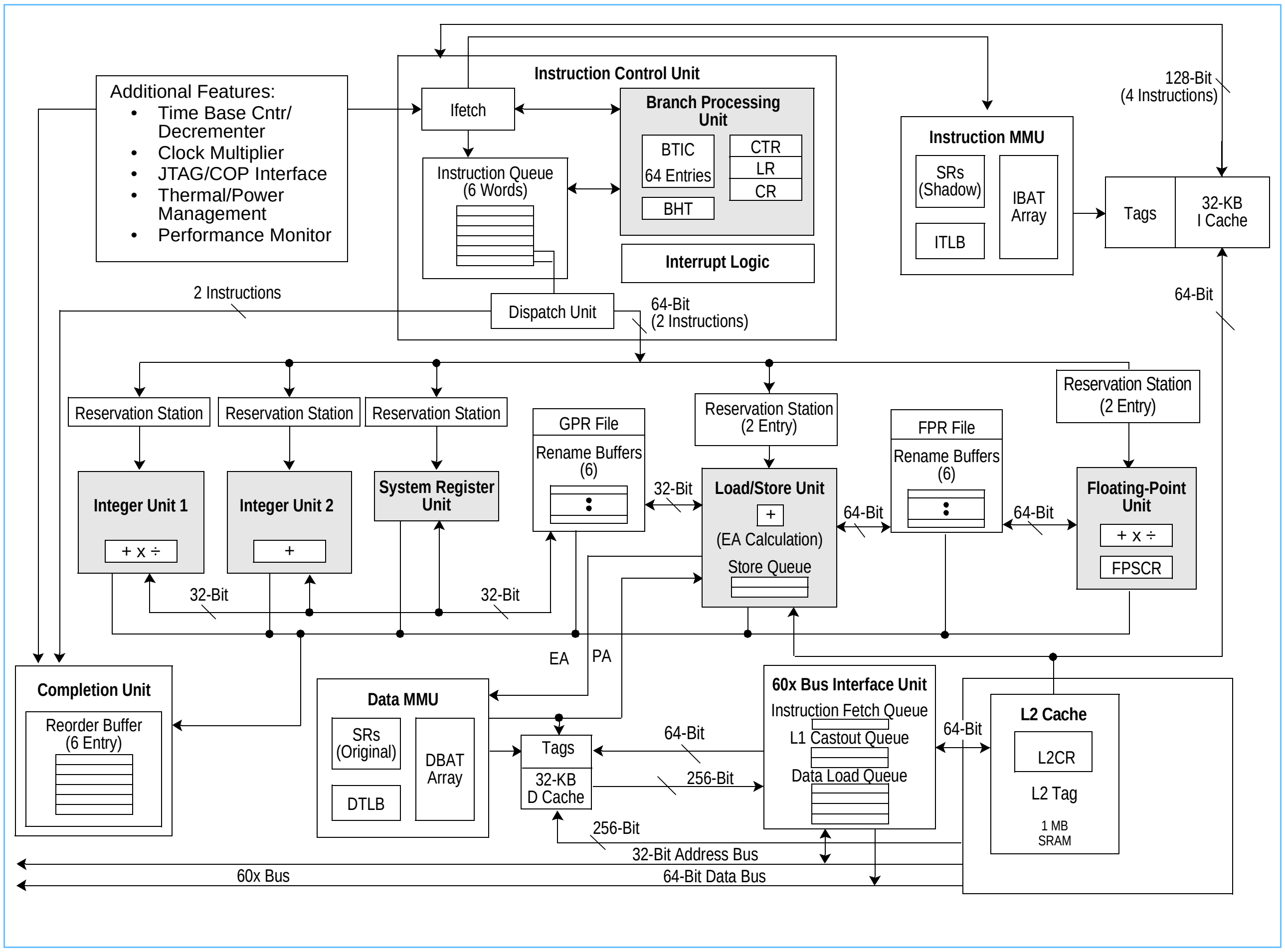
750GX has independent on-chip, 32-KB, 8-way set-associative, physically addressed caches for instructions and data, and independent instruction and data memory management units (MMUs). Each memory management unit has a 128-entry, 2-way set-associative translation lookaside buffer (DTLB and ITLB) that saves recently used page-address translations. Block-address translation is done through the 8-entry instruction and data block-address-translation (IBAT and DBAT) arrays, defined by the PowerPC Architecture. During block translation, effective addresses are compared simultaneously with all eight block-address-translation (BAT) entries.

For information about the L1 cache, see *Chapter 3, Instruction-Cache and Data-Cache Operation,* on page 121. The L2 cache is implemented with an on-chip, 4-way set-associative tag memory, and an on-chip 1-MB SRAM with error correction code (ECC) protection for data storage. For more information on the L2 Cache, see *Chapter 9* on page 323.

The 750GX has a 32-bit address bus and a 64-bit data bus. Multiple devices compete for system resources through a central external arbiter. The 750GX’s 3-state cache-coherency protocol (MEI) supports the modified, exclusive, and invalid states, a compatible subset of the MESI (modified/exclusive/shared/invalid) 4-state protocol, and it operates coherently in systems with 4-state caches. The 750GX supports single-beat and burst data transfers for external memory accesses and memory-mapped I/O operations. The system interface is described in *Chapter 7, Signal Descriptions,* on page 249 and *Chapter 8, Bus Interface Operation,* on page 279.

The 750GX has four software-controllable power-saving modes. The 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 750GX also provides a thermal assist unit (TAU) and a way to reduce the instruction fetch rate to limit power dissipation. Power management is described in *Chapter 10, Power and Thermal Management,* on page 335.

*Figure 1-1. 750GX Microprocessor Block Diagram*



## 1.2 750GX Microprocessor Features

This section lists features of the 750GX. The interrelationship of these features is shown in *Figure 1-1* on page 25.

Major features of 750GX are:

- 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 and completed 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, 4-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 effectively yielding a zero-cycle branch.

    - 512-entry branch history table (BHT) with two bits per entry for four levels of prediction—not-taken, strongly not-taken, taken, strongly taken.
    - Removal of Branch instructions that do not update the Count Register (CTR) or Link Register (LR) from the instruction stream.

  – Two integer units (IUs) that share 32 general purpose registers (GPRs) for integer operands.
    - IU1 can execute any integer instruction.
    - IU2 can execute all integer instructions except multiply and divide instructions (multiply, divide, 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.

  – 3-stage floating-point unit (FPU).
    - FPU fully compliant with IEEE® 754-1985 for both single-precision and double-precision operations.
    - Support for non-IEEE mode for time-critical operations.
    - Hardware support for denormalized numbers.
    - Hardware support for divide.
    - 2-entry reservation station.
    - Thirty-two 64-bit Floating Point Registers (FPRs) for single and double-precision operations.

  – 2-stage load/store unit (LSU).
    - 2-entry reservation station.
    - 4-entry load queue.
    - Single-cycle, pipelined cache access.
    - Dedicated adder performs effective address (EA) calculations.
    - Performs alignment and precision conversion for floating-point data.
    - Performs alignment and sign extension for integer data.
    - 3-entry store queue.
    - Supports both big-endian and little-endian modes.

  – System register unit (SRU) handles miscellaneous instructions.
    - Executes Condition Register (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 6-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 a sequential programming model and a precise-exception model.
  – Monitors all dispatched instructions and retires them in order.
  – Tracks unresolved branches and flushes instructions from the mispredicted branch path.

  - Retires as many as two instructions per clock.
- Separate on-chip L1 instruction and data caches (Harvard architecture).
  - 32-KB, 8-way set-associative instruction and data caches.
  - Pseudo least-recently-used (PLRU) replacement algorithm.
  - 32-byte (8-word) cache block.
  - Physically indexed/physical tags.

    **Note:** The PowerPC Architecture refers to physical address space as real address space.
  - Cache write-back or write-through operation programmable on a virtual-page or BAT-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 in software.
  - Data-cache coherency (MEI) maintained in hardware.
  - The critical double word is made available to the requesting unit when it is read into the line-fill buffer. The cache is nonblocking, so it can be accessed during block reload.
  - Nonblocking instruction cache (one outstanding miss).
  - Nonblocking data cache (four outstanding misses).
  - No snooping of instruction cache.
  - Parity for L1 tags and caches.
- Integrated L2 cache.
  - 1-MB on-chip ECC SRAMs.
  - On-chip 4-way set-associative tag memory.
  - ECC error correction for most single-bit errors; detection of remaining single-bit errors and all double-bit errors.
  - Copy-back or write-through data cache on a page basis, or for entire L2.
  - 64-byte line size, two sectors per line.
  - L2 frequency at core speed.
  - On-board ECC; parity for L2 tags.
  - Supports up to four outstanding misses (three data and one instruction or four data).
  - Cache locking by way.
- Separate memory management units (MMUs) for instructions and data.
  - 52-bit virtual address; 32-bit physical address.
  - Address translation for virtual pages or variable-sized BAT blocks.
  - Memory programmable as write-back or write-through, cacheable or noncacheable, and coherency enforced or coherency not enforced on a virtual-page or BAT block basis.
  - Separate IBAT and DBAT arrays (eight each) for instructions and data, respectively.
  - Separate virtual instruction and data translation lookaside buffers (TLBs).
    - Both TLBs are 128-entry, 2-way set associative, and use an LRU replacement algorithm.

        - TLBs are hardware-reloadable (the page table search is performed by hardware).
- Bus interface features:
    - Enhanced 60x bus that pipelines back-to-back reads to a depth of four. A dedicated snoop queue that allows snoop copybacks to also pipeline with up to the four maximum reads. Enveloped write transactions supported with the assertion of $\overline{\text{DBWO}}$.
    - Selectable bus-to-core clock frequency ratios of 2x, 2.5x, 3x, 3.5x, 4x, 4.5x, 5x, 5.5x, 6x, 6.5x, 7x, 7.5x, 8x, 8.5x, 9x, 9.5x, 10x, 11x, 12x, 13x, 14x, 15x, 16x, 17x, 18x, 19x, and 20x supported (2x, 2.5x, 3x, and 3.5x not supported with bus pipelining enabled).
    - A 64-bit, split-transaction external data bus with burst transfers.
    - Support for address pipelining and limited out-of-order bus transactions.
    - 8-word reload buffer for the L1 data cache.
    - Single-entry instruction fetch queue.
    - 2-entry L2 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.
    - Selectable I/O interface voltages of 1.8 V, 2.5 V, or 3.3 V
- Multiprocessing support features:
    - Hardware-enforced, 3-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 bus snooping, leaving only the Time Base Register and the PLL in a powered state.
        - Sleep—All internal functional units are disabled, after which external system logic can disable the PLL and SYSCLK.
    - Software-controllable thermal management. Thermal management is performed through the use of three supervisor-level registers and a 750GX-specific thermal-management exception.
    - Software-controlled frequency switching (dual PLL mode) to allow toggling between minimum and maximum frequencies to manage power consumption based on computational load.
    - Instruction-cache throttling provides control to slow instruction fetching to limit power consumption.
- Hardware-assist features for fault-tolerant systems including L2 ECC correction, parity checking on internal arrays, and dual-processor lockstep operation.
- Performance monitor can be used to help debug system designs and improve software efficiency.
- In-system testability and debugging features through Joint Test Action Group (JTAG) boundary-scan capability.

### 1.2.1 Instruction Flow

As shown in *Figure 1-1, 750GX Microprocessor Block Diagram,* on page 25, the 750GX instruction control unit provides centralized control of instruction flow to the execution units. The instruction unit contains a sequential instruction fetch (Ifetch), 6-entry instruction queue (IQ), dispatch unit, and BPU. It determines the address of the next instruction to be fetched based on information from the sequential instruction fetcher and from the BPU. See *Chapter 6, Instruction Timing,* on page 209 for more information.

The sequential instruction fetcher loads instructions from the instruction cache into the instruction queue. The BPU extracts branch instructions from the sequential instruction fetcher. Branch instructions that cannot be resolved immediately are predicted using either 750GX-specific dynamic branch prediction or the architecture-defined static branch prediction.

Branch instructions that do not update the LR or CTR are removed from (folded out of) the instruction stream. Instruction fetching continues along the predicted path of the branch instruction.

Instructions issued to execution units beyond a predicted branch can be executed but are not retired until the branch is resolved. If branch prediction is incorrect, the completion unit flushes all instructions fetched on the predicted path, and instruction fetching resumes along the correct path.

#### *1.2.1.1 Instruction Queue and Dispatch Unit*

The instruction queue (IQ), shown in *Figure 1-1* on page 25, holds as many as six instructions and loads up to four instructions from the instruction cache during a single-processor clock cycle. The instruction fetcher continuously attempts to load as many instructions as there were vacancies created in the IQ in the previous clock cycle. All instructions except branches are dispatched to their respective execution units from the bottom two positions in the instruction queue (IQ0 and IQ1) at a maximum rate of two instructions per cycle. Reservation stations are provided for the IU1, IU2, FPU, LSU, and SRU for dispatched instructions. The dispatch unit checks for source and destination register dependencies, allocates rename buffers, determines whether a position is available in the completion queue, and inhibits subsequent instruction dispatching if these resources are not available.

Branch instructions can be detected, decoded, and predicted from anywhere in the instruction queue. For a more detailed discussion of instruction dispatch, see *Section 6.6.1, Branch, Dispatch, and Completion-Unit Resource Requirements,* on page 237.

#### *1.2.1.2 Branch Processing Unit (BPU)*

The BPU receives branch instructions from the sequential instruction fetcher and performs CR lookahead operations on conditional branches to resolve them early, achieving the effect of a zero-cycle branch in many cases.

Unconditional branch instructions and conditional branch instructions in which the condition is known can be resolved immediately. For unresolved conditional branch instructions, the branch path is predicted using either the architecture-defined static branch prediction or 750GX-specific dynamic branch prediction. Dynamic branch prediction is enabled if the BHT bit in Hardware-Implementation-Dependent Register 0 is set (HID0[BHT] = 1).

When a prediction is made, instruction fetching, dispatching, and execution continue along the predicted path, but instructions cannot be retired and write results back to architected registers until the prediction is determined to be correct (resolved). When a prediction is incorrect, the instructions from the incorrect path are flushed from the processor, and instruction fetching resumes along the correct path. The 750GX allows a second branch instruction to be predicted; instructions from the second predicted branch instruction stream can be fetched but cannot be dispatched. These instructions are held in the instruction queue.

Dynamic prediction is implemented using a 512-entry BHT. The BHT is a cache that provides two bits per entry that together indicate four levels of prediction for a branch instruction—not-taken, strongly not-taken, taken, strongly taken. When dynamic branch prediction is disabled, the BPU uses a bit in the instruction encoding to predict the direction of the conditional branch. Therefore, when an unresolved conditional branch instruction is encountered, the 750GX executes instructions from the predicted path although the results are not committed to architected registers until the conditional branch is resolved. This execution can continue until a second unresolved branch instruction is encountered.

When a branch is taken (or predicted as taken), the instructions from the untaken path must be flushed, and the target instruction stream must be fetched into the IQ. The BTIC is a 64-entry cache that contains the most recently used branch target instructions, typically in pairs. When an instruction fetch hits in the BTIC, the instructions arrive in the instruction queue in the next clock cycle, a clock cycle sooner than they would arrive from the instruction cache. Additional instructions arrive from the instruction cache in the next clock cycle. The BTIC reduces the number of missed opportunities to dispatch instructions and gives the processor a 1-cycle head start on processing the target stream. With the use of the BTIC, the 750GX achieves a zero-cycle delay for branches taken. Coherency of the BTIC table is maintained by table reset on an instruction-cache flash invalidate, Instruction Cache Block Invalidate (**icbi**) or Return from Interrupt (**rfi**) instruction execution, or when an exception is taken.

The BPU contains an adder to compute branch target addresses and three user-control registers—the Link Register (LR), the Count Register (CTR), and the CR. The BPU calculates the return pointer for subroutine calls and saves it into the LR for certain types of branch instructions. The LR also contains the branch target address for the Branch Conditional to Link Register (**bclr***x*) instruction. The CTR contains the branch target address for the Branch Conditional to Count Register (**bcctr***x*) instruction. Because the LR and CTR are special purpose registers (SPRs), their contents can be copied to or from any GPR. Since the BPU uses dedicated registers rather than GPRs or FPRs, execution of branch instructions is largely independent from execution of fixed-point and floating-point instructions.

### *1.2.1.3 Completion Unit*

The completion unit operates closely with the dispatch unit. Instructions are fetched and dispatched in program order. At the point of dispatch, the program order is maintained by assigning each dispatched instruction a successive entry in the 6-entry completion queue. The completion unit tracks instructions from dispatch through execution and retires them in program order from the two bottom entries in the completion queue (CQ0 and CQ1).

Instructions cannot be dispatched to an execution unit unless there is a vacancy in the completion queue and rename buffers are available. Branch instructions that do not update the CTR or LR are removed from the instruction stream and do not occupy a space in the completion queue. Instructions that update the CTR and LR follow the same dispatch and completion procedures as nonbranch instructions, except that they are not issued to an execution unit.

An instruction is retired when it is removed from the completion queue and its results are written to architected registers (GPRs, FPRs, LR, and CTR) from the rename buffers. In-order completion ensures program integrity and the correct architectural state when the 750GX must recover from a mispredicted branch or any exception. Also, the rename buffers assigned to it by the dispatch unit are returned to the available rename buffer pool. These rename buffers are reused by the dispatch unit as subsequent instructions are dispatched.

For a more detailed discussion of instruction completion, see *Section 6.6.1, Branch, Dispatch, and Completion-Unit Resource Requirements,* on page 237.

### 1.2.2 Independent Execution Units

In addition to the BPU, the 750GX has the following five execution units:

- Two integer units (IUs)
- Floating-point unit (FPU)
- Load/store unit (LSU)
- System register unit (SRU)

#### *1.2.2.1 Integer Units (IUs)*

The integer units, IU1 and IU2, are shown in *Figure 1-1* on page 25. IU1 can execute any integer instruction; IU2 can execute any integer instruction except multiplication and division instructions. Each IU has a single-entry reservation station that can receive instructions from the dispatch unit and operands from the GPRs or the rename buffers. The output of the IU is latched in the rename buffer assigned to the instruction by the dispatch unit.

Each IU consists of three single-cycle subunits—a fast adder/comparator, a subunit for logical operations, and a subunit for performing rotates, shifts, and count-leading-zero operations. These subunits handle all 1-cycle arithmetic and logical integer instructions; only one subunit can execute an instruction at a time.

The IU1 has a 32-bit integer multiplier/divider, as well as the adder, shift, and logical units of the IU2. The multiplier supports early exit for operations that do not require full 32 × 32-bit multiplication. Multiply and divide instructions spend several cycles in the execution stage before the results are written to the output rename buffer.

#### *1.2.2.2 Floating-Point Unit (FPU)*

The FPU, shown in *Figure 1-1* on page 25, is designed as a 3-stage pipelined processing unit, where the first stage is for multiply, the second stage is for add, and the third stage is for normalize. A single-precision multiply/add operation is processed with 1-cycle throughput and 3-cycle latency. (A single-precision instruction spends one cycle in each stage of the FPU). A double-precision multiply requires two cycles in the multiply stage and one cycle in each additional stage. A double-precision multiply/add has a 2-cycle throughput and a 4-cycle latency. As instructions are dispatched to the FPU reservation station, source operand data can be accessed from the FPRs or from the FPR rename buffers. Results, in turn, are written to the rename buffers and are made available to subsequent instructions. Instructions pass through the reservation station and the pipeline stages in program order. Stalls due to contention for FPRs are minimized by automatic allocation of the six floating-point rename buffers. The completion unit writes the contents of the rename buffer to the appropriate FPR when floating-point instructions are retired.

The 750GX supports all IEEE 754-1985 floating-point data types (normalized, denormalized, not a number (NaN), zero, and infinity) in hardware, eliminating the latency incurred by software exception routines. (Note that "exception" is also referred to as "interrupt" in the architecture specification.)

#### *1.2.2.3 Load/Store Unit (LSU)*

The LSU executes all load-and-store instructions and provides the data-transfer interface between the GPRs, FPRs, and the data-cache/memory subsystem. The LSU functions as a 2-stage pipelined unit, which calculates effective addresses in the first stage. In the second stage, the address is translated, the cache is accessed, and the data is aligned if necessary. Unless extensive data alignment is required (for example, to cross a double-word boundary), the instructions complete in two cycles with a 1-cycle throughput. The LSU also provides sequencing for load/store string and multiple register transfer instructions.

Load-and-store instructions are translated and issued in program order. However, some memory accesses can occur out of order. Synchronizing instructions can be used to enforce strict ordering if necessary. When there are no data dependencies and the guard bit for the page or block is cleared, a maximum of one out-of-order cacheable load operation can execute per cycle, with a 2-cycle total latency on a cache hit. Data returned from the cache is held in a rename buffer until the completion logic commits the value to a GPR or FPR. Stores cannot be executed out of order and are held in the store queue until the completion logic signals that the store operation is to be completed to memory. The 750GX executes store instructions with a maximum throughput of one per cycle and a 3-cycle latency to the data cache. The time required to perform the actual load or store operation depends on the processor/bus clock ratio and whether the operation involves the L1 cache, the L2 cache, system memory, or an I/O device.

The L/S  unit has two reservation stations, Eib0 and Eib1. For loads, there is also a  hold queue and a miss queue. A load that misses in the dcache advances from Eib0 to the miss queue, where only necessary state for instruction completion like the instruction ID and register rename ID are stored. If another load misses under an outstanding miss, then it is held in the hold queue and Eib0 is free. Two more load instructions may now be dispatched to Eib0 and Eib1. The  Miss-under-Miss feature allows the hold, Eib0, and Eib1 load requests to proceed out to the bus, even though there is an outstanding miss that would normally stall the pending loads.

#### *1.2.2.4 System Register Unit (SRU)*

The SRU executes various system-level instructions, as well as Condition Register logical operations and Move-to/Move-from Special-Purpose Register instructions. To maintain system state, most instructions executed by the SRU are execution-serialized with other instructions; that is, the instruction is held for execution in the SRU until all previously issued instructions have been retired. Results from execution-serialized instructions executed by the SRU are not available or forwarded for subsequent instructions until the instruction completes.

### 1.2.3 Memory Management Units (MMUs)

The 750GX’s MMUs support up to 4 petabytes ($2^{52}$) of virtual memory and 4 gigabytes ($2^{32}$) of physical memory for instructions and data. The MMUs also control access privileges for these spaces on block and page granularities. Referenced and changed status is maintained by the processor for each page to support demand-paged virtual memory systems.

The LSU, with the aid of the MMU, translates effective addresses for data loads and stores. The effective address is calculated on the first cycle, and the MMU translates it to a physical address at the same time it is accessing the L1 cache on the second cycle. The MMU also provides the necessary control and protection information to complete the access. By the end of the second cycle, the data and control information is available if no miss conditions for translate and cache access were encountered. This yields a 1-cycle throughput and a 2-cycle latency.

The 750GX supports the following types of memory translation:

Real-addressing mode
In this mode, translation is disabled (control bit MSR(IR) = 0 for instructions and control bit MSR(DR) = 0 for data). The effective address is used as the physical address to access memory.

Virtual-page-address translation
Translates from an effective address to a physical address by using the Segment Registers and the TLB and access data from a 4-KB virtual page. This page is either in physical memory or on disk. If the latter, a page-fault exception occurs.

Block-address translation
Translates the effective address into a physical address by using the BAT Registers and accesses a block (128 KB to 256 MB) in memory.

If translation is enabled, the appropriate MMU translates the higher-order bits of the effective address into physical address bits by using either BATs or the page translation method. The lower-order address bits, which are untranslated and therefore, considered both logical and physical, are directed to the L1 caches where they form the index into the 8-way set-associative tag and data arrays. After translating the address, the MMU passes the higher-order physical address bits to the cache, and the cache lookup completes. For caching-inhibited accesses or accesses that miss in the cache, the untranslated lower-order address bits are concatenated with the translated higher-order address bits. The resulting 32-bit physical address is used and accesses the L2 cache or system memory via the 60x bus.

If the BAT Registers are enabled and the address translates via this method, the page translation is canceled and the high-order physical address bits from the BAT Register are forward to the cache/memory access system. There are eight 8-byte BAT Registers, which function like an associative memory. These registers provide cache-control and protection information as well as address translation. Only one of the eight BAT entries should translate a given effective address.

If address relocation is enabled and the effective address does not translate via the BAT method, the virtual-page method is used. The four high-order bits of the effective address are used to access the 16-entry Segment Register array. From this array, a 24-bit Segment Register is accessed and used to form the high-order bits of a 52-bit virtual address. The low-order 28 bits of the effective address are used to form the low-order bits of the virtual address. This 52-bit virtual address is translated into a physical address by doing a lookup in the TLB. If the lookup is successful, a physical address is formed by using 16 low-order bits from the virtual address and 16 high-order bits from the TLB. The TLB also provides cache-control and protection information to be used by the cache/memory system.

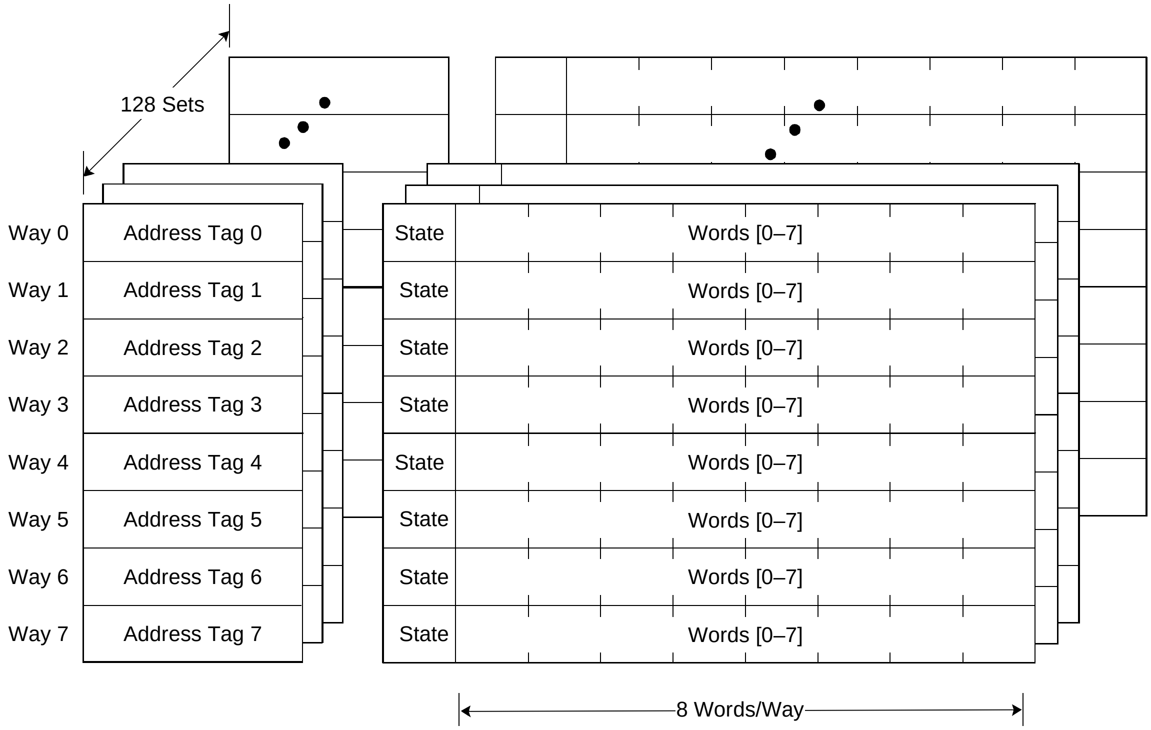
TLBs are 128-entry, 2-way, set-associative caches that contain information about recently translated virtual addresses. When an address translation is not in a TLB, the 750GX automatically generates a page table search in memory to update the TLB. This search could find the desired entry in the L1 or L2 cache or in the page table in memory. The time to reload a TLB entry depends on where it is found; it could be completed in just a few cycles. If memory is searched, a maximum of 16 bus cycles would be needed before a page-fault exception is signaled.

### 1.2.4 On-Chip Level 1 Instruction and Data Caches

The 750GX implements separate instruction and data caches. Each cache is 32-KB and 8-way set-associative. The caches are physically indexed. Each cache block contains eight contiguous words from memory that are loaded from an 8-word boundary (bits EA[27–31] are zeros); thus, a cache block never crosses a page boundary. A miss in the L1 cache causes a block reload from either the L2 cache, if the block is in the L2 cache, or from main memory. The critical double word is accessed first, forwarded to the load/store unit, and written into an 8-word buffer. Subsequent double words are fetched from either the L2 cache or the system memory and written into the buffer. Once the total block is in the buffer, the line is written into the L1 cache in a single cycle. This minimizes write cycles into the L1 cache, leaving more read/write cycles available to the LSU. The L1 is nonblocking and supports hits under misses during this block reload sequence. Misaligned accesses across a block or page boundary can incur a performance penalty. The 750GX L1 data cache supports miss-under-miss access, meaning that with one miss outstanding, the cache can continue to be accessed for up to three more misses. The 750GX L1 data cache also allows the additional misses to initiate a transaction in the bus interface unit, while the first miss is pending.

The 750GX L1 cache organization is shown in *Figure 1-2, L1 Cache Organization*.

*Figure 1-2. L1 Cache Organization*



The data cache provides double-word accesses to the LSU each cycle. Like the instruction cache, the data cache can be invalidated all at once or on a per-cache-block basis. The data cache can be disabled and invalidated by clearing the data-cache enable bit (HID0[DCE]) and setting the data-cache flash invalidate bit (HID0[DCFI]). The data cache can be locked by setting HID0[DLOCK]. To ensure cache coherency, the data cache supports the 3-state MEI protocol. The data-cache tags are single-ported, so a simultaneous load or store and a snoop access represent a resource collision, and an LSU access is delayed for one cycle. If a snoop hit occurs and a castout is required, the LSU is blocked internally for one cycle to allow the 8-word block of data to be copied to the write-back buffer.

The instruction cache provides up to four instructions to the instruction queue in a single cycle. Like the data cache, the instruction cache can be invalidated all at once or on a cache-block basis. The instruction cache can be disabled and invalidated by clearing the instruction-cache enable bit (HID0[ICE]) and setting the instruction-cache flash invalidate bit (HID0[ICFI]). The instruction cache can be locked by setting HID0[ILOCK]. The instruction cache supports only the valid and invalid states, and requires software to maintain coherency if the underlying program changes.

The 750GX also implements a 64-entry (16-set, 4-way set-associative) branch target instruction cache (BTIC). The BTIC is a cache of branch instructions that have been encountered in branch/loop code sequences. If the 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, the BTIC contains the first two instructions in the target stream. The BTIC can be disabled and invalidated through software.

Coherency of the BTIC is transparent to the running software and is coupled with various functions in the 750GX processor. When the BTIC is enabled and loaded with instruction pairs to support zero-cycle delay on branches taken, the table must be invalidated if the underlying program changes. (This is also true for the instruction cache.) The BTIC is invalidated on an instruction-cache flash invalidate, an **icbi** or **rfi** instruction, and any exception.

For more information and timing examples showing cache hit and cache miss latencies, see *Section 6.3.2, Instruction Fetch Timing,* on page 216.

### 1.2.5 On-Chip Level 2 Cache Implementation

The L2 cache is a unified cache that receives memory requests from both the L1 instruction and data caches independently. The L2 cache is implemented with an L2 Cache Control Register (L2CR), an on-chip, 4-way, set-associative tag array, and with a 1-MB, integrated SRAM for data storage. The L2 cache normally operates in write-back mode and supports cache coherency through snooping. The access interface to the L2 is 64 bits for writes and requires four cycles to write a single cache block. The access interface to the L2 is 256 bits for reads and requires one cycle to read a single cache block. The L2 uses ECC on a double word, corrects most single-bit errors, and detects the remaining single-bit errors and all double-bit errors. See *Figure 9-1, L2 Cache,* on page 327.

The L2 cache is organized with 64-byte lines, which in turn are subdivided into 32-byte blocks, the unit at which cache coherency is maintained. This reduces the size of the tag array, and one tag supports two cache blocks. Each 32-byte cache block has its own valid and modified status bits. When a cache line is removed, the contents of both blocks and the tag are removed from the L2 cache. The cache block is only written to system memory if the modified bit is set.

Requests from the L1 cache generally result from instruction misses, data load or store misses, write-through operations, or cache-management instructions. Misses from the L1 cache are looked up in the L2 tags and serviced by the L2 cache if they hit; they are forwarded to the 60x bus interface if they miss.

The L2 cache can accept multiple, simultaneous accesses. However, they are serialized and processed one per cycle. The L1 instruction cache can request an instruction at the same time that the L1 data cache requests one load and two store operations. The L2 cache also services snoop requests from the bus. If there are multiple pending requests to the L2 cache, snoop requests have highest priority. Load-and-store requests from the L1 data cache have the next highest priority. The last priority consists of instruction fetch requests from the L1 instruction cache.

### 1.2.6 System Interface/Bus Interface Unit (BIU)

The PowerPC 750GX uses a reduced system signal set, which eliminates some optional 60x bus protocol pins. The system designer needs to make note of these differences.

The address and data buses operate independently. Address and data tenures of a memory access are decoupled to provide more flexible control of bus traffic. The primary activity of the system interface is transferring data and instructions between the processor and system memory. There are two types of memory accesses:

Single-beat transfers
Allow transfer sizes of 8, 16, 24, 32, or 64 bits in one bus clock cycle. Single-beat transactions are caused by uncacheable read and write operations that access memory directly when caches are disabled, for cache-inhibited accesses, and for stores in write-through mode. The two latter accesses are defined by control bits provided by the MMU during address translation.

4-beat burst (32-byte) data transfers
Burst transactions, which always transfer an entire cache block (32 bytes), are initiated when an entire cache block is transferred. If the caches on the 750GX are enabled and using write-back mode, burst-read operations are the most common memory accesses, followed by burst-write memory operations.

The 750GX also supports address-only operations, which are variants of the burst and single-beat operations (for example, atomic memory operations and global memory operations that are snooped), and address retry activity (for example, when a snooped read access hits a modified block in the cache). The broadcast of some address-only operations is controlled through the address broadcast enable bit (HID0[ABE]). I/O accesses use the same protocol as memory accesses.

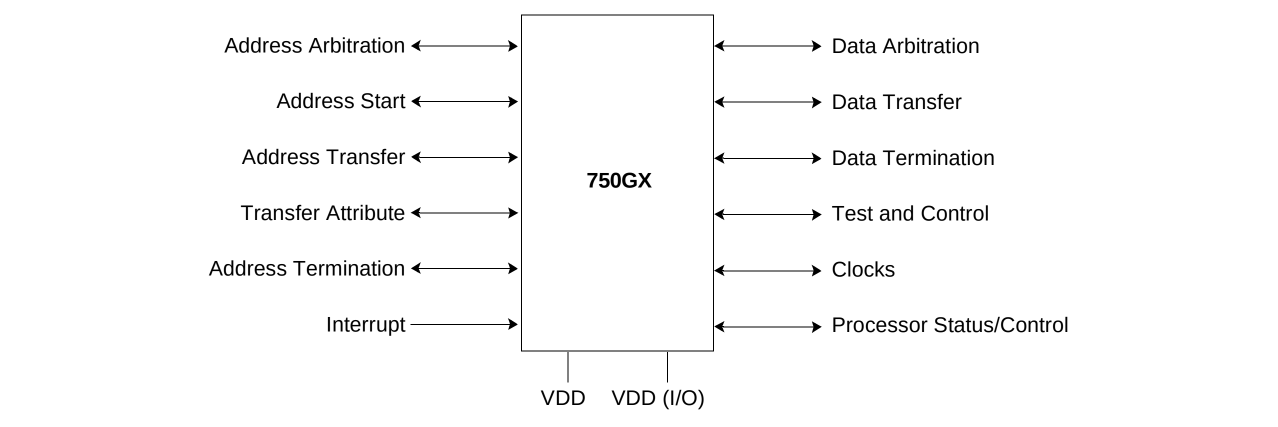
Access to the system interface is granted through an external arbitration mechanism that allows devices to compete for bus mastership. This arbitration mechanism is flexible, allowing the 750GX to be integrated into systems that implement various fairness and bus-parking procedures to avoid arbitration overhead.

Typically, memory accesses are weakly ordered—sequences of operations, including load/store string and multiple instructions, do not necessarily complete in the order they begin. This maximizes the efficiency of the bus without sacrificing data coherency. The 750GX allows read operations to go ahead of store operations except when a dependency exists, or when a noncacheable access is performed. It also allows a write operation to go ahead of a previously queued read data tenure (for example, letting a snoop push be enveloped between address and data tenures of a read operation). Because the 750GX can dynamically optimize runtime ordering of load/store traffic, overall performance is improved.

The system interface is specific for each PowerPC microprocessor implementation.

The 750GX signals are grouped as shown in *Figure 1-3, System Interface*. Test and control signals provide diagnostics for selected internal circuits.

*Figure 1-3. System Interface*



The system interface supports address pipelining, which allows the address tenure of one transaction to overlap the data tenure of another. The 750GX can support up to five outstanding transactions on the bus, including up to one snoop copyback, up to four loads, and up to four stores. The extent of the pipelining depends on external arbitration and control circuitry. Similarly, the 750GX supports split-bus transactions for systems with multiple potential bus masters—one device can be master of the address bus while another is master of the data bus. Allowing multiple bus transactions to occur simultaneously increases the available bus bandwidth for other activity.

The 750GX's clocking structure supports a wide range of processor-to-bus clock ratios.

### 1.2.7 Signals

The 750GX's signals are grouped as follows:

Address arbitration
The 750GX uses these signals to arbitrate for address-bus mastership.

Address start
This signal indicates that a bus master has begun a transaction on the address bus.

Address transfer
These signals include the address bus and are used to transfer the address.

Transfer attribute
These signals provide information about the type of transfer, such as the transfer size and whether the transaction is burst, write-through, or caching-inhibited.

Address termination
These signals are used to acknowledge the end of the address phase of the transaction. They also indicate whether a condition exists that requires the address phase to be repeated.

Data arbitration
The 750GX uses these signals to arbitrate for data-bus mastership.

Data transfer
These signals include the data bus and are used to transfer the data.

Data termination
These signals are required after each data beat in a data transfer. In a single-beat transaction, a data termination signal also indicates the end of the tenure. In burst accesses, data termination signals apply to individual beats and indicate the end of the tenure only after the final data beat.

Interrupt — These signals include the interrupt signal, checkstop signals, and both soft reset and hard reset signals. These signals are used to generate interrupt exceptions and, under various conditions, to reset the processor.

Processor status/control — These signals are used to indicate miscellaneous bus functions.

Clocks — These signals determine the system clock frequency. These signals can also be used to synchronize multiprocessor systems.

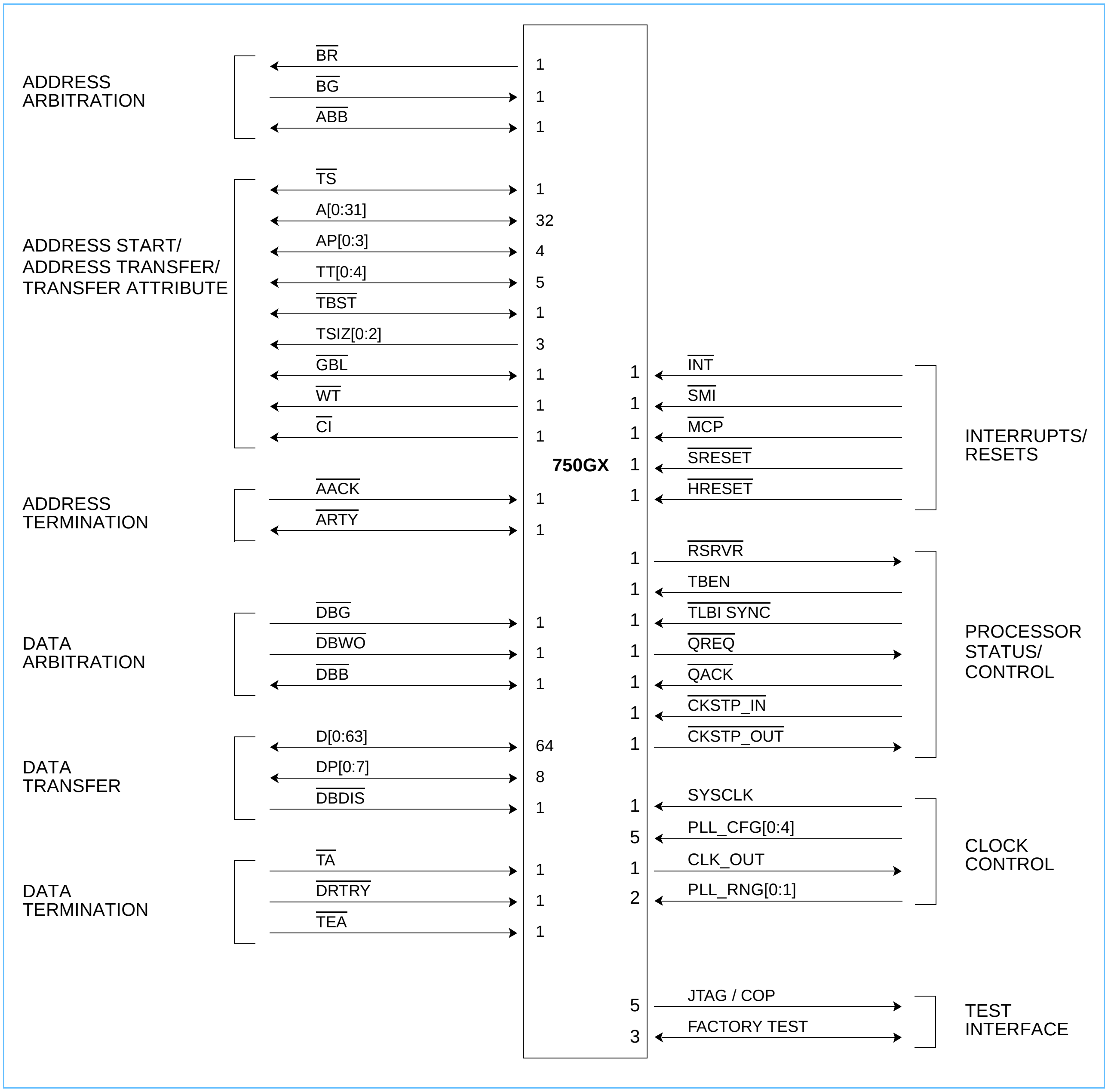
Test and control — The common on-chip processor (COP) unit provides a serial interface to the system for performing board-level boundary scan interconnect tests.

**Note:** A bar over a signal name indicates that the signal is active low—for example, $\overline{\text{ARTRY}}$ (address retry) and $\overline{\text{TS}}$ (transfer start). Active-low signals are referred to as asserted (active) when they are low and as negated when they are high. Signals that are not active low, such as A[0–31] (address-bus signals) and TT[0–4] (transfer type signals) are referred to as asserted when they are high and as negated when they are low.

### 1.2.8 Signal Configuration

*Figure 1-4* shows the 750GX's logical pin configuration. The signals are grouped by function.

*Figure 1-4. 750GX Microprocessor Signal Groups*



Signal functionality is described in detail in *Chapter 7, Signal Descriptions,* on page 249 and *Chapter 8, Bus Interface Operation,* on page 279.

**Note:** See the *PowerPC 750GX Datasheet* for a complete list of signal pins.

### 1.2.9 Clocking

The 750GX requires a single system clock input, SYSCLK, that represents the bus interface frequency. Internally, the processor uses a phase-locked loop (PLL) circuit to generate a master core clock that is frequency-multiplied and phase-locked to the SYSCLK input. This core frequency is used to operate the internal circuitry.

The PLL is configured by the PLL_CFG[0:4] signals, which select the multiplier that the PLL uses to multiply the SYSCLK frequency up to the internal core frequency. In addition, the 750GX has two PLL_RNG bits that set the proper operation frequency range. The feedback in the PLL guarantees that the processor clock is phase locked to the bus clock, regardless of process variations, temperature changes, or parasitic capacitances.

The PLL also ensures a 50% duty cycle for the processor clock.

The 750GX supports various processor-to-bus clock frequency ratios, although not all ratios are available for all frequencies. Configuration of the processor/bus clock ratios is displayed through a 750GX-specific register, HID1. For information about supported clock frequencies, see the *PowerPC 750GX Datasheet*.

## 1.3 750GX Microprocessor Implementation

The PowerPC Architecture is derived from the Performance Optimized with Enhanced RISC (POWER™) architecture. The PowerPC Architecture shares the benefits of the POWER architecture optimized for single-chip implementations. The PowerPC Architecture design facilitates parallel instruction execution, and is scalable to take advantage of future technological gains.

The remainder of this chapter describes the PowerPC Architecture in general, and specific details about the implementation of 750GX as a low-power, 32-bit member of the PowerPC processor family. The structure of the remainder of this chapter reflects the organization of the user’s manual; each section provides an overview of the corresponding chapter. The following sections summarize the features of the 750GX, distinguishing those that are defined by the architecture from those that are unique to the 750GX implementation.

Registers and programming model

*Section 1.4, PowerPC Registers and Programming Model,* on page 42 describes the registers for the operating environment architecture common among PowerPC processors and describes the programming model. It also describes the registers that are unique to the 750GX. The information in this section is described more fully in *Chapter 2, Programming Model,* on page 57.

Instruction set and addressing modes

*Section 1.5, Instruction Set,* on page 45 describes the PowerPC instruction set and addressing modes for the PowerPC operating environment architecture, defines the PowerPC instructions implemented in the 750GX, and describes new instruction set extensions to improve the performance of single-precision floating-point operations and the capability of data transfer. The information in this section is described more fully in *Section 2.3, Instruction Set Summary,* on page 86.

Cache implementation

*Section 1.6, On-Chip Cache Implementation,* on page 47 describes the cache model that is defined generally for PowerPC processors by the virtual environment architecture. It also provides specific details about the 750GX L2 cache implementation. The information in this section is described more fully in *Chapter 3, Instruction-Cache and Data-Cache Operation,* on page 121.

Exception mode

*Section 1.7, Exception Model,* on page 48 describes the exception model of the PowerPC operating environment architecture and the differences in the 750GX exception model. The information in this section is described more fully in *Chapter 4, Exceptions,* on page 151.

Memory management

*Section 1.8, Memory Management,* on page 51 describes in general terms the conventions for memory management among the PowerPC processors. This section also describes the 750GX's implementation of the 32-bit PowerPC memory-management specification. The information in this section is described more fully in *Chapter 5, Memory Management,* on page 179.

Instruction timing

*Section 1.9, Instruction Timing,* on page 52 provides a general description of the instruction timing provided by the superscalar, parallel execution supported by the PowerPC Architecture and the 750GX. The information in this section is described in more detail in *Chapter 6, Instruction Timing,* on page 209.

Power management

*Section 1.10, Power Management,* on page 54 describes how power management can be used to reduce power consumption when the processor, or portions of it, are idle. The information in this section is described more fully in *Chapter 10, Power and Thermal Management,* on page 335.

Thermal management

*Section 1.11, Thermal Management,* on page 55 describes how the thermal-management unit and its associated registers (THRM1–THRM4) and exception processing can be used to manage system activity in a way that prevents exceeding system and junction temperature thresholds. This is particularly useful in high-performance portable systems, which cannot use the same cooling mechanisms (such as fans) that control overheating in desktop systems. The information in this section is described more fully in *Chapter 10, Power and Thermal Management,* on page 335.

Performance monitor

*Section 1.12, Performance Monitor,* on page 56 describes the performance-monitor facility, which system designers can use to help bring up, debug, and optimize software performance. The information in this section is described more fully in *Chapter 11, Performance Monitor and System Related Features,* on page 349.

The PowerPC Architecture consists of the following layers, and adherence to the PowerPC Architecture can be described in terms of which of the following levels of the architecture is implemented.

PowerPC user instruction set architecture (UISA)

Defines the base user-level instruction set, user-level registers, data types, floating-point exception model, memory models for a uniprocessor environment, and programming model for a uniprocessor environment.

PowerPC virtual environment architecture (VEA)

Describes the memory model for a multiprocessor environment, defines cache-control instructions, and describes other aspects of virtual environments. Implementations that conform to the VEA also adhere to the UISA, but might not necessarily adhere to the OEA.

PowerPC operating environment architecture (OEA)

Defines the memory-management model, supervisor-level registers, synchronization requirements, and the exception model. Implementations that conform to the OEA also adhere to the UISA and the VEA.

## 1.4 PowerPC Registers and Programming Model

The PowerPC Architecture defines register-to-register operations for most computational instructions. Source operands for these instructions are accessed from the registers or are provided as immediate values embedded in the instruction itself. The 3-register instruction format allows specification of a target register distinct from the two source operands. Only load-and-store instructions transfer data between registers and memory.

PowerPC processors have two levels of privilege: supervisor mode and user mode.The supervisor mode of operation is typically used by the operating system. The user mode of operation, also called the problem state, is typically used by the application software. The programming models incorporate 32 GPRs, 32 FPRs, Special-Purpose Registers (SPRs), and several miscellaneous registers. Each PowerPC microprocessor also has its own unique set of Hardware-Implementation-Dependent (HID) Registers.

While running in supervisor mode, the operating system is able to execute all instructions and access all registers defined in the PowerPC Architecture. In this mode, the operating system establishes all address translations and protection mechanisms, loads all Processor State Registers, and sets up all other control mechanisms defined in the PowerPC 750GX processor. While running in user mode (problem state), many of these registers and facilities are not accessible, and any attempt to read or write these register results in a program exception.

*Figure 2-1, PowerPC 750GX Microprocessor Programming Model—Registers,* on page 58 shows all the 750GX registers available at the user and supervisor levels. The numbers to the right of the SPRs indicate the number that is used in the syntax of the instruction operands to access the register. For more information, see *Chapter 2, Programming Model,* on page 57.

The following tables summarize the PowerPC registers implemented in 750GX, and describe registers (excluding SPRs) defined by the architecture.

*Table 1-1. Architecture-Defined Registers (Excluding SPRs)*

| Register | Level | Function |
|---|---|---|
| CR | User | The Condition Register (CR) consists of eight 4-bit fields that reflect the results of certain operations, such as move, integer and floating-point compare, arithmetic, and logical instructions. The register provides a mechanism for testing and branching. |
| FPRs | User | The 32 Floating Point Registers (FPRs) serve as the data source or destination for floating-point instructions. These 64-bit registers can hold single-precision or double-precision floating-point values. |
| FPSCR | User | The Floating-Point Status and Control Register (FPSCR) contains the floating-point exception signal bits, exception summary bits, exception enable bits, and rounding control bits needed for compliance with the IEEE 754-1985 standard. |
| GPRs | User | The 32 GPRs contain the address and data arguments addressed from source or destination fields in integer instructions. Also, floating-point load-and-store instructions use GPRs to address memory. |
| MSR | Supervisor | The Machine State Register (MSR) defines the processor state. Its contents are saved when an exception is taken and restored when exception handling completes. The 750GX implements MSR[POW], defined by the architecture as optional, which is used to enable the power management feature. The 750GX-specific MSR[PM] bit is used to mark a process for the performance monitor. |
| SR0–SR15 | Supervisor | The sixteen 32-bit Segment Registers (SRs) define the 4-GB space as sixteen 256-MB segments.The 750GX implements Segment Registers as two arrays—a main array for data accesses and a shadow array for instruction accesses (see *Figure 1-1* on page 25). Loading a segment entry with the Move-to Segment Register (**mtsr**) instruction loads both arrays. The **mfsr** instruction reads the master register, shown as part of the data MMU in *Figure 1-1* on page 25. |

The OEA defines numerous Special-Purpose Registers that serve a variety of functions, such as providing controls, indicating status, configuring the processor, and performing special operations. During normal execution, a program can access the registers shown in *Figure 2-1* on page 58, depending on the program's access privilege (supervisor or user, determined by the privilege-level (PR) bit in the MSR). GPRs and FPRs are accessed through operands that are defined in the instructions. Access to registers can be explicit (that is, through the use of specific instructions for that purpose such as Move-to Special-Purpose Register (**mtspr**) and Move-from Special-Purpose Register (**mfspr**) instructions) or implicit, as the part of the execution of an instruction. Some registers can be accessed both explicitly and implicitly.

In the 750GX, all SPRs are 32 bits wide. *Table 1-2* describes the architecture-defined SPRs implemented by the 750GX. In the *PowerPC Microprocessor Family: The Programming Environments Manual*, these registers are described in detail, including bit descriptions. *Section 2.1.1, Register Set,* on page 57 describes how these registers are implemented in the 750GX. In particular, that section describes those features defined as optional in the PowerPC Architecture that are implemented on the 750GX.

*Table 1-2. Architecture-Defined SPRs Implemented* (Page 1 of 2)

| Register | Level | Function |
|---|---|---|
| LR | User | The Link Register (LR) can be used to provide the branch target address and to hold the return address after branch and link instructions. |
| BATs | Supervisor | The architecture defines eight Block Address Translation Registers (BATs), each implemented as a pair of 32-bit SPRs. In the 750GX, the BAT facility has been extended to include 16 BATs (32 total SPRs), eight for instruction translation and eight for data translation. BATs are used to define and configure blocks of memory. |
| CTR | User | The Count Register (CTR) is decremented and tested by branch-and-count instructions. |
| DABR | Supervisor | The optional Data Address Breakpoint Register (DABR) supports the data address breakpoint facility. |
| DAR | User | The Data Address Register (DAR) holds the address of an access after an alignment or data-storage interrupt (DSI) exception. |
| DEC | Supervisor | The Decrementer Register (DEC) is a 32-bit decrementing counter that provides a way to schedule time-delayed exceptions. |
| DSISR | User | The Data Storage Interrupt Status Register (DSISR) defines the cause of data access and alignment exceptions. |
| EAR | Supervisor | The External Access Register (EAR) controls access to the external access facility through the External Control In Word Indexed (**eciwx**) and External Control Out Word Indexed (**ecowx**) instructions. |
| PVR | Supervisor | The Processor Version Register (PVR) is a read-only register that identifies the processor version and revision level. |
| SDR1 | Supervisor | Storage Description Register 1 (SDR1) specifies the page table address and size used in virtual-to-physical page-address translation. |
| SRR0 | Supervisor | The Machine Status Save/Restore Register 0 (SRR0) saves the address used for restarting an interrupted program when an **rfi** instruction executes (also known as exceptions). |
| SRR1 | Supervisor | The Machine Status Save/Restore Register 1 (SRR1) is used to save machine status on exceptions and to restore machine status when an **rfi** instruction is executed. |

*Table 1-2. Architecture-Defined SPRs Implemented* (Page 2 of 2)

| Register | Level | Function |
|---|---|---|
| SPRG0–SPRG3 | Supervisor | The general-purpose SPRs (SPRG0–SPRG3) are provided for operating system use. |
| TB | User: read<br>Supervisor: read/write | The Time Base Register (TB) is a 64-bit register that maintains the time and date variable. The TB consists of two 32-bit fields—time-base upper (TBU) and time-base lower (TBL). |
| XER | User | The Integer Exception Register (XER) contains the summary overflow bit, integer carry bit, overflow bit, and a field specifying the number of bytes to be transferred by a Load String Word Indexed (**lswx**) or Store String Word Indexed (**stswx**) instruction. |

*Table 1-3* describes the SPRs in 750GX that are not defined by the PowerPC Architecture. *Section 2.1.2, PowerPC 750GX-Specific Registers,* on page 64 gives detailed descriptions of these registers, including bit descriptions.

*Table 1-3. Implementation-Specific Registers*

| Register | Level | Function |
|---|---|---|
| HID0 | Supervisor | The Hardware-Implementation-Dependent Register 0 (HID0) provides checkstop enables and other functions. |
| HID1 | Supervisor | The Hardware-Implementation-Dependent Register 1 (HID1) controls the dual PLLs. |
| HID2 | Supervisor | The Hardware-Implementation-Dependent Register 2 (HID2) provides control and status of special cache-related parity functions. |
| IABR | Supervisor | The Instruction Address Breakpoint Register (IABR) supports instruction address breakpoint exceptions. It can hold an address to compare with instruction addresses in the IQ. An address match causes an instruction address breakpoint exception. |
| ICTC | Supervisor | The Instruction Cache-Throttling Control Register (ICTC) has bits for controlling the interval at which instructions are fetched into the instruction buffer in the instruction unit. This helps control the 750GX’s overall junction temperature. |
| L2CR | Supervisor | The L2 Cache Control Register (L2CR) is used to configure and operate the L2 cache. |
| MMCR0–MMCR1 | Supervisor | The Monitor Mode Control Registers (MMCR0–MMCR1) are used to enable various performance monitoring interrupt functions. UMMCR0–UMMCR1 provide user-level read access to MMCR0–MMCR1. |
| PMC1–PMC4 | Supervisor | The Performance-Monitor Counter Registers (PMC1–PMC4) are used to count specified events. UPMC1–UPMC4 provide user-level read access to these registers. |
| SIA | Supervisor | The Sampled Instruction Address Register (SIA) holds the EA of an instruction executing at or around the time the processor signals the performance-monitor interrupt condition. The USIA register provides user-level read access to the SIA. |
| THRM1, THRM2 | Supervisor | THRM1 and THRM2 provide a way to compare the junction temperature against two user-provided thresholds. The thermal assist unit (TAU) can be operated so that the thermal sensor output is compared to only one threshold, selected in THRM1 or THRM2. |
| THRM3 | Supervisor | THRM3 is used to enable the TAU and to control the output sample time. |
| THRM4 | Supervisor | THRM4 provides the temperature offset to junction temperature for accurate operation of the thermal assist unit. |
| UMMCR0–UMMCR1 | User | The User Monitor Mode Control Registers (UMMCR0–UMMCR1) provide user-level read access to MMCR0–MMCR1. |
| UPMC1–UPMC4 | User | The User Performance-Monitor Counter Registers (UPMC1–UPMC4) provide user-level read access to PMC1–PMC4. |
| USIA | User | The User Sampled Instruction Address Register (USIA) provides user-level read access to the SIA register. |

## 1.5 Instruction Set

All PowerPC instructions are encoded as single-word (32-bit) instructions. Instruction formats are consistent among all instruction types (the primary operation code is always 6 bits, register operands are always specified in the same bit fields in the instruction), permitting efficient decoding to occur in parallel with operand accesses. This fixed instruction length and consistent format greatly simplify instruction pipelining.

For more information, see *Chapter 2, Programming Model,* on page 57.

### 1.5.1 PowerPC Instruction Set

The PowerPC instructions are divided into the following categories.

- Integer instructions—These include computational and logical instructions.
  - Integer arithmetic instructions
  - Integer compare instructions
  - Integer logical instructions
  - Integer rotate and shift instructions
- Floating-point instructions—These include floating-point computational instructions, as well as instructions that affect the FPSCR.
  - Floating-point arithmetic instructions
  - Floating-point multiply/add instructions
  - Floating-point rounding and conversion instructions
  - Floating-point compare instructions
  - Floating-point status and control instructions
- Load/store instructions—These include integer and floating-point load-and-store instructions.
  - Integer load-and-store instructions
  - Integer load-and-store multiple instructions
  - Floating-point load and store
  - Primitives used to construct atomic memory operations (Load Word and Reserve Indexed [**lwarx**] and Store Word Conditional Indexed [**stwcx.**] instructions)
- Flow-control instructions—These include branching instructions, Condition Register logical instructions, trap instructions, and other instructions that affect the instruction flow.
  - Branch and trap instructions
  - Condition Register logical instructions (sets conditions for branches)
  - System call
- Processor control instructions—These instructions are used to synchronize memory accesses and to manage caches, TLBs, and the Segment Registers.
  - Move-to/Move-from SPR instructions
  - Move-to/Move-from MSR
  - Synchronize (processor and memory system)
  - Instruction synchronize
  - Order loads and stores
- Memory control instructions—To provide control of caches, TLBs, and SRs.
  - Supervisor-level cache-management instructions
  - User-level cache instructions
  - Segment Register manipulation instructions

- Translation-lookaside-buffer management instructions

These categories do not indicate the execution unit that executes a particular instruction or group of instructions.

Integer instructions operate on byte, half-word, and word operands. Floating-point instructions operate on single-precision (one word) and double-precision (two words) floating-point operands. The PowerPC Architecture uses instructions that are four bytes long and word-aligned. It provides for integer byte, half-word, and word operand loads and stores between memory and a set of 32 GPRs. It also provides for single and double-precision loads and stores between memory and a set of 32 Floating Point Registers (FPRs).

Computational instructions do not access memory. To use a memory operand in a computation and then modify the same or another memory location, the memory contents must be loaded into a register, modified, and then written back to the target location using three or more instructions.

PowerPC processors follow the program flow when they are in the normal execution state; however, the flow of instructions can be interrupted directly by the execution of an instruction or by an asynchronous event. Either type of exception will cause the associated exception handler to be invoked.

Effective address computations for both data and instruction accesses use 32-bit signed two’s complement binary arithmetic. A carry from bit 0 and overflow are ignored.

### 1.5.2 750GX Microprocessor Instruction Set

750GX instruction set is defined as follows.

- 750GX provides hardware support for all PowerPC instructions.
- 750GX implements the following instructions, which are optional in the PowerPC Architecture.
  - External Control In Word Indexed (**eciwx**).
  - External Control Out Word Indexed (**ecowx**).
  - Floating Select (**fsel**).
  - Floating Reciprocal Estimate Single-Precision (**fres**).
  - Floating Reciprocal Square Root Estimate (**frsqrte**).
  - Store Floating-Point as Integer Word (**stfiw**).

**Note:** The **fres** and **frsqrte** instructions are implemented in the 750GX with 12-bit precision (better than one part in 4000), which significantly exceeds the minimum precision required by the architecture.

## 1.6 On-Chip Cache Implementation

The following subsections describe the PowerPC Architecture’s treatment of cache in general, and the 750GX-specific implementation. A detailed description of the 750GX L1 cache implementation is provided in *Chapter 3, Instruction-Cache and Data-Cache Operation,* on page 121. A detailed description of the L2 cache is provided in *Chapter 9, L2 Cache,* on page 323.

### 1.6.1 PowerPC Cache Model

The PowerPC Architecture does not define hardware aspects of cache implementations. For example, PowerPC processors can have unified caches, separate instruction and data caches (Harvard architecture), or no cache at all. PowerPC microprocessors control the following memory-access modes on a virtual-page or block (BAT) basis

- Write-back/write-through mode
- Caching-inhibited mode
- Memory coherency

The caches are physically addressed, and the data cache can operate in either write-back or write-through mode, as specified by the PowerPC Architecture.

The PowerPC Architecture defines the term ‘cache block’ as the cacheable unit. The VEA and OEA define cache-management instructions that a programmer can use to affect cache contents.

### 1.6.2 750GX Microprocessor Cache Implementation

750GX cache implementation is described in *Section 1.2.4, On-Chip Level 1 Instruction and Data Caches,* on page 33 and *Section 1.2.5, On-Chip Level 2 Cache Implementation,* on page 35.

The BPU also contains a cache, the 64-entry BTIC, that provides immediate access to an instruction pair for taken branches. For more information, see *Section 1.2.1.2, Branch Processing Unit (BPU),* on page 29.

## 1.7 Exception Model

The following sections describe the PowerPC exception model and the 750GX implementation. A detailed description of the 750GX exception model is provided in *Chapter 4, Exceptions,* on page 151 in this manual.

### 1.7.1 PowerPC Exception Model

The PowerPC exception model allows the processor to interrupt the instruction flow to handle certain situations caused by external signals, errors, or unusual conditions arising from the instruction execution. When exceptions occur, information about the state of the processor is saved to certain registers, and the processor begins execution at an address (exception vector) predetermined for each exception. System software must complete the saving of the processor state prior to servicing the exception. Exception processing proceeds in supervisor mode.

Although multiple exception conditions can map to a single exception vector, a more specific condition can be determined by examining a register associated with the exception. For example, the MSR, DSISR, and FPSCR contain status bits that further identify the exception condition. Additionally, some exception conditions can be explicitly enabled or disabled by software.

The PowerPC Architecture requires that exceptions be handled in specific priority and program order. Therefore, although a particular implementation might recognize exception conditions out of order, they are handled in program order. When an instruction-caused exception is recognized, any unexecuted instructions that appear earlier in the instruction stream, including any that are not dispatched, must complete before the exception is taken. Any exceptions those instructions cause must also be handled first. Likewise, asynchronous, precise exceptions are recognized when they occur. However, they are not handled until the instructions currently in the completion queue successfully retire or generate an exception, and the completion queue is emptied.

Unless a catastrophic condition causes a system reset or machine-check exception, only one exception is handled at a time. For example, if one instruction encounters multiple exception conditions, those conditions are handled sequentially in priority order. After the exception handler completes, the instruction processing continues until the next exception condition is encountered. Recognizing and handling exception conditions sequentially guarantees system integrity.

When an exception is taken, information about the processor state before the exception was taken is saved in SRR0 and SRR1. Exception handlers must save the information stored in SRR0 and SRR1 early to prevent the program state from being lost due to a system reset and machine-check exception or due to an instruction-caused exception in the exception handler, and before re-enabling external interrupts. The exception handler must also save and restore any GPR registers used by the handler.

The PowerPC Architecture supports four types of exceptions:

Synchronous, precise
: These are caused by instructions. All instruction-caused exceptions are handled precisely. That is, the machine state at the time the exception occurs is known and can be completely restored. This means that (excluding the trap and system call exceptions) the address of the faulting instruction is provided to the exception handler and that neither the faulting instruction nor subsequent instructions in the code stream will complete execution before the exception is taken. Once the exception is processed, execution resumes at the address of the faulting instruction (or at an alternate address provided by the exception handler). When an exception is taken due to a trap or system call instruction, execution resumes at an address provided by the handler.

Synchronous, imprecise
: The PowerPC Architecture defines two imprecise floating-point exception modes, recoverable and nonrecoverable. Even though the 750GX provides a means to enable the imprecise modes, it implements these modes identically to the precise mode (that is, enabled floating-point exceptions are always precise).

Asynchronous, maskable
: The PowerPC Architecture defines external and decrementer interrupts as maskable, asynchronous exceptions. When these exceptions occur, their handling is postponed until the next instruction, and any exceptions associated with that instruction completes execution. If no instructions are in the execution units, the exception is taken immediately upon determination of the correct restart address (for loading SRR0). As shown in the *Table 1-4, 750GX Microprocessor Exception Classifications,* the 750GX implements additional asynchronous, maskable exceptions.

Asynchronous, nonmaskable
: There are two nonmaskable asynchronous exceptions: system reset and the machine-check exception. These exceptions might not be recoverable, or might provide a limited degree of recoverability. Exceptions report recoverability through the MSR[RI] bit.

### 1.7.2 750GX Microprocessor Exception Implementation

The 750GX exception classes described above are shown in the *Table 1-4*. Although exceptions have other characteristics, such as priority and recoverability, *Table 1-4* describes the precise or imprecise characteristics of exceptions the 750GX uniquely handles. *Table 1-4* includes no synchronous imprecise exceptions; although the PowerPC Architecture supports imprecise handling of floating-point exceptions, the 750GX implements these exception modes precisely.

*Table 1-4. 750GX Microprocessor Exception Classifications*

| Synchronous/Asynchronous | Precise/Imprecise | Exception Type |
|---|---|---|
| Asynchronous, nonmaskable | Imprecise | Machine check, system reset |
| Asynchronous, maskable | Precise | External, decrementer, system-management, performance-monitor, and thermal-management interrupts |
| Synchronous | Precise | Instruction-caused exceptions |

*Table 1-5* on page 50 lists the 750GX exceptions and conditions that cause them. Exceptions specific to the 750GX are indicated.

*Table 1-5. Exceptions and Conditions*

| Exception Type | Vector Offset (hex) | Causing Conditions |
|---|---|---|
| Reserved | 00000 | — |
| System reset | 00100 | Assertion of either $\overline{\text{HRESET}}$ or $\overline{\text{SRESET}}$ or a power-on reset. |
| Machine check | 00200 | Assertion of the transfer error acknowledge ($\overline{\text{TEA}}$) during a data-bus transaction, assertion of a machine-check interrupt (MCP), an address, data or L2 double-bit error. MSR[ME] must be set. |
| Data storage interrupt | 00300 | As defined in the PowerPC Architecture (for example, a page fault occurs). |
| Instruction storage interrupt (ISI) | 00400 | As defined by the PowerPC Architecture (for example, a page fault occurs). |
| External interrupt | 00500 | MSR[EE] = 1 and interrupt ($\overline{\text{INT}}$) is asserted. |
| Alignment | 00600 | • A floating-point load/store, Store Multiple Word (**stmw**), Store Word Conditional Indexed (**stwcx.**), Load Multiple Word (**lmw**), Load Word and Reserved Indexed (**lwarx**), **eciwx**, or **ecowx** instruction operand is not word-aligned.<br>• A multiple/string load/store operation is attempted in little-endian mode.<br>• The operand of Data Cache Block Zero (**dcbz**) is in memory that is write-through-required or caching-inhibited, or the cache is disabled. |
| Program | 00700 | As defined by the PowerPC Architecture. |
| Floating-point unavailable | 00800 | As defined by the PowerPC Architecture. |
| Decrementer | 00900 | As defined by the PowerPC Architecture, when the most significant bit of the DEC register changes from 0 to 1 and MSR[EE] = 1. |
| Reserved | 00A00–00BFF | — |
| System call | 00C00 | Execution of the System Call (**sc**) instruction. |
| Trace | 00D00 | MSR[SE] = 1 or a branch instruction completes and MSR[BE] = 1. Unlike the architecture definition, Instruction Synchronization (**isync)** does not cause a trace exception |
| Reserved | 00E00 | The 750GX does not generate an exception to this vector. Other PowerPC processors might use this vector for floating-point assist exceptions. |
| Reserved | 00E10–00EFF | — |
| Performance monitor[1] | 00F00 | The limit specified in a Performance-Monitor Control (PMC) register is reached and MMCR0[ENINT] = 1. |
| Instruction address breakpoint[1] | 01300 | IABR[0–29] matches EA[0–29] of the next instruction to complete,<br>IABR[TE] matches MSR[IR], and<br>IABR[BE] = 1. |
| System management exception | 01400 | A system management exception is enabled if MSR[EE] = 1 and is signaled to the 750GX by the assertion of an input signal pin (SMI). |
| Reserved | 01500–016FF | — |
| Thermal-management interrupt[1] | 01700 | Thermal management is enabled, the junction temperature exceeds the threshold specified in THRM1 or THRM2, and MSR[EE] = 1. |
| Reserved | 01800–02FFF | — |

[1] 750GX-specific

## 1.8 Memory Management

The following subsections describe the memory-management features of the PowerPC Architecture, and the 750GX implementation. A detailed description of the 750GX MMU implementation is provided in *Chapter 5, Memory Management,* on page 179.

### 1.8.1 PowerPC Memory-Management Model

The primary functions of the MMU are to translate logical (effective) addresses to physical addresses for memory accesses and to provide access protection on blocks and pages of memory. There are two types of accesses generated by the 750GX that require address translation—instruction fetches, and data accesses to memory generated by load, store, and cache-control instructions.

The PowerPC Architecture defines different resources for 32-bit and 64-bit processors. The 750GX implements the 32-bit memory-management model. The memory management unit provides two types of memory-access models: block-address translate (BAT) model and a virtual address model. The BAT block sizes range from 128 KB to 256 MB, are selectable from high-order effective address bits, and have priority over the virtual model. The virtual model employs a 52-bit virtual address space made up of a 24-bit segment address space and a 28-bit effective address space. The virtual model uses a demand paging method with a 4-KB page size. In both models, address translation is done completely by hardware, in parallel with cache accesses, with no additional cycles incurred.

The 750GX MMU provides independent 8-entry BAT arrays for instructions and data that maintain address translations for blocks of memory. These entries define blocks that can vary from 128 KB to 256 MB. The BAT arrays are maintained by system software. Instructions and data share the same virtual address model, but could operate in separate segment spaces.

The PowerPC 750GX MMU and exception model support demand-paged virtual memory. Virtual memory management permits execution of programs larger than the size of physical memory. Demand-paged implies that individual pages for data and instructions are loaded into physical memory from the system disk only when they are required by an executing program. Infrequently used pages in memory are returned to disk or discarded if they have not been modified.

The hashed page table is a fixed-sized data structure[1] that contains 8-byte page table entries (PTEs), which define the mapping between virtual pages and physical pages. The page table size is a power of two and is boundary aligned in memory based on the size of the table. The page table contains a number of page-table-entry groups (PTEGs). Since a PTEG contains eight PTEs of eight bytes each, each PTEG is 64 bytes long. PTEG addresses are entry points for table-search operations. A given page translation can be found in one of two possible PTEGs. The size and location in memory of the page table is defined in the SDR1 register.

Setting MSR[IR] enables instruction address translations and setting MSR[DR] enables data address translations. If the bit is cleared, the respective effective address is used as the physical address.

[1] Size should be determined by the amount of physical memory available to the system.

### 1.8.2 750GX Microprocessor Memory-Management Implementation

The 750GX implements separate MMUs for instructions and data. It implements a copy of the Segment Registers in the instruction MMU. However, read and write accesses (Move-from Segment Register [**mfsr**] and Move-to Segment Register [**mtsr**]) are handled through the Segment Registers implemented as part of the data MMU. The 750GX MMU is described in *Section 1.2.3, Memory Management Units (MMUs),* on page 32.

The R (referenced) bit is set in the PTE in memory during a page table search due to a TLB miss. Updates to the changed (C) bit are treated like TLB misses. The page table is searched again to find the correct PTE to update when the C bit changes from 0 to 1.

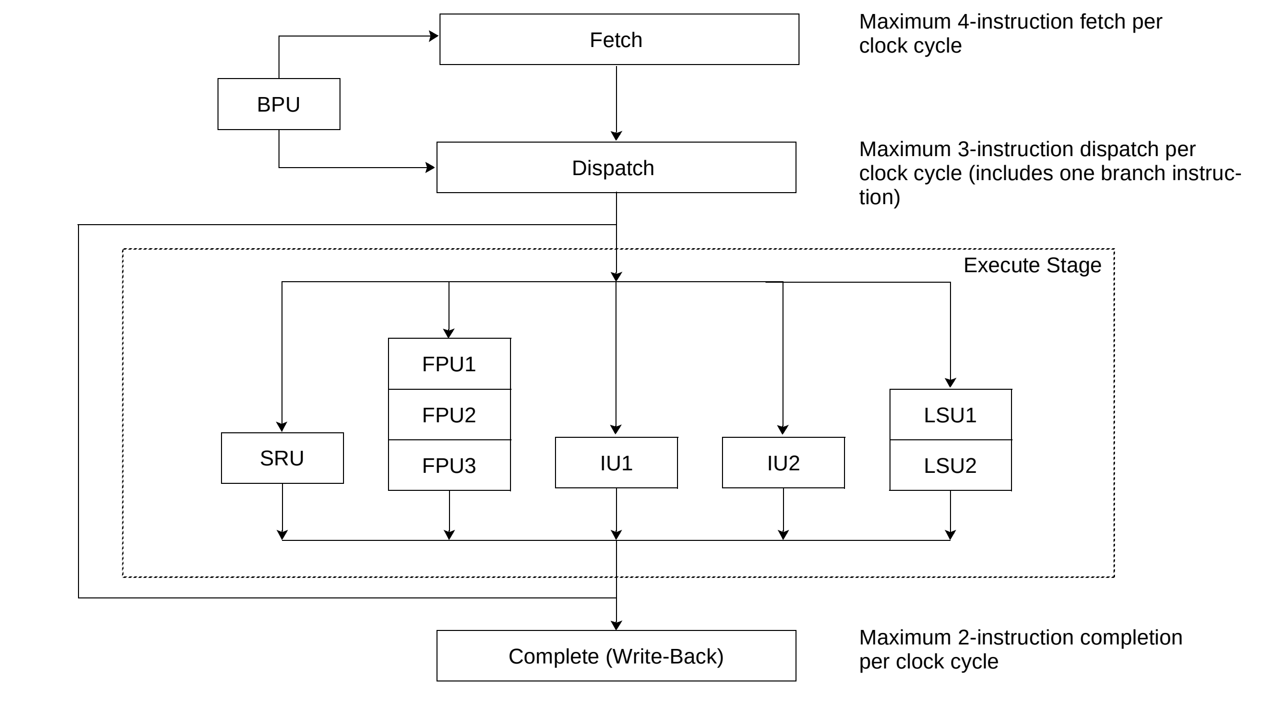
## 1.9 Instruction Timing

The 750GX is a pipelined, superscalar processor. A pipelined processor is one in which instruction processing is divided into discrete stages, allowing work to be done on multiple instructions in each stage. For example, after an instruction completes one stage, it can pass on to the next stage leaving the previous stage available to a subsequent instruction. This improves overall instruction throughput.

A superscalar processor is one that issues multiple independent instructions to separate execution units in a single cycle, allowing multiple instructions to execute in parallel. The 750GX has six independent execution units, two for integer instructions, and one each for floating-point instructions, branch instructions, load-and-store instructions, and system-register instructions. Having separate GPRs and FPRs allows integer, floating-point calculations, and load-and-store operations to occur simultaneously without interference. Additionally, rename buffers are provided to allow operations to post completed results for use by subsequent instructions without committing them to the architected FPR and GPR register files.

As shown in *Figure 1-5* on page 53, the common pipeline of the 750GX has four stages through which all instructions must pass—fetch, decode/dispatch, execute, and complete/write back. Instructions flow sequentially through each stage. However, at dispatch, a position is made available in the completion queue at the same time it enters the execution stage. This simplifies the completion operation when instructions are retired in program order. Both the load/store and floating-point units have multiple stages to execute their instructions. An instruction occupies only one stage at a time in all execution units. At each stage, an instruction might proceed without delay or might stall. Stalls are caused by the requirement for additional processing or other events. For example, divide instructions require multiple cycles to complete the operation; load-and-store instructions might stall waiting for address translation (during TLB reload or page fault, for example).

*Figure 1-5. Pipeline Diagram*


Fetch
BPU
Dispatch
Maximum 4-instruction fetch per clock cycle
Maximum 3-instruction dispatch per clock cycle (includes one branch instruction)
Execute Stage
FPU1
FPU2
FPU3
SRU
IU1
IU2
LSU1
LSU2
Complete (Write-Back)
Maximum 2-instruction completion per clock cycle

**Note:** *Figure 1-5* does not show features such as reservation stations and rename buffers that reduce stalls and improve instruction throughput.

The instruction pipeline in the 750GX has four major pipeline stages. They are fetch, dispatch, execute, and complete:

- The fetch pipeline stage primarily involves fetching instructions from the memory system and keeping the instruction queue full. The BPU decodes branches after they are fetched and removes (folds out) those that do not update the CTR or LR from the instruction stream. If the branch is taken or predicted as taken, the fetch unit is informed of the new address and fetching resumes along the taken path. For branches not taken or predicted as not taken, sequential fetching continues.
- The dispatch unit is responsible for taking instructions from the bottom two locations of the instruction queue and delivering them to an execution unit for further processing. Dispatch is responsible for decoding the instructions and determining which instructions can be dispatched. To qualify for dispatch, a reservation station, a rename buffer, and a position in the completion queue all must be available. A branch instruction could be processed by the BPU on the same clock cycle for a maximum of three instructions dispatched per cycle.

  The dispatch stage accesses operands, assigns a rename buffer for operands that update architected registers, which include the GPRs, FPRs, and CR, and delivers the instruction to the reservation registers of the respective execution units. If a source operand is not available because a previous instruction is updating the item in a rename buffer, dispatch provides a tag that indicates which rename buffer will supply the operand when it becomes available. At the end of the dispatch stage, the instructions are removed from the instruction queue, latched into reservation stations at the appropriate execution unit, and assigned positions in the completion buffers in sequential program order.

- The execution units process instructions from their reservation stations using the operands provided from dispatch, and notifies the completion stage when the instruction has finished execution. With the exception of multiply and divide, integer instructions complete execution in a single cycle.

  The FPU has three stages (multiply, add, and normalize) for processing floating-point arithmetic. All single-precision arithmetic (add, subtract, multiply, and multiply/add) instructions are processed without stalls at each stage. They have a 1-cycle throughput and a 3-cycle latency. Three different arithmetic instructions can be in the execution unit at one time, with one instruction completing execution each cycle. Double-precision arithmetic multiply requires two cycles in the multiply stage, one cycle in the add stage, and one cycle in the normalize stage, which yields a 2-cycle throughput and a 4-cycle latency. All divide instructions require multiple cycles in the first stage for processing.

  The load/store unit has two reservation registers and two pipeline stages. The first stage is for effective address calculation and the second stage is for MMU translation and accessing the L1 data cache. Load instructions have a 1-cycle throughput and a 2-cycle latency.

  In the case of an internal exception, the execution unit reports the exception to the completion pipeline stage and (except for the FPU) discontinues instruction execution until the exception is handled. The exception is not signaled until it is determined that all previous instructions have completed to a point where they will not signal an exception.
- The completion unit retires instructions from the bottom two positions of the completion queue in program order. This maintains the correct architectural machine state and transfers execution results from the rename buffers to the GPRs and FPRs (and CTR and LR, for some instructions) as instructions are retired. If the completion logic detects an instruction causing an exception, all subsequent instructions are cancelled, their execution results in rename buffers are discarded, and instructions are fetched from the appropriate exception vector.

Because the PowerPC Architecture can be applied to such a wide variety of implementations, instruction timing varies among PowerPC processors. For a detailed discussion of instruction timing with examples and a table of latencies for each execution unit, see *Chapter 6, Instruction Timing,* on page 209.

## 1.10 Power Management

The 750GX provides the following four power modes, selectable by setting the appropriate control bits in the MSR and HID0 registers:

Full-power
This is the default power state of the 750GX. The 750GX is fully powered, and the internal functional units are operating at the full processor clock speed. If the dynamic power management mode is enabled, functional units that are idle will automatically enter a low-power state without affecting performance, software execution, or external hardware.

Doze
All the functional units of the 750GX are disabled except for the Time Base/Decrementer Registers and the bus snooping logic. When the processor is in doze mode, an external asynchronous interrupt, a system management interrupt, a decrementer exception, a hard or soft reset, or a machine check brings the 750GX into the full-power state. The 750GX in doze mode maintains the PLL in a fully powered state and locked to the system external clock input (SYSCLK) so a transition to the full-power state takes only a few processor clock cycles.

Nap

The nap mode further reduces power consumption by disabling bus snooping, leaving only the Time Base Register and the PLL in a powered state. The 750GX returns to the full-power state upon receipt of an external asynchronous interrupt, a system management interrupt, a decrementer exception, a hard or soft reset, or a machine-check interrupt ($\overline{\text{MCP}}$). A return to full-power state from nap state takes only a few processor clock cycles. When the processor is in nap mode, if $\overline{\text{QACK}}$ is negated, the processor is put in doze mode to support snooping.

Sleep

Sleep mode minimizes power consumption by disabling all internal functional units, after which external system logic can disable the PLL and SYSCLK. Returning the 750GX to the full-power state requires enabling the PLL and SYSCLK, followed by the assertion of an external asynchronous interrupt, a system management interrupt, a hard or soft reset, or a machine-check interrupt ($\overline{\text{MCP}}$) signal after the time required to relock the PLL.

In addition, the 750GX allows software-controlled toggling between two operating frequencies. During periods of processor inactivity or for applications requiring reduced computing performance, the processor may be toggled to a lower frequency to conserve power.

*Chapter 10, Power and Thermal Management,* on page 335 provides information about power-saving and thermal-management modes for the 750GX.

## 1.11 Thermal Management

The 750GX’s thermal assist unit (TAU) provides a way to control heat dissipation. This ability is particularly useful in portable computers, which, due to power consumption and size limitations, cannot use desktop cooling solutions such as fans. Therefore, better heat sink designs coupled with intelligent thermal management is of critical importance for high-performance portable systems.

Primarily, the thermal-management system monitors and regulates the system’s operating temperature. For example, if the temperature is about to exceed a set limit, the system can be made to slow down or even suspend operations temporarily in order to lower the temperature.

The thermal-management facility also ensures that the processor’s junction temperature does not exceed the operating specification. To avoid the inaccuracies that arise from measuring junction temperature with an external thermal sensor, the 750GX’s on-chip thermal sensor and logic tightly couple the thermal-management implementation.

The TAU consists of a thermal sensor, digital-to-analog convertor, comparator, control logic, and the dedicated SPRs described in *Section 1.4, PowerPC Registers and Programming Model,* on page 42. The TAU does the following.

- Compares the junction temperature against user-programmable thresholds.
- Generates a thermal-management interrupt if the temperature crosses the threshold.
- Enables the user to estimate the junction temperature by using a software successive approximation routine.

The TAU is controlled through the privileged **mtspr** and **mfspr** instructions to the four SPRs provided for configuring and controlling the sensor control logic. The SPRs function as follows.

- THRM1 and THRM2 provide the ability to compare the junction temperature against two user-provided thresholds. Having dual thresholds gives the thermal-management software finer control of the junction temperature. In single-threshold mode, the thermal sensor output is compared to only one threshold in either THRM1 or THRM2.
- THRM3 is used to enable the TAU and to control the comparator output sample time. The thermal-management logic manages the thermal-management interrupt generation and time multiplexed comparisons in the dual-threshold mode, as well as other control functions.
- THRM4 is used to improve accuracy in determining the actual junction temperature.

Instruction-cache throttling provides control of the 750GX’s overall junction temperature by determining the interval at which instructions are fetched. This feature is accessed through the ICTC register. *Chapter 10, Power and Thermal Management,* on page 335 provides information about power-saving and thermal-management modes for the 750GX.

## 1.12 Performance Monitor

The 750GX incorporates a performance-monitor facility that system designers can use to help bring up, debug, and optimize software performance. The performance monitor counts events during execution of code, which relate to dispatch, execution, completion, and memory accesses.

The performance monitor incorporates several registers that can be read and written to by supervisor-level software. User-level versions of these registers provide read-only access for user-level applications. These registers are described in *Section 1.4, PowerPC Registers and Programming Model,* on page 42. Performance-Monitor Control Registers, MMCR0 or MMCR1, can be used to specify which events are to be counted and the conditions for which a performance-monitoring interrupt is taken. Additionally, the Sampled Instruction Address Register, SIA (USIA), holds the address of the first instruction to complete after the counter overflowed.

Attempting to write to a user-read-only Performance-Monitor Register causes a program exception, regardless of the MSR[PR] setting. When a performance-monitoring interrupt occurs, program execution continues from vector offset 0x00F00.

*Chapter 11, Performance Monitor and System Related Features,* on page 349 describes the operation of the performance-monitor diagnostic tool incorporated in the 750GX.

# 2. Programming Model

This chapter describes the 750GX programming model, emphasizing those features specific to the 750GX processor and summarizing those that are common to PowerPC processors. It consists of three major sections, which describe the following topics.

- Registers implemented in the 750GX
- Operand conventions
- 750GX instruction set

For detailed information about architecture-defined features, see the *PowerPC Microprocessor Family: The Programming Environments Manual*.

## 2.1 PowerPC 750GX Processor Register Set

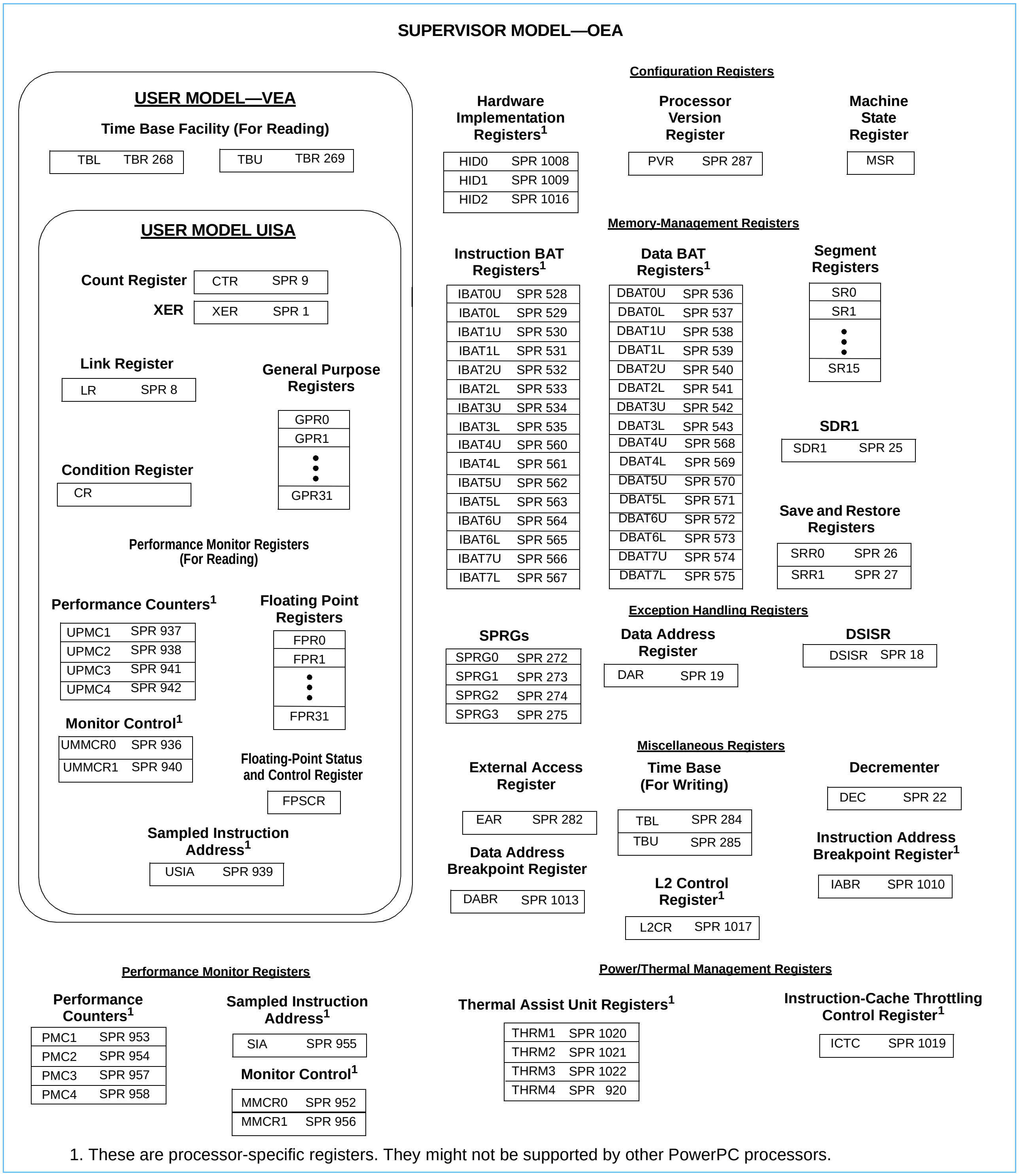
This section describes the registers implemented in the 750GX. It includes an overview of registers defined by the PowerPC Architecture, highlighting differences in how these registers are implemented in the 750GX, and a detailed description of 750GX-specific registers. Full descriptions of the architecture-defined register set are provided in Chapter 2, “PowerPC Register Set” in the *PowerPC Microprocessor Family: The Programming Environments Manual*.

Registers are defined at all three levels of the PowerPC Architecture—user instruction set architecture (UISA), virtual environment architecture (VEA), and operating environment architecture (OEA). The PowerPC Architecture defines register-to-register operations for all computational instructions. Source data for these instructions are accessed from the on-chip registers or are provided as immediate values embedded in the opcode. The 3-register instruction format allows specification of a target register distinct from the two source registers, thus preserving the original data for use by other instructions and reducing the number of instructions required for certain operations. Data is transferred between memory and registers with explicit load-and-store instructions only.

### 2.1.1 Register Set

The registers implemented on the 750GX are shown in *Figure 2-1* on page 58. The number to the right of the special-purpose registers (SPRs) indicates the number that is used in the syntax of the instruction operands to access the register (for example, the number used to access the Integer Exception Register (XER) is SPR 1). These registers can be accessed using the Move-to Special Purpose Register (**mtspr**) and Move-from Special Purpose Register (**mfspr**) instructions.

*Figure 2-1. PowerPC 750GX Microprocessor Programming Model—Registers*



**SUPERVISOR MODEL—OEA**

**USER MODEL—VEA**

**Time Base Facility (For Reading)**

| | |
|---|---|
| TBL | TBR 268 |
| TBU | TBR 269 |

**USER MODEL UISA**

| | | |
|---|---|---|
| Count Register | CTR | SPR 9 |
| XER | XER | SPR 1 |

**Link Register**

| | |
|---|---|
| LR | SPR 8 |

**General Purpose Registers**

GPR0
GPR1
•
•
•
GPR31

**Condition Register**

CR

**Performance Monitor Registers (For Reading)**

**Performance Counters[1]**

| | |
|---|---|
| UPMC1 | SPR 937 |
| UPMC2 | SPR 938 |
| UPMC3 | SPR 941 |
| UPMC4 | SPR 942 |

**Floating Point Registers**

FPR0
FPR1
•
•
•
FPR31

**Monitor Control[1]**

| | |
|---|---|
| UMMCR0 | SPR 936 |
| UMMCR1 | SPR 940 |

**Floating-Point Status and Control Register**

FPSCR

**Sampled Instruction Address[1]**

| | |
|---|---|
| USIA | SPR 939 |

**Configuration Registers**

**Hardware Implementation Registers[1]**

| | |
|---|---|
| HID0 | SPR 1008 |
| HID1 | SPR 1009 |
| HID2 | SPR 1016 |

**Processor Version Register**

| | |
|---|---|
| PVR | SPR 287 |

**Machine State Register**

MSR

**Memory-Management Registers**

**Instruction BAT Registers[1]**

| | |
|---|---|
| IBAT0U | SPR 528 |
| IBAT0L | SPR 529 |
| IBAT1U | SPR 530 |
| IBAT1L | SPR 531 |
| IBAT2U | SPR 532 |
| IBAT2L | SPR 533 |
| IBAT3U | SPR 534 |
| IBAT3L | SPR 535 |
| IBAT4U | SPR 560 |
| IBAT4L | SPR 561 |
| IBAT5U | SPR 562 |
| IBAT5L | SPR 563 |
| IBAT6U | SPR 564 |
| IBAT6L | SPR 565 |
| IBAT7U | SPR 566 |
| IBAT7L | SPR 567 |

**Data BAT Registers[1]**

| | |
|---|---|
| DBAT0U | SPR 536 |
| DBAT0L | SPR 537 |
| DBAT1U | SPR 538 |
| DBAT1L | SPR 539 |
| DBAT2U | SPR 540 |
| DBAT2L | SPR 541 |
| DBAT3U | SPR 542 |
| DBAT3L | SPR 543 |
| DBAT4U | SPR 568 |
| DBAT4L | SPR 569 |
| DBAT5U | SPR 570 |
| DBAT5L | SPR 571 |
| DBAT6U | SPR 572 |
| DBAT6L | SPR 573 |
| DBAT7U | SPR 574 |
| DBAT7L | SPR 575 |

**Segment Registers**

SR0
SR1
•
•
•
SR15

**SDR1**

| | |
|---|---|
| SDR1 | SPR 25 |

**Save and Restore Registers**

| | |
|---|---|
| SRR0 | SPR 26 |
| SRR1 | SPR 27 |

**Exception Handling Registers**

**SPRGs**

| | |
|---|---|
| SPRG0 | SPR 272 |
| SPRG1 | SPR 273 |
| SPRG2 | SPR 274 |
| SPRG3 | SPR 275 |

**Data Address Register**

| | |
|---|---|
| DAR | SPR 19 |

**DSISR**

| | |
|---|---|
| DSISR | SPR 18 |

**Miscellaneous Registers**

**External Access Register**

| | |
|---|---|
| EAR | SPR 282 |

**Time Base (For Writing)**

| | |
|---|---|
| TBL | SPR 284 |
| TBU | SPR 285 |

**Decrementer**

| | |
|---|---|
| DEC | SPR 22 |

**Data Address Breakpoint Register**

| | |
|---|---|
| DABR | SPR 1013 |

**L2 Control Register[1]**

| | |
|---|---|
| L2CR | SPR 1017 |

**Instruction Address Breakpoint Register[1]**

| | |
|---|---|
| IABR | SPR 1010 |

**Performance Monitor Registers**

**Performance Counters[1]**

| | |
|---|---|
| PMC1 | SPR 953 |
| PMC2 | SPR 954 |
| PMC3 | SPR 957 |
| PMC4 | SPR 958 |

**Sampled Instruction Address[1]**

| | |
|---|---|
| SIA | SPR 955 |

**Monitor Control[1]**

| | |
|---|---|
| MMCR0 | SPR 952 |
| MMCR1 | SPR 956 |

**Power/Thermal Management Registers**

**Thermal Assist Unit Registers[1]**

| | |
|---|---|
| THRM1 | SPR 1020 |
| THRM2 | SPR 1021 |
| THRM3 | SPR 1022 |
| THRM4 | SPR 920 |

**Instruction-Cache Throttling Control Register[1]**

| | |
|---|---|
| ICTC | SPR 1019 |

1. These are processor-specific registers. They might not be supported by other PowerPC processors.

The PowerPC UISA registers are user-level. General Purpose Registers (GPRs) and Floating Point Registers (FPRs) are accessed through instruction operands. Access to registers can be explicit (by using instructions for that purpose such as **mtspr** and **mfspr** instructions) or implicit as part of the execution of an instruction. Some registers are accessed both explicitly and implicitly.

**Implementation Note:** The 750GX fully decodes the SPR field of the instruction. If the SPR specified is undefined, an illegal instruction program exception occurs.

Descriptions of the PowerPC user-level registers follow:

- **User-level registers** (UISA)—The user-level registers can be accessed by all software with either user or supervisor privileges. They include the following registers:
  - General Purpose Registers (GPRs). The 32 GPRs (GPR0–GPR31) serve as data source or destination registers for integer instructions and provide data for generating addresses. See “General Purpose Registers (GPRs)” in Chapter 2, “PowerPC Register Set” of the *PowerPC Microprocessor Family: The Programming Environments Manual* for more information.
  - Floating Point Registers (FPRs). The 32 FPRs (FPR0–FPR31) serve as the data source or destination for all floating-point instructions. See “Floating Point Registers (FPRs)” in Chapter 2, “PowerPC Register Set” of the *PowerPC Microprocessor Family: The Programming Environments Manual*.
  - Condition Register (CR). The 32-bit CR consists of eight 4-bit fields, CR0–CR7, that reflect results of certain arithmetic operations and provide a mechanism for testing and branching. See “Condition Register (CR)” in Chapter 2, “PowerPC Register Set” of the *PowerPC Microprocessor Family: The Programming Environments Manual*.
  - Floating-Point Status and Control Register (FPSCR). The FPSCR contains all floating-point exception signal bits, exception summary bits, exception enable bits, and rounding control bits needed for compliance with the IEEE 754-1985 standard. See “Floating-Point Status and Control Register (FPSCR)” in Chapter 2, “PowerPC Register Set” of the *PowerPC Microprocessor Family: The Programming Environments Manual*.

  The remaining user-level registers are SPRs. Note that the PowerPC Architecture provides a separate mechanism for accessing SPRs (the **mtspr** and **mfspr** instructions). These instructions are commonly used to explicitly access certain registers, while other SPRs are more typically accessed as the side effect of executing other instructions.

  - Integer Exception Register (XER). The XER indicates overflow and carries for integer operations. See “XER Register (XER)” in Chapter 2, “PowerPC Register Set” of the *PowerPC Microprocessor Family: The Programming Environments Manual* for more information.

    **Implementation Note**: To allow emulation of the Load String and Compare Byte Indexed (**lscbx**) instruction defined by the POWER architecture, XER[16–23] is implemented so that it can be read with **mfspr** and written with Move-to Fixed-Point Exception Register (**mtxer**) instructions.
  - Link Register (LR). The LR provides the branch target address for the Branch Conditional to Link Register (**bclr***x*) instruction, and can be used to hold the logical address of the instruction that follows a branch and link instruction, typically used for linking to subroutines. See “Link Register (LR)” in Chapter 2, “PowerPC Register Set” of the *PowerPC Microprocessor Family: The Programming Environments Manual*.
  - Count Register (CTR). The CTR holds a loop count that can be decremented during execution of appropriately coded branch instructions. The CTR can also provide the branch target address for the Branch Conditional to Count Register (**bcctr***x*) instruction. See “Count Register (CTR)” in Chapter 2,

“PowerPC Register Set” of the *PowerPC Microprocessor Family: The Programming Environments Manual*.

- **User-level registers** (VEA)—The PowerPC VEA defines the time-base facility (TB), which consists of two 32-bit registers—Time Base Upper (TBU) and Time Base Lower (TBL). The Time Base Registers can be written to only by supervisor-level instructions, but can be read by both user-level and supervisor-level software. For more information, see “PowerPC VEA Register Set—Time Base” in Chapter 2, “PowerPC Register Set” of the *PowerPC Microprocessor Family: The Programming Environments Manual*.
- **Supervisor-level registers** (OEA)—The OEA defines the registers an operating system uses for memory management, configuration, exception handling, and other operating system functions. The OEA defines the following supervisor-level registers for 32-bit implementations:
  - Configuration registers
    - Machine State Register (MSR). The MSR defines the state of the processor. The MSR can be modified by the Move-to Machine State Register (**mtmsr**), System Call (**sc**), and Return from Exception (**rfi**) instructions. It can be read by the Move-from Machine State Register (**mfmsr**) instruction. When an exception is taken, the contents of the MSR are saved to the Machine Status Save/Restore Register 1 (SRR1), which is described below. See “Machine State Register (MSR)” in Chapter 2, “PowerPC Register Set” of the *PowerPC Microprocessor Family: The Programming Environments Manual* for more information.

      **Implementation Note**: *Table 2-1* describes MSR bits the 750GX implements that are not required by the PowerPC Architecture.

      *Table 2-1. Additional MSR Bits*

      | Bit | Name | Description |
      |---|---|---|
      | 13 | POW | Power management enable. Optional in the PowerPC Architecture.<br>0 Power management is disabled.<br>1 Power management is enabled.<br>The processor can enter a power-saving mode when additional conditions are present. The mode chosen is determined by the DOZE, NAP, and SLEEP bits in the Hardware-Implementation-Dependent Register 0 (HID0), described in *Section 2.1.2.2* on page 65.<br>To set the POW bit, see *Table 10-2, HID0 Power Saving Mode Bit Settings*, on page 337. The 750GX will clear the POW bit when it leaves a power saving mode. |
      | 29 | PM | Performance-monitor marked mode. This bit is specific to the 750GX, and is defined as reserved by the PowerPC Architecture. See *Chapter 10, Power and Thermal Management,* on page 335.<br>0 Process is not a marked process.<br>1 Process is a marked process.<br>The MSR[PM] bit is used by the Performance-Monitor to help determine when it should count events. For a description of the Performance-Monitor, see *Chapter 11, Performance Monitor and System Related Features,* on page 349. |

      **Note:** Setting MSR[EE] masks not only the architecture-defined external interrupt and decrementer exceptions, but also the 750GX-specific system management, performance-monitor, and thermal-management exceptions.
    - Processor Version Register (PVR). This register is a read-only register that identifies the version (model) and revision level of the PowerPC processor. For more information, see “Processor Version Register (PVR)” in Chapter 2, “PowerPC Register Set” of the *PowerPC Microprocessor Family: The Programming Environments Manual*.

      **Note:** The Processor Version Number is x’7002’ for the 750GX. The processor revision level will start at x’0100’ and will be incremented for each revision of the chip.

- Memory-management registers
  - Block-Address Translation (BAT) Registers. The PowerPC OEA includes an array of Block Address Translation Registers that can be used to specify eight blocks of instruction space and eight blocks of data space. The BAT registers are implemented in pairs—eight pairs of instruction BATs (IBAT0U–IBAT7U and IBAT0L–IBAT7L) and eight pairs of data BATs (DBAT0U–DBAT7U and DBAT0L–DBAT7L). *Figure 2-1, PowerPC 750GX Microprocessor Programming Model—Registers* lists the SPR numbers for the BAT registers. For more information, see "BAT Registers" in Chapter 2, "PowerPC Register Set" of the *PowerPC Microprocessor Family: The Programming Environments Manual*. Because BAT upper and lower words are loaded separately, software must ensure that BAT translations are correct during the time that both BAT entries are being loaded.

    The 750GX implements the G bit in the IBAT registers. However, attempting to execute code from an IBAT area with G = 1 causes an instruction storage interrupt (ISI) exception. This complies with the revision of the architecture described in the *PowerPC Microprocessor Family: The Programming Environments Manual*.
  - SDR1. The SDR1 register specifies the page table base address used in virtual-to-physical address translation. See "SDR1" in Chapter 2, "PowerPC Register Set" of the *PowerPC Microprocessor Family: The Programming Environments Manual*."
  - Segment Registers (SR). The PowerPC OEA defines sixteen 32-bit Segment Registers (SR0–SR15). Note that the SRs are implemented on 32-bit implementations only. The fields in the Segment Register are interpreted differently depending on the value of bit 0. See "Segment Registers" in Chapter 2, "PowerPC Register Set" of the *PowerPC Microprocessor Family: The Programming Environments Manual* for more information.

    **Note:** The 750GX implements separate memory management units (MMUs) for instruction and data. It associates the architecture-defined SRs with the data MMU (DMMU). It reflects the values of the SRs in separate, so-called 'shadow' Segment Registers in the instruction MMU (IMMU).
- Exception-handling registers
  - Data Address Register (DAR). After a data-storage interrupt (DSI) exception or an alignment exception, DAR is set to the effective address (EA) generated by the instruction at fault. See "Data Address Register (DAR)" in Chapter 2, "PowerPC Register Set" of the *PowerPC Microprocessor Family: The Programming Environments Manual* for more information.
  - SPRG0–SPRG3. The SPRG0–SPRG3 registers are provided for operating system use. See "SPRG0–SPRG3" in Chapter 2, "PowerPC Register Set" of the *PowerPC Microprocessor Family: The Programming Environments Manual* for more information.
  - DSISR. The Data Storage Interrupt Status Register (DSISR) defines the cause of DSI and alignment exceptions. See "DSISR" in Chapter 2, "PowerPC Register Set" of the *PowerPC Microprocessor Family: The Programming Environments Manual* for more information.
  - Machine Status Save/Restore Register 0 (SRR0). The SRR0 register is used to save the address of the instruction at which execution continues when an **rfi** executes at the end of an exception handler routine. See "Machine Status Save/Restore Register 0 (SRR0)" in Chapter 2, "PowerPC Register Set" of the *PowerPC Microprocessor Family: The Programming Environments Manual* for more information.
  - Machine Status Save/Restore Register 1 (SRR1). The SRR1 is used to save machine status on exceptions and to restore machine status when **rfi** executes. See "Machine Status Save/Restore

Register 1 (SRR1)” in Chapter 2, “PowerPC Register Set” of the *PowerPC Microprocessor Family: The Programming Environments Manual* for more information.

**Note:** When a machine-check exception occurs, the 750GX sets one or more error bits in SRR1. *Table 2-2* describes SRR1 bits 750GX implements that are not required by the PowerPC Architecture.

*Table 2-2. Additional SRR1 Bits*

| Bit | Name | Description |
|---|---|---|
| 4 | CP | Internal cache parity error. |
| 11 | L2DBERR | Set by a double-bit error checking and correction (ECC) error in the L2. |
| 12 | MCpin | Set by the assertion of the machine-check interrupt ($\overline{\text{MCP}}$). |
| 13 | TEA | Set by a transfer error acknowledge ($\overline{\text{TEA}}$) assertion on the 60x bus. |
| 14 | DP | Set by a data-parity error on the 60x bus. |
| 15 | AP | Set by an address-parity error on the 60x bus. |

– Miscellaneous registers

- Time Base (TB). The TB is a 64-bit structure provided for maintaining the time of day and operating interval timers. The TB consists of two 32-bit registers—Time Base Upper (TBU) and Time Base Lower (TBL). The Time Base Registers can be written to only by supervisor-level software, but can be read by both user- and supervisor-level software. See “Time Base Facility (TB)—OEA” in Chapter 2, “PowerPC Register Set” of the *PowerPC Microprocessor Family: The Programming Environments Manual* for more information.
- Decrementer Register (DEC). This register is a 32-bit decrementing counter that provides a mechanism for causing a decrementer exception after a programmable delay; the frequency is a subdivision of the processor clock. See “Decrementer Register (DEC)” in Chapter 2, “PowerPC Register Set” of the *PowerPC Microprocessor Family: The Programming Environments Manual* for more information.

  **Note:** In the 750GX, the Decrementer Register is decremented and the time base is incremented at a speed that is one-fourth the speed of the bus clock.
- Data Address Breakpoint Register (DABR)—This optional register is used to cause a breakpoint exception if a specified data address is encountered. See “Data Address Breakpoint Register (DABR)” in Chapter 2, “PowerPC Register Set” of the *PowerPC Microprocessor Family: The Programming Environments Manual*.
- External Access Register (EAR). This optional register is used in conjunction with the External Control In Word Indexed (**eciwx**) and External Control Out Word Indexed (**ecowx**) instructions. Note that the EAR and the **eciwx** and **ecowx** instructions are optional in the PowerPC Architecture and might not be supported in all PowerPC processors that implement the OEA. See “External Access Register (EAR)” in Chapter 2, “PowerPC Register Set” of the *PowerPC Microprocessor Family: The Programming Environments Manual* for more information.

- 750GX**-specific registers**—The PowerPC Architecture allows implementation-specific SPRs. Those described below are incorporated in the 750GX. Note that, in the 750GX, these registers are all supervisor-level registers.

  – Instruction Address Breakpoint Register (IABR)—This register can be used to cause a breakpoint exception if a specified instruction address is encountered.

- Hardware-Implementation-Dependent Register 0 (HID0)—This register controls various functions, such as enabling checkstop conditions, and locking, enabling, and invalidating the instruction and data caches, power modes, miss-under-miss, and others.
- Hardware-Implementation-Dependent Register 1 (HID1)—This register reflects the state of PLL_CFG[0:4] clock signals, and phase-locked loop (PLL) selection and range bits.
- Hardware-Implementation-Dependent Register 2 (HID2)—This register controls parity enablement.
- L2 Cache Control Register (L2CR)—This register is used to configure and operate the L2 cache.
- Performance-monitor registers. The following registers are used to define and count events for use by the performance monitor:
  - The Performance-Monitor Counter Registers (PMC1–PMC4) are used to record the number of times a certain event has occurred. UPMC1–UPMC4 provide user-level read access to these registers.
  - The Monitor Mode Control Registers (MMCR0–MMCR1) are used to enable various performance-monitor interrupt functions. UMMCR0–UMMCR1 provide user-level read access to these registers.
  - The Sampled Instruction Address Register (SIA) contains the effective address of an instruction executing at or around the time that the processor signals the performance-monitor interrupt condition. USIA provides user-level read access to the SIA.
  - The 750GX does not implement the Sampled Data Address Register (SDA) or the user-level, read-only USDA registers. However, for compatibility with processors that do, those registers can be written to by boot code without causing an exception. SDA is SPR 959; USDA is SPR 943.
- Instruction Cache Throttling Control Register (ICTC)—This register has bits for enabling the instruction-cache throttling feature and for controlling the interval at which instructions are forwarded to the instruction buffer in the fetch unit. This provides control over the processor’s overall junction temperature.
- Thermal-Management Registers (THRM1, THRM2, THRM3, and THRM4)—Used to enable and set thresholds for the thermal-management facility.
  - THRM1 and THRM2 provide the ability to compare the junction temperature against two user-provided thresholds. The dual thresholds allow the thermal-management software differing degrees of action in lowering the junction temperature. The TAU can be also operated in a single-threshold mode in which the thermal sensor output is compared to only one threshold in either THRM1 or THRM2.
  - THRM3 is used to enable the thermal-management assist unit (TAU) and to control the comparator output sample time.
  - THRM4 is a read-only register containing a temperature offset (determined at the factory) applied to junction temperature measurements for improved accuracy.

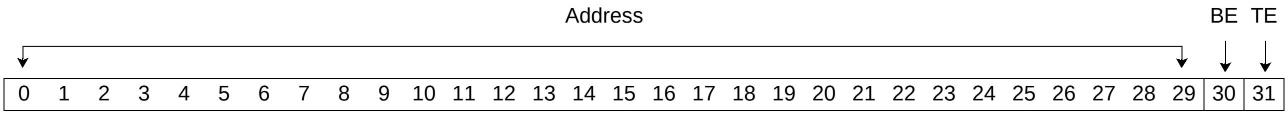
**Note:** While it is not guaranteed that the implementation of 750GX-specific registers is consistent among PowerPC processors, other processors may implement similar or identical registers.

## 2.1.2 PowerPC 750GX-Specific Registers

This section describes registers that are defined for the 750GX but are not included in the PowerPC Architecture.

### *2.1.2.1 Instruction Address Breakpoint Register (IABR)*

The Instruction Address Breakpoint Register (IABR) supports the instruction address breakpoint exception. When this exception is enabled, instruction fetch addresses are compared with an effective address stored in the IABR. If the word specified in the IABR is fetched, the instruction breakpoint handler is invoked. The instruction that triggers the breakpoint does not execute before the handler is invoked. For more information, see *Section 4.5.14, Instruction Address Breakpoint Exception (0x01300),* on page 173. The IABR can be accessed with **mtspr** and **mfspr** using the SPR 1010.



| Bits | Field Name | Description |
|---|---|---|
| 0:29 | Address | Word address to be compared. |
| 30 | BE | Breakpoint enabled. Setting this bit indicates that breakpoint checking is to be done. |
| 31 | TE | Translation enabled. An IABR match is signaled if this bit matches MSR[IR]. |

### *2.1.2.2 Hardware-Implementation-Dependent Register 0 (HID0)*

The Hardware-Implementation-Dependent Register 0 (HID0) controls the state of several functions within 750GX. HID0 can be accessed with **mtspr** and **mfspr** using SPR 1008.

| EMCP | DBP | EBA | EBD | Reserved | | | PAR | DOZE | NAP | SLEEP | DPM | RISEG | Reserved | MUM | NHR | ICE | DCE | ILOCK | DLOCK | ICFI | DCFI | SPD | IFEM | SGE | DCFA | BTIC | Reserved | ABE | BHT | Reserved | NOOPTI |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |

| Bits | Field Name | Description |
|---|---|---|
| 0 | EMCP | Enable $\overline{\text{MCP}}$. The primary purpose of this bit is to mask out further machine-check exceptions caused by assertion of $\overline{\text{MCP}}$, similar to how MSR[EE] can mask external interrupts.<br>0 Masks $\overline{\text{MCP}}$. Asserting $\overline{\text{MCP}}$ does not generate a machine-check exception or a checkstop.<br>1 Asserting $\overline{\text{MCP}}$ causes a checkstop if MSR[ME] = 0 or a machine-check exception if ME = 1. |
| 1 | DBP | Disable 60x bus address-parity and data-parity generation.<br>0 Parity generation is enabled.<br>1 Disable parity generation. If the system does not use address or data parity and the respective parity checking is disabled (HID0[EBA] or HID0[EBD] = 0), input receivers for those signals are disabled, require no pull-up resistors, and thus should be left unconnected. If all parity generation is disabled, all parity checking should also be disabled and parity signals need not be connected. |
| 2 | EBA[1] | Enable/disable 60x bus address-parity checking<br>0 Prevents address-parity checking.<br>1 Allows an address-parity error to cause a checkstop if MSR[ME] = 0 or a machine-check exception if MSR[ME] = 1.<br>EBA and EBD allow the processor to operate with memory subsystems that do not generate parity. |
| 3 | EBD[1] | Enable 60x bus data-parity checking<br>0 Parity checking is disabled.<br>1 Allows a data-parity error to cause a checkstop if MSR[ME] = 0 or a machine-check exception if MSR[ME] = 1.<br>EBA and EBD allow the processor to operate with memory subsystems that do not generate parity. |
| 4 | — | Reserved. Must set to 0. |
| 5 | — | Not used. Defined as EICE on some earlier processors. |
| 6 | — | Reserved. Must set to 0. |
| 7 | PAR | Disable precharge of $\overline{\text{ARTRY}}$.<br>0 Precharge of $\overline{\text{ARTRY}}$ enabled.<br>1 Alters bus protocol slightly by preventing the processor from driving $\overline{\text{ARTRY}}$ to high (negated) state. If this is done, the system must restore the signals to the high state. |
| 8 | DOZE[2] | Doze mode enable. Operates in conjunction with MSR[POW].<br>0 Doze mode disabled.<br>1 Doze mode enabled. Doze mode is invoked by setting MSR[POW] while this bit is set. In doze mode, the phase-locked loop (PLL), time base, and snooping remain active. |

1. For additional information, see *Section 11.9, Checkstops,* on page 361.
2. For additional information about power-saving modes, see *Table 10-2, HID0 Power Saving Mode Bit Settings*, on page 337.

| Bits | Field Name | Description |
|---|---|---|
| 9 | NAP[2] | Nap mode enable. Operates in conjunction with MSR[POW].<br>0 Nap mode disabled.<br>1 Nap mode enabled. Doze mode is invoked by setting MSR[POW] while this bit is set. In nap mode, the PLL and the time base remain active. |
| 10 | SLEEP[2] | Sleep mode enable. Operates in conjunction with MSR[POW].<br>0 Sleep mode disabled.<br>1 Sleep mode enabled. Sleep mode is invoked by setting MSR[POW] while this bit is set. QREQ is asserted to indicate that the processor is ready to enter sleep mode. If the system logic determines that the processor can enter sleep mode, the quiesce acknowledge signal, QACK, is asserted back to the processor. Once QACK assertion is detected, the processor enters sleep mode after several processor clocks. At this point, the system logic can turn off the PLL by first configuring PLL_CFG[0:4] to PLL bypass mode, then disabling SYSCLK. |
| 11 | DPM | Dynamic power management enable.<br>0 Dynamic power management is disabled.<br>1 Functional units enter a low-power mode automatically if the unit is idle. This does not affect operational performance and is transparent to software or any external hardware. |
| 12 | RISEG | Read Instruction Segment Register (for test only). |
| 13 | — | Reserved. |
| 14 | MUM | Miss-under-Miss enable.<br>0 Function disabled.<br>1 Function enabled. |
| 15 | NHR | Not a hard reset (software-use only). Helps software distinguish a hard reset from a soft reset.<br>0 A hard reset has occurred if software previously set this bit.<br>1 A hard reset has not occurred. If software sets this bit after a hard reset, when a reset occurs and this bit remains set, software can tell it was a soft reset. |
| 16 | ICE | Instruction-cache enable<br>0 The instruction cache is neither accessed nor updated. All pages are accessed as if they were marked cache-inhibited (WIM = X1X). Potential cache accesses from the bus (snoop and cache operations) are ignored. In the disabled state for the L1 caches, the cache tag state bits are ignored and all accesses are propagated to the L2 cache or bus as single-beat transactions. For those transactions, however, Cache Inhibit (CI) reflects the original state determined by address translation regardless of cache disabled status. ICE is zero at power-up.<br>1 The instruction cache is enabled |
| 17 | DCE | Data-cache enable<br>0 The data cache is neither accessed nor updated. All pages are accessed as if they were marked cache-inhibited (WIM = X1X). Potential cache accesses from the bus (snoop and cache operations) are ignored. In the disabled state for the L1 caches, the cache tag state bits are ignored and all accesses are propagated to the L2 cache or bus as single-beat transactions. For those transactions, however, CI reflects the original state determined by address translation regardless of cache disabled status. DCE is zero at power-up.<br>1 The data cache is enabled. |

1. For additional information, see *Section 11.9, Checkstops,* on page 361.
2. For additional information about power-saving modes, see *Table 10-2, HID0 Power Saving Mode Bit Settings*, on page 337.

| Bits | Field Name | Description |
|---|---|---|
| 18 | ILOCK | Instruction-cache lock<br>0 Normal operation.<br>1 Instruction cache is locked. A locked cache supplies data normally on a hit, but is treated as a cache-inhibited transaction on a miss. On a miss, the transaction to the bus or the L2 cache is single-beat. However, $\overline{CI}$ still reflects the original state as determined by address translation independent of cache locked or disabled status.<br>To prevent locking during a cache access, an Instruction Synchronization (**isync**) instruction must precede the setting of ILOCK. |
| 19 | DLOCK | Data-cache lock.<br>0 Normal operation.<br>1 Data cache is locked. A locked cache supplies data normally on a hit, but is treated as a cache-inhibited transaction on a miss. On a miss, the transaction to the bus or the L2 cache is single-beat. However, $\overline{CI}$ still reflects the original state as determined by address translation independent of cache locked or disabled status. A snoop hit to a locked L1 data cache performs as if the cache were not locked. A cache block invalidated by a snoop remains invalid until the cache is unlocked.<br>To prevent locking during a cache access, a **sync** instruction must precede the setting of DLOCK. |
| 20 | ICFI | Instruction-cache flash invalidate<br>0 The instruction cache is not invalidated. The bit is cleared when the invalidation operation begins (usually the next cycle after the write operation to the register). The instruction cache must be enabled for the invalidation to occur.<br>1 An invalidate operation is issued that marks the state of each instruction-cache block as invalid without writing back modified cache blocks to memory. Cache access is blocked during this time. Bus accesses to the cache are signaled as misses during invalidate-all operations. Setting ICFI clears all the valid bits of the blocks and the pseudo least-recently used (PLRU) bits to point to way L0 of each set. Once the L1 flash invalidate bits are set through an **mtspr** operation, hardware automatically resets these bits in the next cycle (provided the corresponding cache enable bits are set in HID0).<br>**Note:** In the PowerPC 603 and PowerPC 603e processors, the proper use of the ICFI and DCFI bits was to set them and clear them in two consecutive **mtspr** operations. Software that already has this sequence of operations does not need to be changed to run on the 750GX. |
| 21 | DCFI | Data-cache flash invalidate<br>0 The data cache is not invalidated. The bit is cleared when the invalidation operation begins (usually the next cycle after the write operation to the register). The data cache must be enabled for the invalidation to occur.<br>1 An invalidate operation is issued that marks the state of each data-cache block as invalid without writing back modified cache blocks to memory. Cache access is blocked during this time. Bus accesses to the cache are signaled as a miss during invalidate-all operations. Setting DCFI clears all the valid bits of the blocks and the PLRU bits to point to way L0 of each set. Once the L1 flash invalidate bits are set through an **mtspr** operation, hardware automatically resets these bits in the next cycle (provided that the corresponding cache enable bits are set in HID0).<br>Setting this bit clears all the valid bits of the blocks and the PLRU bits to point to way L0 of each set.<br>**Note:** In the PowerPC 603 and PowerPC 603e processors, the proper use of the ICFI and DCFI bits was to set them and clear them in two consecutive **mtspr** operations. Software that already has this sequence of operations does not need to be changed to run on the 750GX. |

1. For additional information, see *Section 11.9, Checkstops,* on page 361.
2. For additional information about power-saving modes, see *Table 10-2, HID0 Power Saving Mode Bit Settings*, on page 337.

| Bits | Field Name | Description |
|---|---|---|
| 22 | SPD | Speculative cache access disable<br>0 Speculative bus accesses to nonguarded space (G = 0) from both the instruction and data caches are enabled.<br>1 Speculative bus accesses to nonguarded space in both caches are disabled. |
| 23 | IFEM | Enable M bit on bus for instruction fetches.<br>0 M bit disabled. Instruction fetches are treated as nonglobal on the bus.<br>1 Instruction fetches reflect the M bit from the WIM settings. |
| 24 | SGE | Store gathering enable<br>0 Store gathering is disabled.<br>1 Integer store gathering is performed for write-through to nonguarded space or for cache-inhibited stores to nonguarded space for 4-byte, word-aligned stores. The load store unit (LSU) combines stores to form a double word that is sent out on the 60x bus as a single-beat operation. Stores are gathered only if successive, eligible stores are queued and pending. Store gathering is performed regardless of address order or endian mode. The store-gathering feature is enabled by setting the HID0[SGE] bit (bit 24). |
| 25 | DCFA | Data-cache flush assist. (Force data cache to ignore invalid sets on miss replacement selection.)<br>0 The data-cache flush assist facility is disabled.<br>1 The miss replacement algorithm ignores invalid entries and follows the replacement sequence defined by the PLRU bits. This reduces the series of uniquely addressed **load** or Data Cache Block Zero (**dcbz**) instructions to eight per set. The bit should be set just before beginning a cache flush routine, and should be cleared when the series of instructions completes. |
| 26 | BTIC | Branch target instruction-cache enable—used to enable use of the 64-entry branch instruction cache.<br>0 The BTIC is disabled, the contents are invalidated, and the BTIC behaves as if it were empty. New entries cannot be added until the BTIC is enabled.<br>1 The BTIC is enabled, and new entries can be added. |
| 27 | — | Not used. Defined as FBIOB on earlier 603-type processors. |
| 28 | ABE | Address broadcast enable—controls whether certain address-only operations (such as cache operations, Enforce In-Order Execution of I/O [**eieio**], and Synchronization [**sync**]) are broadcast on the 60x bus.<br>0 Address-only operations affect only local L1 and L2 caches and are not broadcast.<br>1 Address-only operations are broadcast on the 60x bus. Affected instructions are **eieio**, **sync**, Data Cache Block Invalidate (**dcbi**), Data Cache Block Flush (**dcbf**), and Data Cache Block Store (**dcbst**). A **sync** instruction completes only after a successful broadcast. Execution of **eieio** causes a broadcast that can be used to prevent any external devices, such as a bus bridge chip, from store gathering.<br>**Note:** A Data Cache Block Set to Zero (**dcbz**) instruction (with M = 1, coherency required) always broadcasts on the 60x bus regardless of the setting of this bit. An Instruction Cache Block Invalidate (**icbi**) is never broadcast. No cache operations, except **dcbz**, are snooped by the 750GX regardless of whether the ABE is set. Bus activity caused by these instructions results directly from performing the operation on the 750GX cache. |

1. For additional information, see *Section 11.9, Checkstops,* on page 361.
2. For additional information about power-saving modes, see *Table 10-2, HID0 Power Saving Mode Bit Settings*, on page 337.

| Bits | Field Name | Description |
|---|---|---|
| 29 | BHT | Branch history table enable<br>0 BHT disabled. The 750GX uses static branch prediction as defined by the PowerPC User Instruction Set Architecture (UISA) for those branch instructions the BHT would have otherwise used to predict (that is, those that use the CR as the only mechanism to determine direction). For more information on static branch prediction, see “Conditional Branch Control,” in Chapter 4 of the *PowerPC Microprocessor Family: The Programming Environments Manual*.<br>1 Allows the use of the 512-entry branch history table (BHT).<br>The BHT is disabled at power-on reset. All entries are set to weakly, not-taken. |
| 30 | Reserved | Reserved. |
| 31 | NOOPTI | No-op the data-cache touch instructions.<br>0 The Data Cache Block Touch (**dcbt**) and Data Cache Block Touch for Store (**dcbtst**) instructions are enabled.<br>1 The **dcbt** and **dcbtst** instructions are no-oped globally. |

1. For additional information, see *Section 11.9, Checkstops,* on page 361.
2. For additional information about power-saving modes, see *Table 10-2, HID0 Power Saving Mode Bit Settings*, on page 337.

### *2.1.2.3 Hardware-Implementation-Dependent Register 1 (HID1)*

The Hardware-Implementation-Dependent Register 1 (HID1) reflects the state of the PLL_CFG[0:4] signals. HID1 can be accessed with **mtspr** and **mfspr** using SPR 1009.

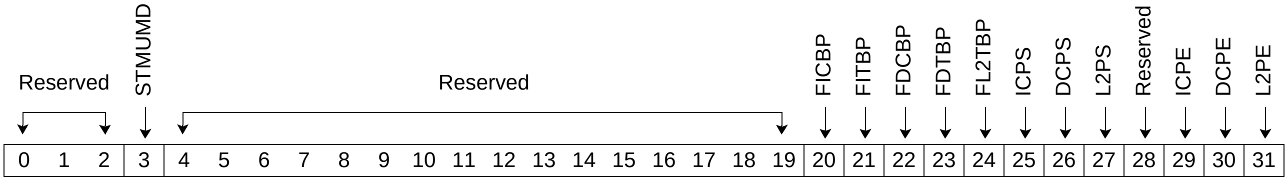
| PCE | | | | | PRE | | PSTAT1 | ECLK | Reserved | | | Reserved | | PI0 | PS | PC0 | | | | | PR0 | | Reserved | PC1 | | | | | PR1 | | Reserved |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |

| Bits | Field Name | Description |
|---|---|---|
| 0:4 | PCE | PLL external configuration bits (read-only). |
| 5:6 | PRE | PLL external range bits (read-only). |
| 7 | PSTAT1 | PLL status. Specifies the PLL clocking the processor:<br>0 PLL0 is the processor clock source<br>1 PLL1 is the processor clock source. |
| 8 | ECLK | Set to 1 to enable the CLKOUT pin. |
| 9:11 | Reserved | Select the internal clock to be output on the CLKOUT pin with the following decode:<br>000 Factory use only<br>001 PLL0 core clock (freq/2)<br>010 Factory use only<br>011 PLL1 core clock (freq/2)<br>100 Factory use only<br>101 Core clock (freq/2)<br>Other Reserved<br>**Note:** These clock configuration bits reflect the state of the PLL_CFG[0:4] pins. Clock options should only be used for design debug and characterization. |
| 12:13 | Reserved | Reserved. |
| 14 | PI0 | PLL 0 internal configuration select.<br>0 Select external configuration and range bits to control PLL 0.<br>1 Select internal fields in HID1 to control PLL0. |
| 15 | PS | PLL select.<br>0 Select PLL 0 as the source for the processor clock.<br>1 Select PLL 1 as the source for the processor clock. |
| 16:20 | PC0 | PLL 0 configuration bits. |
| 21:22 | PR0 | PLL 0 range select bits. |
| 23 | Reserved | Reserved. |
| 24:28 | PC1 | PLL 1 configuration bits. |
| 29:30 | PR1 | PLL 1 range bits. |
| 31 | Reserved | Reserved. |

### *2.1.2.4 Hardware-Implementation-Dependent Register 2 (HID2)*

The Hardware-Implementation-Dependent Register 2 (HID2) enables parity. The status bits (25:27) are set when a parity error is detected and cleared by writing '0' to each bit. See the *IBM PowerPC 750GX RISC Microprocessor Datasheet* for details.

HID2 can be accessed with **mtspr** and **mfspr** using SPR 1016.



| Bits | Field Name | Description | Notes |
|---|---|---|---|
| 0:2 | Reserved | Reserved | 1 |
| 3 | STMUMD | Disable store miss-under-miss processing (changes the allowed outstanding store misses from two to one. | |
| 4:19 | Reserved | Reserved | 1 |
| 20 | FICBP | Force instruction-cache bad parity. | |
| 21 | FITBP | Force instruction-tag bad parity. | |
| 22 | FDCBP | Force data-cache bad parity. | |
| 23 | FDTBP | Force data-tag bad parity. | |
| 24 | FL2TBP | Force L2-tag bad parity. | |
| 25 | ICPS | L1 instruction-cache/instruction-tag parity error status/mask. | |
| 26 | DCPS | L1 data-cache/data-tag parity error status/mask. | |
| 27 | L2PS | L2 tag parity error status/mask. | |
| 28 | Reserved | Reserved. | 1 |
| 29 | ICPE | Enable L1 instruction-cache/instruction-tag parity checking. | |
| 30 | DCPE | Enable L1 data-cache/data-tag parity checking. | |
| 31 | L2PE | Enable L2 tag parity checking. | |

1. Reserved. Used as factory test bits. Do not change from their power-up state unless indicated to do so.

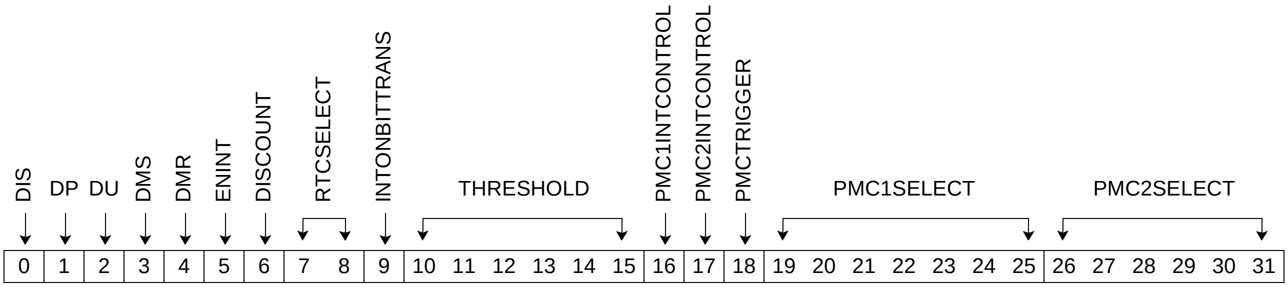
### *2.1.2.5 Performance-Monitor Registers*

This section describes the registers used by the performance monitor, which is described in *Chapter 11, Performance Monitor and System Related Features,* on page 349.

*Monitor Mode Control Register 0 (MMCR0)*

The Monitor Mode Control Register 0 (MMCR0) is a 32-bit SPR provided to specify events to be counted and recorded. The MMCR0 can be accessed only in supervisor mode. User-level software can read the contents of MMCR0 by issuing an **mfspr** instruction to UMMCR0, described in the following section.

This register must be cleared at power up. Reading this register does not change its contents. MMCR0 can be accessed with **mtspr** and **mfspr** using SPR 952.



| Bits | Field Name | Description |
|---|---|---|
| 0 | DIS | Disables counting unconditionally.<br>0 The values of the PMC*n* counters can be changed by hardware.<br>1 The values of the PMC*n* counters cannot be changed by hardware. |
| 1 | DP | Disables counting while in supervisor mode.<br>0 The PMC*n* counters can be changed by hardware.<br>1 If the processor is in supervisor mode (MSR[PR] is cleared), the counters are not changed by hardware. |
| 2 | DU | Disables counting while in user mode.<br>0 The PMC*n* counters can be changed by hardware.<br>1 If the processor is in user mode (MSR[PR] is set), the PMC*n* counters are not changed by hardware. |
| 3 | DMS | Disables counting while MSR[PM] is set.<br>0 The PMC*n* counters can be changed by hardware.<br>1 If MSR[PM] is set, the PMC*n* counters are not changed by hardware. |
| 4 | DMR | Disables counting while MSR[PM] is zero.<br>0 The PMC*n* counters can be changed by hardware.<br>1 If MSR[PM] is cleared, the PMC*n* counters are not changed by hardware. |
| 5 | ENINT | Enables performance-monitor interrupt signaling.<br>0 Interrupt signaling is disabled.<br>1 Interrupt signaling is enabled.<br>Cleared by hardware when a performance-monitor interrupt is signaled. To re-enable these interrupt signals, software must set this bit after handling the performance-monitor interrupt. |

| Bits | Field Name | Description |
|---|---|---|
| 6 | DISCOUNT | Disables counting of PMC*n* when a performance-monitor interrupt is signaled (that is, ((PMC*n*INTCONTROL = '1') & (PMC*n*[0] = '1') & (ENINT = '1')) or when an enabled time-base transition occurs with ((INTONBITTRANS = '1') & (ENINT = '1')).<br>0 Signaling a performance-monitor interrupt does not affect the counting status of PMC*n*.<br>1 Signaling a performance-monitor interrupt prevents changing of the PMC1 counter. The PMC*n* counter does not change if PMC2COUNTCTL = '0'.<br>Because a time-base signal could have occurred along with an enabled counter overflow condition, software should always reset INTONBITTRANS to zero, if the value in INTONBITTRANS was a one. |
| 7:8 | RTCSELECT | 64-bit time base, bit selection enable.<br>00 Pick bit 63 to count.<br>01 Pick bit 55 to count.<br>10 Pick bit 51 to count.<br>11 Pick bit 47 to count. |
| 9 | INTONBITTRANS | Cause interrupt signaling when the bit identified in RTCSELECT transitions from off to on.<br>0 Do not allow interrupt signal if chosen bit transitions.<br>1 Signal interrupt if chosen bit transitions.<br>Software is responsible for setting and clearing INTONBITTRANS. |
| 10:15 | THRESHOLD | Threshold value. The 750GX supports all six bits, allowing threshold values from 0–63. The intent of the THRESHOLD support is to characterize L1 data-cache misses. |
| 16 | PMC1INTCONTROL | Enables interrupt signaling due to PMC1 counter overflow.<br>0 Disable PMC1 interrupt signaling due to PMC1 counter overflow.<br>1 Enable PMC1 interrupt signaling due to PMC1 counter overflow. |
| 17 | PMCINTCONTROL | Enable interrupt signaling due to any PMC2–PMC4 counter overflow. Overrides the setting of DISCOUNT.<br>0 Disable PMC2–PMC4 interrupt signaling due to PMC2–PMC4 counter overflow.<br>1 Enable PMC2–PMC4 interrupt signaling due to PMC2–PMC4 counter overflow. |
| 18 | PMCTRIGGER | Can be used to trigger counting of PMC2–PMC4 after PMC1 has overflowed or after a performance-monitor interrupt is signaled.<br>0 Enable PMC2–PMC4 counting.<br>1 Disable PMC2–PMC4 counting until either PMC1[0] = 1 or a performance-monitor interrupt is signaled. |
| 19:25 | PMC1SELECT | PMC1 input selector; 128 events selectable. |
| 26:31 | PMC2SELECT | PMC2 input selector; 64 events selectable. |

### *User Monitor Mode Control Register 0 (UMMCR0)*

The contents of MMCR0 are reflected to UMMCR0, which can be read by user-level software. MMCR0 can be accessed with **mfspr** using SPR 936.

### *Monitor Mode Control Register 1 (MMCR1)*

The Monitor Mode Control Register 1 (MMCR1) functions as an event selector for Performance-Monitor Counter Registers 3 and 4 (PMC3 and PMC4). Corresponding events to the MMCR1 bits are described in *Performance-Monitor Counter Registers (PMCn)*.

MMCR1 can be accessed with **mtspr** and **mfspr** using SPR 956. User-level software can read the contents of MMCR1 by issuing an **mfspr** instruction to UMMCR1, described in the following section.

| PMC3SELECT | PMC4SELECT | Reserved |
|---|---|---|
| 0 1 2 3 4 | 5 6 7 8 9 | 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 |

| Bits | Field Name | Description |
|---|---|---|
| 0:4 | PMC3SELECT | PMC3 input selector. Thirty-two events selectable. See *Performance-Monitor Counter Registers (PMCn)* on page 74 for defined selections. |
| 5:9 | PMC4SELECT | PMC4 input selector. Thirty-two events selectable. See *Performance-Monitor Counter Registers (PMCn)* on page 74 for defined selections. |
| 10:31 | Reserved | Reserved. |

### *User Monitor Mode Control Register 1 (UMMCR1)*

The contents of MMCR1 are reflected to UMMCR1, which can be read by user-level software. MMCR1 can be accessed with **mfspr** using SPR 940.

### *Performance-Monitor Counter Registers (PMCn)*

PMC1–PMC4 are 32-bit counters that can be programmed to generate interrupt signals when they overflow. Counters are considered to overflow when the high-order bit (the sign bit) becomes set; that is, they reach the value 2147483648 (0x8000_0000). However, an interrupt is not signaled unless both PMC*n*[INTCONTROL] and MMCR0[ENINT] are also set.

**Note:** The interrupts can be masked by clearing MSR[EE]; the interrupt signal condition can occur with MSR[EE] cleared, but the exception is not taken until EE is set. Setting MMCR0[DISCOUNT] forces counters to stop counting when a counter interrupt occurs.

Software is expected to use **mtspr** to set PMC explicitly to nonoverflow values. If software sets an overflow value, an erroneous exception might occur. For example, if both PMC*n*[INTCONTROL] and MMCR0[ENINT] are set and **mtspr** loads an overflow value, an interrupt signal will be generated without any event counting having taken place.

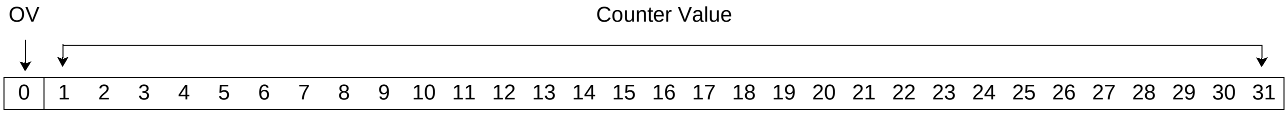
The event to be monitored by PMC1 can be chosen by setting MMCR0[19:25]. The event to be monitored by PMC2 can be chosen by setting MMCR0[26:31]. The event to be monitored by PMC3 can be chosen by setting MMCR1[0:4]. The event to be monitored by PMC4 can be chosen by setting MMCR1[5:9]. The selected events are counted beginning when MMCR0 is set until either MMCR0 is reset or a performance-monitor interrupt is generated.

The following tables list the selectable events and their encodings:

- *Table 11-2, PMC1 Events—MMCR0[19:25] Select Encodings*, on page 352.
- *Table 11-3, PMC2 Events—MMCR0[26:31] Select Encodings*, on page 352.
- *Table 11-4, PMC3 Events—MMCR1[0:4] Select Encodings*, on page 353.
- *Table 11-5, PMC4 Events—MMCR1[5:9] Select Encodings*, on page 354.

The PMC registers can be accessed with **mtspr** and **mfspr** using following SPR numbers:

- PMC1 is SPR 953
- PMC2 is SPR 954
- PMC3 is SPR 957
- PMC4 is SPR 958



| Bits | Field Name | Description |
|---|---|---|
| 0 | OV | Overflow. When this bit is set it indicates that this counter has reached its maximum value. |
| 1:31 | Counter Value | Indicates the number of occurrences of the specified event. |

### *User Performance-Monitor Counter Registers (UPMCn)*

The contents of the PMC1–PMC4 are reflected to UPMC1–UPMC4, which can be read by user-level software. The UPMC registers can be read with **mfspr** using the following SPR numbers:

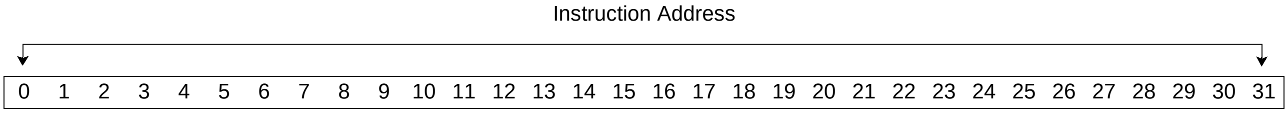
- UPMC1 is SPR 937
- UPMC2 is SPR 938
- UPMC3 is SPR 941
- UPMC4 is SPR 942

### *Sampled Instruction Address Register (SIA)*

The Sampled Instruction Address Register (SIA) is a supervisor-level register that contains the effective address of an instruction executing at or around the time that the processor signals the performance-monitor interrupt condition.

If the performance-monitor interrupt is triggered by a threshold event, the SIA contains the exact instruction (called the sampled instruction) that caused the counter to overflow.

If the performance-monitor interrupt was caused by something besides a threshold event, the SIA contains the address of the last instruction completed during that cycle. SIA can be accessed with the **mtspr** and **mfspr** instructions using SPR 955.



#### *User Sampled Instruction Address Register (USIA)*

The contents of SIA are reflected to USIA, which can be read by user-level software. USIA can be accessed with the **mfspr** instructions using SPR 939.

#### *Sampled Data Address Register (SDA) and User Sampled Data Address Register (USDA)*

The 750GX does not implement the Sampled Data Address Register (SDA) or the user-level, read-only USDA registers. However, for compatibility with processors that do, those registers can be written to by boot code without causing an exception. SDA is SPR 959; USDA is SPR 943.

### 2.1.3 Instruction Cache Throttling Control Register (ICTC)

Reducing the rate of instruction fetching can control junction temperature without the complexity and overhead of dynamic clock control. System software can control instruction forwarding by writing a nonzero value to the supervisor-level ICTC register. The overall junction temperature reduction comes from the dynamic power management of each functional unit when the 750GX is idle in between instruction fetches. PLL (phase-locked loop) and DLL (delay-locked loop) configurations are unchanged.

Instruction-cache throttling is enabled by setting ICTC[E] and writing the instruction forwarding interval into ICTC[FI]. Enabling, disabling, and changing the instruction forwarding interval immediately affect instruction forwarding.

The ICTC register can be accessed with the **mtspr** and **mfspr** instructions using SPR 1019.

| Reserved | | | | | | | | | | | | | | | | | | | | | | | FI | | | | | | | | E |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |

| Bits | Field Name | Description |
|---|---|---|
| 0:22 | Reserved | Reserved for future use. The system software should always write zeros to these bits when writing to the THRM SPRs. |
| 23:30 | FI | Instruction forwarding interval expressed in processor clocks.<br>0x00 0 clock cycles<br>0x01 1 clock cycle<br>.<br>.<br>0xFF 255 clock cycles |
| 31 | E | Cache throttling enable<br>0 Disable instruction-cache throttling.<br>1 Enable instruction-cache throttling. |

## 2.1.4 Thermal-Management Registers (THRMn)

The on-chip thermal-management assist unit provides the following functions:

- Compares the junction temperature against user programmed thresholds
- Generates a thermal-management interrupt if the temperature crosses the threshold
- Provides a way for a successive approximation routine to estimate junction temperature

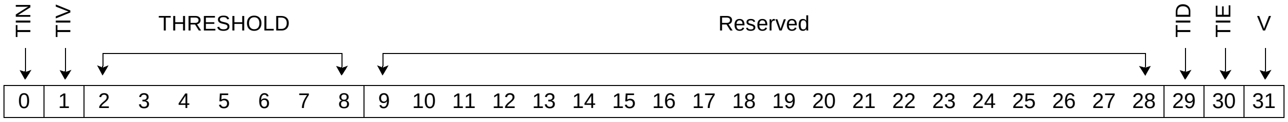
Control and access to the thermal-management assist unit is through the privileged **mtspr** and **mfspr** instructions to the four THRM registers.

### *2.1.4.1 Thermal-Management Registers 1–2 (THRM1–THRM2)*

THRM1 and THRM2 provide the ability to compare the junction temperature against two user-provided thresholds. Having dual thresholds allows thermal-management software differing degrees of action in reducing junction temperature. Thermal management can use a single-threshold mode in which the thermal sensor output is compared to only one threshold in either THRM1 or THRM2.

If an **mtspr** affects a THRM register that contains operating parameters for an ongoing comparison during operation of the thermal assist unit, the respective TIV bits are cleared and the comparison is restarted. Changing THRM3 forces the TIV bits of both THRM1 and THRM2 to 0, and restarts the comparison if THRM3[E] is set (see *Section 2.1.4.2* on page 79).

Examples of valid THRM1/THRM2 bit settings are shown in *Table 2-3* on page 79.



| Bits | Field Name | Description |
|---|---|---|
| 0 | TIN | Thermal-management interrupt bit. Read only. This bit is set if the thermal sensor output crosses the threshold specified in the SPR. The state of this bit is valid only if TIV is set. The interpretation of the TIN bit is controlled by the TID bit. See *Table 2-3*. |
| 1 | TIV | Thermal-management interrupt valid. Read only. This bit is set by the thermal assist logic to indicate that the thermal-management interrupt (TIN) state is valid. See *Table 2-3*. |
| 2:8 | THRESHOLD | Threshold that the thermal sensor output is compared to. The range is 0°–127°C in increments of 1°C. Note that this is not the resolution of the thermal sensor. |
| 9:28 | Reserved | Reserved. System software should clear these bits when writing to the THRM*n* SPRs. |
| 29 | TID | Thermal-management interrupt direction bit. Selects the result of the temperature comparison to set TIN and to assert a thermal-management interrupt if TIE is set. If TID is cleared, TIN is set and an interrupt occurs if the junction temperature exceeds the threshold. If TID is set, TIN is set and an interrupt is indicated if the junction temperature is below the threshold. See *Table 2-3*. |
| 30 | TIE | Thermal-management interrupt enable. Enables assertion of the thermal-management interrupt signal. The thermal-management interrupt is maskable by the MSR[EE] bit. If TIE is cleared and THRM*n* is valid, the TIN bit records the status of the junction temperature versus threshold comparison without causing an exception. This feature allows system software to make a successive approximation to estimate the junction temperature. See *Table 2-3* on page 79. |
| 31 | V | SPR valid bit. Setting this bit indicates that the SPR contains a valid threshold, TID, and TIE control bit. Setting THRM1/2[V] and THRM3[E] to 1 enables operation of the thermal sensor. See *Table 2-3* on page 79. |

*Table 2-3. Valid THRM1/THRM2 Bit Settings*

| TIN[1] | TIV[1] | TID | TIE | V | Description |
|---|---|---|---|---|---|
| x | x | x | x | 0 | Invalid entry. The threshold in the SPR is not used for comparison. |
| x | x | x | 0 | 1 | Disable thermal-management interrupt assertion. |
| x | x | 0 | x | 1 | Set TIN and assert thermal-management interrupt if TIE = 1 and the junction temperature exceeds the threshold. If TIE = 0, then no interrupt will be taken when the threshold is achieved. |
| x | x | 1 | x | 1 | Set TIN and assert thermal-management interrupt if TIE = 1 and the junction temperature is less than the threshold. |
| x | 0 | x | x | 1 | The state of the TIN bit is not valid. |
| 0 | 1 | 0 | x | 1 | The junction temperature is less than the threshold and as a result the thermal-management interrupt is not generated for TIE = 1. |
| 1 | 1 | 0 | x | 1 | The junction temperature is greater than the threshold and as a result the thermal-management interrupt is generated if TIE = 1. |
| 0 | 1 | 1 | x | 1 | The junction temperature is greater than the threshold and as a result the thermal-management interrupt is not generated for TIE = 1. |
| 1 | 1 | 1 | x | 1 | The junction temperature is less than the threshold and as a result the thermal-management interrupt is generated if TIE = 1 |

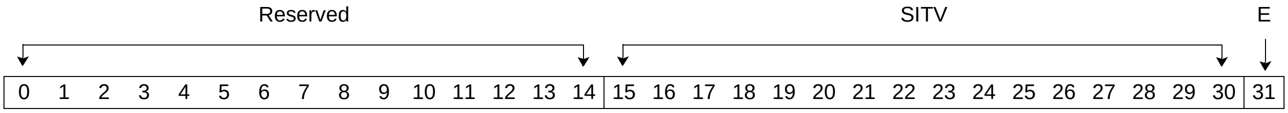
[1] TIN and TIV are read-only status bits.

### 2.1.4.2 Thermal-Management Register 3 (THRM3)

The THRM3 register is used to enable the thermal assist unit and to control the timing of the output sample comparison. The thermal assist logic manages the thermal-management interrupt generation and time-multiplexed comparisons in dual-threshold mode, as well as other control functions.

The THRM registers can be accessed with the **mtspr** and **mfspr** instructions using the following SPR numbers:

- THRM1 is SPR 1020
- THRM2 is SPR 1021
- THRM3 is SPR 1022



| Bits | Field Name | Description |
|---|---|---|
| 0:14 | Reserved | Reserved for future use. System software should clear these bits when writing to THRM3. |
| 15:30 | SITV | Sample interval timer value. Number of elapsed processor clock cycles before a junction temperature versus threshold comparison result is sampled to set the TIN bit and generate an interrupt. This is necessary due to the thermal sensor, the digital-to-analog converter (DAC), and because the analog comparator settling time is greater than the processor cycle time. The value should be configured to allow a sampling interval of 20 microseconds. |
| 31 | E | Enables the thermal sensor compare operation if either THRM1[V] or THRM2[V] is set. |

### *2.1.4.3 Thermal-Management Register 4 (THRM4)*

Due to process and thermal sensor variations, a temperature offset is provided that can be read via an **mfspr** instruction to THRM4. The TOFFSET field is an 8-bit signed integer that represents the temperature offset measured; it is burned into the THRM4 Register at the factory to allow for enhanced accuracy. When in TAU single-threshold or dual-threshold mode, TOFFSET should be subtracted from the desired temperature before setting the THRMn(THRESHOLD) field. In junction-temperature-determination mode, TOFFSET must be added to the final threshold number to determine the temperature. The temperature, in °C, equals:

THRMn[THRESHOLD] + sign-extended [TOFFSET]

The THRM4 register can be accessed with the **mfspr** instruction using SPR 920.

| Reserved | | | | | | | | | | | | | | | | | | | | | | | | TOFFSET | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |

| Bits | Field Name | Description |
|---|---|---|
| 0:23 | Reserved | Reserved for future use. Always read as zeros. |
| 24:31 | TOFFSET | Thermal calibration offset field set during factory test.<br>The °C offset value is in an 8-bit, signed, two’s complement format. |

### 2.1.5 L2 Cache Control Register (L2CR)

The L2 Cache Control Register is a supervisor-level, implementation-specific SPR used to configure and operate the L2 cache. It is cleared by a hard reset or power-on reset.

The L2 cache interface is described in *Chapter 9, L2 Cache,* on page 323. The L2CR register can be accessed with the **mtspr** and **mfspr** instructions using SPR 1017.

| L2E | CE | Reserved | DO | GI | Reserved | WT | TS | Reserved | LOCKLO | LOCKHI | SHEE | SHERR | LOCK | IO | Reserved | IP |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 3 4 5 6 7 8 | 9 | 10 | 11 | 12 | 13 | 14 15 16 17 18 19 | 20 | 21 | 22 | 23 | 24 25 26 27 | 28 | 29 30 | 31 |

| Bits | Field Name | Description |
|---|---|---|
| 0 | L2E | L2 enable. Enables and disables the operation of the L2 cache, starting with the next transaction. |
| 1 | CE | L2 double-bit error checkstop enable. L2 cache double-bit errors can result in a checkstop condition. |
| 2:8 | Reserved | Reserved. |
| 9 | DO | L2 data-only. Setting this bit inhibits the caching of instructions in the L2 cache. All accesses from the L1 instruction cache are treated as cache-inhibited by the L2 cache (bypass L2 cache, no L2 tag look-up performed). |
| 10 | GI | L2 global invalidate. Setting GI invalidates the L2 cache globally by clearing the L2 status bits. |
| 11 | Reserved | Reserved. |
| 12 | WT | L2 write-through. Setting WT selects write-through mode (rather than the default copy-back mode) so all writes to the L2 cache also write through to the 60x bus. |
| 13 | TS | L2 test support. Setting TS 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. If TS is set, it causes single-beat store operations that miss in the L2 cache to be discarded. |
| 14:19 | Reserved | Reserved. |
| 20 | LOCKLO | Lock lower half of the L2 cache (ways 0 and 1). This provides a form of backward compatibility for L2 locking. New applications should use bits 24:25. |
| 21 | LOCKHI | Lock upper half of the L2 cache (ways 2 and 3). This provides a form of backward compatibility for L2 locking. New applications should use bits 26:27. |
| 22 | SHEE | Snoop hit in locked line error enable. Enables a snoop hit in a locked line to raise a machine check. |
| 23 | SHERR | Snoop hit in locked line error. Set by a snoop hit to a locked line. Once set, this sticky bit remains set until cleared by a **mtspr** to the L2CR. |
| 24:27 | LOCK | Cache lock control. Setting one or more of bits 24, 25, 26, and 27 locks ways 0, 1, 2, and 3 respectively |
| 28 | IO | L2 instruction-only. Setting this bit inhibits the caching of data in the L2 cache. All accesses from the L1 data cache are treated as cache-inhibited by the L2 cache (bypass L2 cache, no L2 tag look-up performed). |
| 29:30 | Reserved | Reserved. |
| 31 | IP | L2 global invalidate in progress (read only). This read-only bit indicates whether an L2 global invalidate is occurring. |

## 2.2 Operand Conventions

This section describes the operand conventions as they are represented in two levels of the PowerPC Architecture—UISA and VEA. Detailed descriptions of conventions used for storing values in registers and memory, accessing PowerPC registers, and representing data in these registers can be found in Chapter 3, “Operand Conventions” in the *PowerPC Microprocessor Family: The Programming Environments Manual*.

### 2.2.1 Data Organization in Memory and Data Transfers

Bytes in memory are numbered consecutively starting with 0. Each number is the address of the corresponding byte.

Memory operands can be bytes, half words, words, or double words, or, for the load/store multiple and load/store string instructions, a sequence of bytes or words. The address of a memory operand is the address of its first byte (the lowest-numbered byte). Operand length is implicit for each instruction.

### 2.2.2 Alignment and Misaligned Accesses

The operand of a single-register memory-access instruction has an alignment boundary equal to its length. An operand’s address is misaligned if it is not a multiple of its width. Operands for single-register memory-access instructions have the characteristics shown in *Table 2-4*. Although not permitted as memory operands, quadwords are shown because quadword alignment is desirable for certain memory operands.

*Table 2-4. Memory Operands*

| Operand | Length | Addr[28-31] If Aligned |
|---|---|---|
| Byte | 8 bits | xxxx |
| Half word | 2 bytes | xxx0 |
| Word | 4 bytes | xx00 |
| Double word | 8 bytes | x000 |
| Quadword | 16 bytes | 0000 |

**Note:** An “x” in an address bit position indicates that the bit can be 0 or 1 independent of the state of other bits in the address.

The concept of alignment is also applied more generally to data in memory. For example, a 12-byte data item is said to be word-aligned if its address is a multiple of four.

Some instructions require their memory operands to have a certain alignment. In addition, alignment can affect performance. For single-register memory-access instructions, the best performance is obtained when memory operands are aligned. Instructions are 32 bits (one word) long and must be word-aligned.

The 750GX does not provide hardware support for floating-point memory that is not word-aligned. If a floating-point operand is not aligned, the 750GX invokes an alignment exception, and it is left up to software to break up the offending storage access operation appropriately. In addition, some non-double-word–aligned memory accesses suffer performance degradation as compared to an aligned access of the same type.

In general, floating-point word accesses should always be word-aligned, and floating-point double-word accesses should always be double-word–aligned. Frequent use of misaligned accesses is discouraged since they can degrade overall performance.

### 2.2.3 Floating-Point Operand and Execution Models—UISA

The IEEE 754-1985 standard defines conventions for 64-bit and 32-bit arithmetic. The standard requires that single-precision arithmetic be provided for single-precision operands. The standard permits double-precision arithmetic instructions to have either (or both) single-precision or double-precision operands, but states that single-precision arithmetic instructions should not accept double-precision operands.

The PowerPC UISA follows these guidelines:

- Double-precision arithmetic instructions can have single-precision operands but always produce double-precision results.
- Single-precision arithmetic instructions require all operands to be single-precision and always produce single-precision results.

For arithmetic instructions, conversion from double to single-precision must be done explicitly by software, while conversion from single to double-precision is done implicitly by the processor. For the 750GX, single-precision multiply type instructions usually operate faster than their double-precision equivalents. For details on instruction timings, see *Chapter 6, Instruction Timing,* on page 209.

All PowerPC implementations provide the equivalent of the execution models described in Chapter 3.3 of the *PowerPC Microprocessor Family: The Programming Environments Manual* to ensure that identical results are obtained. The definition of the arithmetic instructions for infinities, denormalized numbers, and not a numbers (NaNs) follow the conventions described in that section.

Although the double-precision format specifies an 11-bit exponent, exponent arithmetic uses two additional bit positions to avoid potential transient overflow conditions. An extra bit is required when denormalized double-precision numbers are prenormalized. A second bit is required to permit computation of the adjusted exponent value in the following examples when the corresponding exception enable bit is one:

- Underflow during multiplication using a denormalized operand
- Overflow during division using a denormalized divisor

The 750GX provides hardware support for all single and double-precision floating-point operations for most value representations and all rounding modes. This architecture provides for hardware to implement a floating-point system as defined in ANSI/IEEE standard 754-1985, *IEEE Standard for Binary Floating Point Arithmetic*. Detailed information about the floating-point execution model can be found in Chapter 3, “Operand Conventions” in the *PowerPC Microprocessor Family: The Programming Environments Manual*.

#### *2.2.3.1 Denormalized Number Support*

The 750GX supports denormalized numbers in hardware. When loading or storing a single-precision denormalized number, the load/store unit converts between the internal double-precision format and the external single-precision format.

#### *2.2.3.2 Non-IEEE Mode (Nondenormalized Mode)*

The 750GX supports a nondenormalized mode of operation. In this mode, when a denormalized result is produced, a default result of zero is generated. The generated zero will have the same sign as the denormalized number. This mode is not strictly IEEE compliant. The 750GX is in this mode when the Floating-Point non-IEEE Enable (NI) bit of the Floating-Point Status and Control Register (FPSCR) is set.

### *2.2.3.3 Time-Critical Floating-Point Operation*

For time-critical applications where deterministic floating-point performance is required, the FPSCR bits must be set with: the non-IEEE mode enabled, the floating-point exception masked, and all sticky bits set to one. With these settings, the 750GX will not cause exceptions nor generate denormalized numbers, either of which slows performance.

### *2.2.3.4 Floating-Point Storage Access Alignment*

The 750GX does not provide hardware support for floating-point storage that is not word aligned. In these cases, the 750GX invokes an alignment exception, and it is left up to software to break up the offending storage access operation appropriately. In addition, some non-double-word-aligned storage accesses will suffer a performance degradation as compared to an aligned access of the same type.

In general, floating-point single-word accesses should always be word aligned and floating-point double-word accesses should always be double-word aligned. The frequent use of misaligned accesses is discouraged since they can compromise the overall performance of the processor.

### *2.2.3.5 Optional Floating-Point Graphics Instructions*

The 750GX implements the graphics instructions Store Floating-Point as Integer Word Indexed (**stfiwx**), Floating Select **fsel(.)**, **fres(.)**, and **frsqrte(.)**. For Floating Reciprocal Estimate Single A-Form (**fres**), the estimate is 12 bits of precision. For Floating Reciprocal Square-root Estimate A-Form (**frsqrte**), the estimate is 12 bits of precision with the remaining bits zero.

*Table 2-5. Floating-Point Operand Data-Type Behavior* (Page 1 of 2)

| Operand A<br>Data Type | Operand B<br>Data Type | Operand C<br>Data Type | IEEE Mode<br>(NI = 0) | Non-IEEE Mode<br>(NI = 1) |
|---|---|---|---|---|
| Single denormalized<br>Double denormalized | Single denormalized<br>Double denormalized | Single denormalized<br>Double denormalized | Normalize all three | Zero all three |
| Single denormalized<br>Double denormalized | Single denormalized<br>Double denormalized | Normalized or zero | Normalize A and B | Zero A and B |
| Normalized or zero | Single denormalized<br>Double denormalized | Single denormalized<br>Double denormalized | Normalize B and C | Zero B and C |
| Single denormalized<br>Double denormalized | Normalized or zero | Single denormalized<br>Double denormalized | Normalize A and C | Zero A and C |
| Single denormalized<br>Double denormalized | Normalized or zero | Normalized or zero | Normalize A | Zero A |
| Normalized or zero | Single denormalized<br>Double denormalized | Normalized or zero | Normalize B | Zero B |
| Normalized or zero | Normalized or zero | Single denormalized<br>Double denormalized | Normalize C | Zero C |
| Single quiet not-a-number (QNaN)<br>Single signaling not-a-number (SNaN)<br>Double QNaN<br>Double SNaN | Don’t care | Don’t care | QNaN[1] | QNaN[1] |

[1] Prioritize according to Chapter 3, “Operand Conventions,” in the *PowerPC Microprocessor Family: The Programming Environments Manual*.

*Table 2-5. Floating-Point Operand Data-Type Behavior* (Page 2 of 2)

| Operand A<br>Data Type | Operand B<br>Data Type | Operand C<br>Data Type | IEEE Mode<br>(NI = 0) | Non-IEEE Mode<br>(NI = 1) |
|---|---|---|---|---|
| Don’t care | Single QNaN<br>Single SNaN<br>Double QNaN<br>Double SNaN | Don’t care | QNaN[1] | QNaN[1] |
| Don’t care | Don’t care | Single QNaN<br>Single SNaN<br>Double QNaN<br>Double SNaN | QNaN[1] | QNaN[1] |
| Single normalized<br>Single infinity<br>Single zero<br>Double normalized<br>Double infinity<br>Double zero | Single normalized<br>Single infinity<br>Single zero<br>Double normalized<br>Double infinity<br>Double zero | Single normalized<br>Single infinity<br>Single zero<br>Double normalized<br>Double infinity<br>Double zero | Do the operation | Do the operation |

1. Prioritize according to Chapter 3, “Operand Conventions,” in the *PowerPC Microprocessor Family: The Programming Environments Manual*.

*Table 2-6* summarizes the mode behavior for results.

*Table 2-6. Floating-Point Result Data-Type Behavior*

| Precision | Data Type | IEEE Mode (NI = 0) | Non-IEEE Mode (NI = 1) |
|---|---|---|---|
| Single | Denormalized | Return single-precision denormalized number with trailing zeros. | Return zero. |
| Single | Normalized, infinity, zero | Return the result. | Return the result. |
| Single | QNaN, SNaN | Return QNaN. | Return QNaN. |
| Single | Integer | Place integer into low word of FPR. | If (Invalid Operation)<br>then<br>Place (0x8000) into FPR[32–63]<br>else<br>Place integer into FPR[32–63]. |
| Double | Denormalized | Return double-precision denormalized number. | Return zero. |
| Double | Normalized, infinity, zero | Return the result. | Return the result. |
| Double | QNaN, SNaN | Return QNaN. | Return QNaN. |
| Double | INT | Not supported by the 750GX | Not supported by the 750GX |

## 2.3 Instruction Set Summary

This section describes instructions and addressing modes defined for the 750GX. These instructions are divided into the following functional categories:

Integer
These include arithmetic and logical instructions. For more information, see *Section 2.3.4.1* on page 92.

Floating-point
These include floating-point arithmetic instructions (single-precision and double-precision), as well as instructions that affect the Floating-Point Status and Control Register (FPSCR). For more information, see *Section 2.3.4.2* on page 95.

Load and store
These include integer and floating-point (including quantized) load-and-store instructions. For more information, see *Section 2.3.4.3* on page 98.

Flow control
These include branching instructions, Condition Register logical instructions, trap instructions, and other instructions that affect the instruction flow. For more information, see *Section 2.3.4.4* on page 106.

Processor control
These instructions are used for synchronizing memory accesses and managing caches, translation lookaside buffers (TLBs), and Segment Registers. For more information, see *Section 2.3.4.6* on page 108, *Section 2.3.5.1* on page 113, and *Section 2.3.6.2* on page 118.

Memory synchronization
These instructions are used for memory synchronizing. For more information, see *Section 2.3.4.7* on page 113 and *Section 2.3.5.2* on page 114.

Memory control
These instructions provide control of caches, TLBs, and Segment Registers. For more information, see *Section 2.3.5.3* on page 115 and *Section 2.3.6.3* on page 119.

External control
These include instructions for use with special input/output devices. For more information, see *Section 2.3.5.4* on page 117.

**Note:** This grouping of instructions does not necessarily indicate the execution unit that processes a particular instruction or group of instructions. That information, which is useful for scheduling instructions most effectively, is provided in *Chapter 6, Instruction Timing,* on page 209.

Integer instructions operate on word operands. Floating-point instructions operate on single-precision and double-precision floating-point operands. The PowerPC Architecture uses instructions that are 4 bytes long and word-aligned. It provides for byte, half-word, and word operand loads and stores between memory and a set of 32 General Purpose Registers (GPRs). It provides for word and double-word operand loads and stores between memory and a set of 32 Floating Point Registers (FPRs).

Arithmetic and logical instructions do not read or modify memory. To use the contents of a memory location in a computation and then modify the same or another memory location, the memory contents must be loaded into a register, modified, and then written to the target location using load-and-store instructions.

The description of each instruction beginning on page 92 includes the mnemonic and a formatted list of operands. To simplify assembly language programming, a set of simplified mnemonics and symbols is provided for some of the frequently-used instructions; see Appendix F, “Simplified Mnemonics,” in the *PowerPC Microprocessor Family: The Programming Environments Manual* for a complete list of simplified mnemonics. Note that the architecture specification refers to simplified mnemonics as extended mnemonics. Programs written to be portable across the various assemblers for the PowerPC Architecture should not assume the existence of mnemonics not described in that document.

### 2.3.1 Classes of Instructions

The 750GX instructions belong to one of the following three classes.

- Defined
- Illegal
- Reserved

Note that while the definitions of these terms are consistent among the PowerPC processors, the assignment of these classifications is not. For example, PowerPC instructions defined for 64-bit implementations are treated as illegal by 32-bit implementations such as the 750GX.

The class is determined by examining the primary opcode and the extended opcode, if any. If the opcode, or combination of opcode and extended opcode, is not that of a defined instruction or of a reserved instruction, the instruction is illegal.

Instruction encodings that are now illegal might be assigned to instructions in the architecture or might be reserved by being assigned to processor-specific instructions.

#### *2.3.1.1 Definition of Boundedly Undefined*

If instructions are encoded with incorrectly set bits in reserved fields, the results on execution can be said to be boundedly undefined. If a user-level program executes the incorrectly coded instruction, the resulting undefined results are bounded in that a spurious change from user to supervisor state is not allowed, and the level of privilege exercised by the program in relation to memory access and other system resources cannot be exceeded. Boundedly-undefined results for a given instruction might vary between implementations, and between execution attempts in the same implementation.

#### *2.3.1.2 Defined Instruction Class*

Defined instructions are guaranteed to be supported in all PowerPC implementations, except as stated in the instruction descriptions in Chapter 8, “Instruction Set,” of the the *PowerPC Microprocessor Family: The Programming Environments Manual*. The 750GX provides hardware support for all instructions defined for 32-bit implementations.

It does not support the optional Floating Square Root (Double-Precision) (**fsqrt**), Floating Square Root (Single-Precision) (**fsqrts**), and Translation Lookaside Buffer Invalidate All (**tlbia)** instructions.

A PowerPC processor invokes the illegal instruction error handler (part of the program exception) when the unimplemented PowerPC instructions are encountered so they can be emulated in software, as required. Note that the architecture specification refers to exceptions as interrupts.

A defined instruction can have invalid forms. The 750GX provides limited support for instructions represented in an invalid form.

#### *2.3.1.3 Illegal Instruction Class*

Illegal instructions can be grouped into the following categories:

- Instructions not defined in the PowerPC Architecture.The following primary opcodes are defined as illegal, but might be defined to perform new functions in future extensions to the architecture:
  1, 4, 5, 6, 9, 22, 56, 60, 61
- Instructions defined in the PowerPC Architecture but not implemented in a specific PowerPC implementation. For example, instructions that can be executed on 64-bit PowerPC processors are considered illegal by 32-bit processors such as the 750GX.

  The following primary opcodes are defined for 64-bit implementations only and are illegal on the 750GX:
  2, 30, 58, 62
- All unused extended opcodes are illegal. The unused extended opcodes can be determined from information in *Section 2.3.1.4*. Notice that extended opcodes for instructions defined only for 64-bit implementations are illegal in 32-bit implementations, and vice versa.

  The following primary opcodes have unused extended opcodes: 17, 19, 31, 59, 63 (primary opcodes 30 and 62 are illegal for all 32-bit implementations, but as 64-bit opcodes they have some unused extended opcodes.)
- An instruction consisting of only zeros is guaranteed to be an illegal instruction. This increases the probability that an attempt to execute data or uninitialized memory invokes the system illegal instruction error handler (a program exception). Note that if only the primary opcode consists of all zeros, the instruction is considered a reserved instruction, as described in *Section 2.3.1.4*.

The 750GX invokes the system illegal instruction error handler (a program exception) when it detects any instruction from this class or any instructions defined only for 64-bit implementations.

See *Section 4.5.7* on page 170 for additional information about illegal and invalid instruction exceptions. Except for an instruction consisting of binary zeros, illegal instructions are available for additions to the PowerPC Architecture.

#### *2.3.1.4 Reserved Instruction Class*

Reserved instructions are allocated to specific implementation-dependent purposes not defined by the PowerPC Architecture. Attempting to execute an unimplemented reserved instruction invokes the illegal instruction error handler (a program exception). See *Section 4.5.7* on page 170 for information about illegal and invalid instruction exceptions.

The PowerPC Architecture defines four types of reserved instructions:

- Instructions in the POWER architecture not part of the PowerPC UISA. For details on POWER architecture incompatibilities and how they are handled by PowerPC processors, see Appendix B, “POWER Architecture Cross Reference” in the *PowerPC Microprocessor Family: The Programming Environments Manual*.
- Implementation-specific instructions required for the processor to conform to the PowerPC Architecture (none of these are implemented in the 750GX)
- All other implementation-specific instructions
- Architecturally-allowed extended opcodes

### 2.3.2 Addressing Modes

This section provides an overview of conventions for addressing memory and for calculating effective addresses as defined by the PowerPC Architecture for 32-bit implementations. For more detailed information, see “Conventions” in Chapter 4, “Addressing Modes and Instruction Set Summary” of the *PowerPC Microprocessor Family: The Programming Environments Manual*.

#### *2.3.2.1 Memory Addressing*

A program references memory using the effective (logical) address computed by the processor when it executes a memory-access or branch instruction or when it fetches the next sequential instruction. Bytes in memory are numbered consecutively starting with zero. Each number is the address of the corresponding byte.

#### *2.3.2.2 Memory Operands*

Memory operands can be bytes, half words, words, or double words, or, for the load/store multiple and load/store string instructions, a sequence of bytes or words. The address of a memory operand is the address of its first byte (that is, of its lowest-numbered byte). Operand length is implicit for each instruction. The PowerPC Architecture supports both big-endian and little-endian byte ordering. The default byte and bit ordering is big-endian. See “Byte Ordering” in Chapter 3, “Operand Conventions” of the *PowerPC Microprocessor Family: The Programming Environments Manual* for more information about big and little-endian byte ordering.

The operand of a single-register memory-access instruction has a natural alignment boundary equal to the operand length. In other words, the “natural” address of an operand is an integral multiple of the operand length. A memory operand is said to be aligned if it is aligned at its natural boundary; otherwise, it is misaligned.

For a detailed discussion about memory operands, see Chapter 3, “Operand Conventions” of the *PowerPC Microprocessor Family: The Programming Environments Manual*.

#### *2.3.2.3 Effective Address Calculation*

An effective address is the 32-bit sum computed by the processor when executing a memory-access or branch instruction or when fetching the next sequential instruction. For a memory-access instruction, if the sum of the effective address and the operand length exceeds the maximum effective address, the memory operand is considered to wrap around from the maximum effective address through effective address 0, as described in the following paragraphs.

Effective address computations for both data and instruction accesses use 32-bit *signed* two’s complement binary arithmetic. A carry from bit 0 and overflow are ignored.

Load-and-store operations have the following modes of effective address generation:

- EA = (**r**A|0) + offset (including offset = 0) (register indirect with immediate index)
- EA = (**r**A|0) + **r**B (register indirect with index)

See *Integer Load-and-Store Address Generation* on page 99 for a detailed description of effective address generation for load-and-store operations.

Branch instructions have three categories of effective address generation:

- Immediate
- Link register indirect
- Count register indirect

#### *2.3.2.4 Synchronization*

The synchronization described in this section refers to the state of the processor that is performing the synchronization.

*Context Synchronization*

The System Call (**sc**) and Return from Interrupt (**rfi**) instructions perform context synchronization by allowing previously issued instructions to complete before performing a change in context. Execution of one of these instructions ensures the following:

- No higher-priority exception exists (**sc**).
- All previous instructions have completed to a point where they can no longer cause an exception. If a prior memory-access instruction causes direct-store error exceptions, the results are guaranteed to be determined before this instruction is executed.
- Previous instructions complete execution in the context (privilege, protection, and address translation) under which they were issued.
- The instructions following the **sc** or **rfi** instruction execute in the context established by these instructions.

*Execution Synchronization*

An instruction is execution synchronizing if all previously initiated instructions appear to have completed before the instruction is initiated, or in the case of **sync** and **isync**, before the instruction completes. For example, the Move-to Machine State Register (**mtmsr**) instruction is execution synchronizing. It ensures that all preceding instructions have completed execution and cannot cause an exception before the instruction executes, but does not ensure that subsequent instructions execute in the newly established environment.

For example, if the **mtmsr** sets the MSR[PR] bit, unless an **isync** immediately follows the **mtmsr** instruction, a privileged instruction could be executed or privileged access could be performed without causing an exception even though the MSR[PR] bit indicates user mode.

*Instruction-Related Exceptions*

There are two kinds of exceptions in the 750GX—those caused directly by the execution of an instruction and those caused by an asynchronous event (or interrupts). Either can cause components of the system software to be invoked.

Exceptions can be caused directly by the execution of an instruction as follows:

- An attempt to execute an illegal instruction causes the illegal instruction (program exception) handler to be invoked. An attempt by a user-level program to execute the supervisor-level instructions listed below causes the privileged instruction (program exception) handler to be invoked:
  - Data Cache Block Invalidate (**dcbi**)
  - Move-from Machine State Register (**mfmsr**)
  - Move-from Special Purpose Register (**mfspr**)
  - Move-from Segment Register (**mfsr**)
  - Move-from Segment Register Indirect (**mfsrin)**
  - Move-to Machine State Register (**mtmsr**)
  - Move-to Special Purpose Register (**mtspr**)
  - Move-to Segment Register (**mtsr**)
  - Move-to Segment Register Indirect (**mtsrin**)
  - Return from Exception (**rfi**)
  - TLB Invalidate Entry (**tlbie**)
  - TLB Synchronize (**tlbsync**)

  Note that the privilege level of the **mfspr** and **mtspr** instructions depends on the SPR encoding.
- Any **mtspr**, **mfspr**, or Move-from Time Base (**mftb**) instruction with an invalid SPR (or Time Base Register [TBR]) field causes an illegal type program exception. Likewise, a program exception is taken if user-level software tries to access a supervisor-level SPR. An **mtspr** instruction executing in supervisor mode (MSR[PR] = 0) with the SPR field specifying HID1 or PVR (read-only registers) executes as a no-op.
- An attempt to access memory that is not available (page fault) causes the ISI or DSI exception handler to be invoked.
- The execution of an **sc** instruction invokes the system-call exception handler that permits a program to request the system to perform a service.
- The execution of a trap instruction invokes the program exception trap handler.
- The execution of an instruction that causes a floating-point exception while exceptions are enabled in the MSR invokes the program exception handler.

A detailed description of exception conditions is provided in *Chapter 4, Exceptions,* on page 151.

### 2.3.3 Instruction Set Overview

This section provides a brief overview of the PowerPC instructions implemented in the 750GX and highlights any special information about how the 750GX implements a particular instruction. Note that the categories used in this section correspond to those used in Chapter 4, "Addressing Modes and Instruction Set

Summary” in the *PowerPC Microprocessor Family: The Programming Environments Manual*. These categorizations are somewhat arbitrary and are provided for the convenience of the programmer and do not necessarily reflect the PowerPC Architecture specification.

Note that some instructions have the following optional features:

- CR Update—The dot (**.**) suffix on the mnemonic enables the update of the CR.
- Overflow option—The **o** suffix indicates that the overflow bit in the XER is enabled.

### 2.3.4 PowerPC UISA Instructions

The PowerPC UISA includes the base user-level instruction set (excluding a few user-level cache-control, synchronization, and time-base instructions), user-level registers, programming model, data types, and addressing modes. This section discusses the instructions defined in the UISA.

#### *2.3.4.1 Integer Instructions*

This section describes the integer instructions, which consist of:

- Integer arithmetic instructions
- Integer compare instructions
- Integer logical instructions
- Integer rotate and shift instructions

Integer instructions use the content of the GPRs as source operands and place results into GPRs, into the Integer Exception Register (XER), and into Condition Register (CR) fields.

*Integer Arithmetic Instructions*

*Table 2-7* lists the integer arithmetic instructions for PowerPC processors.

*Table 2-7. Integer Arithmetic Instructions* (Page 1 of 2)

| Name | Mnemonic | Syntax |
|---|---|---|
| Add Immediate | **addi** | **r**D,**r**A,SIMM |
| Add Immediate Shifted | **addis** | **r**D,**r**A,SIMM |
| Add | **add** (**add. addo addo.**) | **r**D,**r**A,**r**B |
| Subtract From | **subf** (**subf. subfo subfo.**) | **r**D,**r**A,**r**B |
| Add Immediate Carrying | **addic** | **r**D,**r**A,SIMM |
| Add Immediate Carrying and Record | **addic.** | **r**D,**r**A,SIMM |
| Subtract from Immediate Carrying | **subfic** | **r**D,**r**A,SIMM |
| Add Carrying | **addc** (**addc. addco addco.**) | **r**D,**r**A,**r**B |
| Subtract from Carrying | **subfc** (**subfc. subfco subfco.**) | **r**D,**r**A,**r**B |
| Add Extended | **adde** (**adde. addeo addeo.**) | **r**D,**r**A,**r**B |
| Subtract from Extended | **subfe** (**subfe. subfeo subfeo.**) | **r**D,**r**A,**r**B |
| Add to Minus One Extended | **addme** (**addme. addmeo addmeo.**) | **r**D,**r**A |
| Subtract from Minus One Extended | **subfme** (**subfme. subfmeo subfmeo.)** | **r**D,**r**A |

*Table 2-7. Integer Arithmetic Instructions* (Page 2 of 2)

| Name | Mnemonic | Syntax |
|---|---|---|
| Add to Zero Extended | **addze** (**addze. addzeo addzeo.**) | **r**D,**r**A |
| Subtract from Zero Extended | **subfze** (**subfze. subfzeo subfzeo.**) | **r**D,**r**A |
| Negate | **neg** (**neg. nego nego.**) | **r**D,**r**A |
| Multiply Low Immediate | **mulli** | **r**D,**r**A,SIMM |
| Multiply Low | **mullw** (**mullw. mullwo mullwo.**) | **r**D,**r**A,**r**B |
| Multiply High Word | **mulhw** (**mulhw.**) | **r**D,**r**A,**r**B |
| Multiply High Word Unsigned | **mulhwu** (**mulhwu.**) | **r**D,**r**A,**r**B |
| Divide Word | **divw (divw. divwo divwo.)** | **r**D,**r**A,**r**B |
| Divide Word Unsigned | **divwu divwu. divwuo divwuo.** | **r**D,**r**A,**r**B |

Although there is no Subtract Immediate instruction, its effect can be achieved by using an **addi** instruction with the immediate operand negated. Simplified mnemonics are provided that include this negation. The **subf** instructions subtract the second operand (**r**A) from the third operand (**r**B). Simplified mnemonics are provided in which the third operand is subtracted from the second operand. See Appendix F, “Simplified Mnemonics,” in the *PowerPC Microprocessor Family: The Programming Environments Manual* for examples.

The UISA states that an implementation that executes instructions that set the overflow enable bit (OE) or the carry bit (CA) can either execute these instructions slowly or prevent execution of the subsequent instruction until the operation completes. *Chapter 6, Instruction Timing,* on page 209 describes how the 750GX handles CR dependencies. The summary overflow bit (SO) and overflow bit (OV) in the Integer Exception Register are set to reflect an overflow condition of a 32-bit result. This can happen only when OE = 1.

### *Integer Compare Instructions*

The integer compare instructions algebraically or logically compare the contents of register **r**A with either the zero-extended value of the unsigned immediate value (UIMM) operand, the sign-extended value of the signed immediate value (SIMM) operand, or the contents of register **r**B. The comparison is signed for the **cmpi** and **cmp** instructions, and unsigned for the **cmpli** and **cmpl** instructions. *Table 2-8* summarizes the integer compare instructions. For more information, see the *PowerPC Microprocessor Family: The Programming Environments Manual*.

*Table 2-8. Integer Compare Instructions*

| Name | Mnemonic | Syntax[1] |
|---|---|---|
| Compare Immediate | **cmpi** | **crf**D,L,**r**A,SIMM |
| Compare | **cmp** | **crf**D,L,**r**A,**r**B |
| Compare Logical Immediate | **cmpli** | **crf**D,L,**r**A,UIMM |
| Compare Logical | **cmpl** | **crf**D,L,**r**A,**r**B |

[1] See *Conventions Used in This Manual* on page 20.

The **crf**D operand can be omitted if the result of the comparison is to be placed in CR0. Otherwise, the target CR field must be specified in **crf**D, using an explicit field number.

For information on simplified mnemonics for the integer compare instructions see Appendix F, “Simplified Mnemonics,” in the *PowerPC Microprocessor Family: The Programming Environments Manual*.

### *Integer Logical Instructions*

The logical instructions shown in *Table 2-9* on page 94 perform bit-parallel operations on the specified operands. Logical instructions with CR updating enabled (uses dot suffix) and the AND Immediate (**andi.**) and AND Immediate Shifted (**andis.**) instructions set the CR[CR0] field to characterize the result of the logical operation. Logical instructions do not affect XER[SO], XER[OV], or XER[CA].

See Appendix F, “Simplified Mnemonics,” in the *PowerPC Microprocessor Family: The Programming Environments Manual* for simplified mnemonic examples for integer logical operations.

*Table 2-9. Integer Logical Instructions*

| Name | Mnemonic | Syntax | Implementation Notes |
|---|---|---|---|
| AND Immediate | **andi.** | **r**A,**r**S,UIMM | — |
| AND Immediate Shifted | **andis.** | **r**A,**r**S,UIMM | — |
| OR Immediate | **ori** | **r**A,**r**S,UIMM | The PowerPC Architecture defines **ori r0,r0,0** as the preferred form for the no-op instruction. The dispatcher discards this instruction (except for pending trace or breakpoint exceptions). |
| OR Immediate Shifted | **oris** | **r**A,**r**S,UIMM | — |
| XOR Immediate | **xori** | **r**A,**r**S,UIMM | — |
| XOR Immediate Shifted | **xoris** | **r**A,**r**S,UIMM | — |
| AND | **and** (**and.**) | **r**A,**r**S,**r**B | — |
| OR | **or** (**or.**) | **r**A,**r**S,**r**B | — |
| XOR | **xor** (**xor.**) | **r**A,**r**S,**r**B | — |
| NAND | **nand** (**nand.**) | **r**A,**r**S,**r**B | — |
| NOR | **nor** (**nor.**) | **r**A,**r**S,**r**B | — |
| Equivalent | **eqv** (**eqv.**) | **r**A,**r**S,**r**B | — |
| AND with Complement | **andc** (**andc.**) | **r**A,**r**S,**r**B | — |
| OR with Complement | **orc** (**orc.**) | **r**A,**r**S,**r**B | — |
| Extend Sign Byte | **extsb** (**extsb.**) | **r**A,**r**S | — |
| Extend Sign Half Word | **extsh** (**extsh.**) | **r**A,**r**S | — |
| Count Leading Zeros Word | **cntlzw** (**cntlzw.**) | **r**A,**r**S | — |

### *Integer Rotate Instructions*

Rotation operations are performed on data from a GPR, and the result, or a portion of the result, is returned to a GPR. See Appendix F, “Simplified Mnemonics,” in the *PowerPC Microprocessor Family: The Programming Environments Manual* for a complete list of simplified mnemonics that allows simpler coding of often-used functions such as clearing the leftmost or rightmost bits of a register, left justifying or right justifying an arbitrary field, and simple rotates and shifts.

Integer rotate instructions rotate the contents of a register. The result of the rotation is either inserted into the target register under control of a mask (if a mask bit is 1, the associated bit of the rotated data is placed into the target register, and if the mask bit is 0, the associated bit in the target register is unchanged), or ANDed with a mask before being placed into the target register.

The integer rotate instructions are summarized in *Table 2-10*. For more information, see the *PowerPC Microprocessor Family: The Programming Environments Manual*.

*Table 2-10. Integer Rotate Instructions*

| Name | Mnemonic | Syntax |
|---|---|---|
| Rotate Left Word Immediate then AND with Mask | **rlwinm (rlwinm.)** | **r**A,**r**S,SH,MB,ME |
| Rotate Left Word then AND with Mask | **rlwnm (rlwnm.)** | **r**A,**r**S,**r**B,MB,ME |
| Rotate Left Word Immediate then Mask Insert | **rlwimi (rlwimi.)** | **r**A,**r**S,SH,MB,ME |

*Integer Shift Instructions*

The integer shift instructions perform left and right shifts. Immediate-form logical (unsigned) shift operations are obtained by specifying masks and shift values for certain rotate instructions. Simplified mnemonics (shown in Appendix F, “Simplified Mnemonics,” in the *PowerPC Microprocessor Family: The Programming Environments Manual*) are provided to make coding of such shifts simpler and easier to understand.

Multiple-precision shifts can be programmed as shown in Appendix C, “Multiple-Precision Shifts,” in the *PowerPC Microprocessor Family: The Programming Environments Manual*. The integer shift instructions are summarized in *Table 2-11*.

*Table 2-11. Integer Shift Instructions*

| Name | Mnemonic | Syntax |
|---|---|---|
| Shift Left Word | **slw (slw.)** | **r**A,**r**S,**r**B |
| Shift Right Word | **srw (srw.)** | **r**A,**r**S,**r**B |
| Shift Right Algebraic Word Immediate | **srawi (srawi.)** | **r**A,**r**S,SH |
| Shift Right Algebraic Word | **sraw (sraw.)** | **r**A,**r**S,**r**B |

### 2.3.4.2 Floating-Point Instructions

This section describes the floating-point instructions, which include the following:

- Floating-point arithmetic instructions
- Floating-point multiply/add instructions
- Floating-point rounding and conversion instructions
- Floating-point compare instructions
- Floating-point status and control register instructions
- Floating-point move instructions

See *Section 2.3.4.3* on page 98 for information about floating-point loads and stores.

The PowerPC Architecture supports a floating-point system as defined in the IEEE 754-1985 standard, but requires software support to conform with that standard. All floating-point operations conform to the IEEE 754-1985 standard, except if software sets FPSCR[NI] to the non-IEEE mode.

*Floating-Point Arithmetic Instructions*

The floating-point arithmetic instructions are summarized in *Table 2-12*.

*Table 2-12. Floating-Point Arithmetic Instructions*

| Name | Mnemonic | Syntax |
|---|---|---|
| Floating Add (Double-Precision) | **fadd (fadd.)** | **fr**D,**fr**A,**fr**B |
| Floating Add Single | **fadds (fadds.)** | **fr**D,**fr**A,**fr**B |
| Floating Subtract (Double-Precision) | **fsub (fsub.)** | **fr**D,**fr**A,**fr**B |
| Floating Subtract Single | **fsubs (fsubs.)** | **fr**D,**fr**A,**fr**B |
| Floating Multiply (Double-Precision) | **fmul (fmul.)** | **fr**D,**fr**A,**fr**C |
| Floating Multiply Single | **fmuls (fmuls.)** | **fr**D,**fr**A,**fr**C |
| Floating Divide (Double-Precision) | **fdiv (fdiv.)** | **fr**D,**fr**A,**fr**B |
| Floating Divide Single | **fdivs (fdivs.)** | **fr**D,**fr**A,**fr**B |
| Floating Reciprocal Estimate Single[1] | **fres** (**fres.**) | **fr**D,**fr**B |
| Floating Reciprocal Square Root Estimate[1] | **frsqrte** (**frsqrte.**) | **fr**D,**fr**B |
| Floating Select[1] | **fsel** (**fsel.**) | **fr**D,**fr**A,**fr**C,**fr**B |

[1] The **fres**, **frsqrte**, and **fsel** instructions are optional in the PowerPC Architecture.

Double-precision arithmetic instructions, except those involving multiplication (**fmul**, **fmadd**, **fmsub**, **fnmadd**, **fnmsub**) execute with the same latency as their single-precision equivalents. For additional details on floating-point performance, see *Chapter 6, Instruction Timing,* on page 209.

### *Floating-Point Multiply/Add Instructions*

These instructions combine multiply and add operations without an intermediate rounding operation. The floating-point multiply/add instructions are summarized in *Table 2-13*.

*Table 2-13. Floating-Point Multiply/Add Instructions*

| Name | Mnemonic | Syntax |
|---|---|---|
| Floating Multiply/Add (Double-Precision) | **fmadd (fmadd.)** | **fr**D,**fr**A,**fr**C,**fr**B |
| Floating Multiply/Add Single | **fmadds (fmadds.)** | **fr**D,**fr**A,**fr**C,**fr**B |
| Floating Multiply/Subtract (Double-Precision) | **fmsub (fmsub.)** | **fr**D,**fr**A,**fr**C,**fr**B |
| Floating Multiply/Subtract Single | **fmsubs (fmsubs.)** | **fr**D,**fr**A,**fr**C,**fr**B |
| Floating Negative Multiply/Add (Double-Precision) | **fnmadd (fnmadd.)** | **fr**D,**fr**A,**fr**C,**fr**B |
| Floating Negative Multiply/Add Single | **fnmadds (fnmadds.)** | **fr**D,**fr**A,**fr**C,**fr**B |
| Floating Negative Multiply/Subtract (Double-Precision) | **fnmsub (fnmsub.)** | **fr**D,**fr**A,**fr**C,**fr**B |
| Floating Negative Multiply/Subtract Single | **fnmsubs (fnmsubs.)** | **fr**D,**fr**A,**fr**C,**fr**B |

### *Floating-Point Rounding and Conversion Instructions*

The Floating Round to Single-Precision (**frsp**) instruction is used to truncate a 64-bit double-precision number to a 32-bit single-precision floating-point number. The floating-point convert instructions convert a 64-bit double-precision floating-point number to a 32-bit signed integer number.

Examples of uses of these instructions to perform various conversions can be found in Appendix D, “Floating-Point Models,” in the *PowerPC Microprocessor Family: The Programming Environments Manual.*

*Table 2-14. Floating-Point Rounding and Conversion Instructions*

| Name | Mnemonic | Syntax |
|---|---|---|
| Floating Round to Single | **frsp (frsp.)** | **fr**D,**fr**B |
| Floating Convert to Integer Word | **fctiw (fctiw.)** | **fr**D,**fr**B |
| Floating Convert to Integer Word with Round toward Zero | **fctiwz (fctiwz.)** | **fr**D,**fr**B |

### *Floating-Point Compare Instructions*

Floating-point compare instructions compare the contents of two Floating Point Registers. The comparison ignores the sign of zero (that is, +0 = –0).

The floating-point compare instructions are summarized in *Table 2-15*.

*Table 2-15. Floating-Point Compare Instructions*

| Name | Mnemonic | Syntax |
|---|---|---|
| Floating Compare Unordered | **fcmpu** | **crf**D,**fr**A,**fr**B |
| Floating Compare Ordered | **fcmpo** | **crf**D,**fr**A,**fr**B |

The PowerPC Architecture allows an **fcmpu** or **fcmpo** instruction with the record bit (Rc) set to produce a boundedly-undefined result, which might include an illegal instruction program exception. In the 750GX, **crf**D should be treated as undefined

### *Floating-Point Status and Control Register Instructions*

Every FPSCR instruction appears to synchronize the effects of all floating-point instructions executed by a given processor. Executing an FPSCR instruction ensures that all floating-point instructions previously initiated by the given processor appear to have completed before the FPSCR instruction is initiated and that no subsequent floating-point instructions appear to be initiated by the given processor until the FPSCR instruction has completed.

The FPSCR instructions are summarized in *Table 2-16*. For more information, see the *PowerPC Microprocessor Family: The Programming Environments Manual*.

*Table 2-16. Floating-Point Status and Control Register Instructions*

| Name | Mnemonic | Syntax |
|---|---|---|
| Move-from FPSCR | **mffs (mffs.)** | **fr**D |
| Move-to Condition Register from FPSCR | **mcrfs** | **crf**D,**crf**S |
| Move-to FPSCR Field Immediate | **mtfsfi (mtfsfi.)** | **crf**D,IMM |
| Move-to FPSCR Fields | **mtfsf (mtfsf.)** | FM,**fr**B |
| Move-to FPSCR Bit 0 | **mtfsb0 (mtfsb0.)** | **crb**D |
| Move-to FPSCR Bit 1 | **mtfsb1 (mtfsb1.)** | **crb**D |

**Note:** The PowerPC Architecture states that, in some implementations, the move-to FPSCR fields (**mtfsf**) instruction might perform more slowly when only some of the fields are updated as opposed to all of the fields. In the 750GX, there is no degradation of performance.

*Floating-Point Move Instructions*

Floating-point move instructions copy data from one FPR to another. The floating-point move instructions do not modify the FPSCR. The CR update option in these instructions controls the placing of result status into CR1. *Table 2-17* summarizes the floating-point move instructions.

*Table 2-17. Floating-Point Move Instructions*

| Name | Mnemonic | Syntax |
|---|---|---|
| Floating Move Register | **fmr (fmr.)** | **fr**D,**fr**B |
| Floating Negate | **fneg (fneg.)** | **fr**D,**fr**B |
| Floating Absolute Value | **fabs (fabs.)** | **fr**D,**fr**B |
| Floating Negative Absolute Value | **fnabs (fnabs.)** | **fr**D,**fr**B |

### *2.3.4.3 Load-and-Store Instructions*

Load-and-store instructions are issued and translated in program order; however, the accesses can occur out of order. Synchronizing instructions are provided to enforce strict ordering. This section describes the load-and-store instructions, which consist of the following:

- Integer load instructions
- Integer store instructions
- Integer load-and-store with byte-reverse instructions
- Integer load-and-store multiple instructions
- Floating-point load instructions, including quantized loads
- Floating-point store instructions, including quantized stores
- Memory synchronization instructions

The 750GX provides hardware support for misaligned memory accesses. It performs those accesses within a single cycle if the operand lies within a double-word boundary. Misaligned memory accesses that cross a double-word boundary degrade performance.

For string operations, the hardware makes no attempt to combine register values to reduce the number of discrete accesses. Combining stores enhances performance if store gathering is enabled and the accesses meet the criteria described in *Section 6.4.7, Integer Store Gathering,* on page 234. Note that the PowerPC Architecture requires load/store multiple instruction accesses to be aligned. At a minimum, additional cache access cycles are required.

Although many unaligned memory accesses are supported in hardware, the frequent use of them is discouraged since they can compromise the overall performance of the processor.

Accesses that cross a translation boundary might be restarted. That is, a misaligned access that crosses a page boundary is completely restarted if the second portion of the access causes a page fault. This might cause the first access to be repeated. On some processors, such as the PowerPC 603, a TLB reload would cause an instruction restart. On the 750GX, TLB reloads are done transparently, and only a page fault causes a restart.

### *Little Endian Misaligned Accesses*

The 750GX supports misaligned single register load-and-store accesses in little-endian mode without causing an alignment exception. However, execution of a load/store multiple or string instruction causes an alignment exception.

### *Self-Modifying Code*

When a processor modifies a memory location that might be contained in the instruction cache, software must ensure that memory updates are visible to the instruction-fetching mechanism. This can be achieved by the following instruction sequence:

```
dcbst    # update memory
sync     # wait for update
icbi     # remove (invalidate) copy in instruction cache
isync    # remove copy in own instruction buffer
```

These operations are required because the data cache is a write-back cache. Since instruction fetching bypasses the data cache, changes to items in the data cache cannot be reflected in memory until the fetch operations complete.

Special care must be taken to avoid coherency paradoxes in systems that implement unified secondary caches, and designers should carefully follow the guidelines for maintaining cache coherency that are provided in the VEA, and discussed in Chapter 5, “Cache Model and Memory Coherency,” in the *PowerPC Microprocessor Family: The Programming Environments Manual*. Because the 750GX does not broadcast the M bit for instruction fetches, external caches are subject to coherency paradoxes.

### *Integer Load-and-Store Address Generation*

Integer load-and-store operations generate effective addresses using register indirect with immediate index mode, register indirect with index mode, or register indirect mode. See *Section 2.3.2.3* on page 90 for information about calculating effective addresses. Note that in some implementations, operations that are not naturally aligned might suffer performance degradation. See *Section 4.5.6* on page 170 for additional information about load-and-store address alignment exceptions.

### *Integer Load Instructions*

For integer load instructions, the byte, half word, or word addressed by the EA is loaded into **r**D. Many integer load instructions have an update form, in which **r**A is updated with the generated effective address. For these forms, if **r**A ≠ 0 and **r**A ≠ **r**D (otherwise invalid), the EA is placed into **r**A and the memory element (byte, half word, or word) addressed by the EA is loaded into **r**D. Note that the PowerPC Architecture defines load with update instructions with operand **r**A = 0 or **r**A = **r**D as invalid forms.

*Table 2-18* summarizes the integer load instructions.

*Table 2-18. Integer Load Instructions* (Page 1 of 2)

| Name | Mnemonic | Syntax |
|---|---|---|
| Load Byte and Zero | **lbz** | **r**D,d(**r**A) |
| Load Byte and Zero Indexed | **lbzx** | **r**D,rA,**r**B |
| Load Byte and Zero with Update | **lbzu** | **r**D,d(**r**A) |

*Table 2-18. Integer Load Instructions* (Page 2 of 2)

| Name | Mnemonic | Syntax |
|---|---|---|
| Load Byte and Zero with Update Indexed | **lbzux** | **r**D,**r**A,**r**B |
| Load Half Word and Zero | **lhz** | **r**D,d(**r**A**)** |
| Load Half Word and Zero Indexed | **lhzx** | **r**D,**r**A,**r**B |
| Load Half Word and Zero with Update | **lhzu** | **r**D,d(**r**A**)** |
| Load Half Word and Zero with Update Indexed | **lhzux** | **r**D,**r**A,**r**B |
| Load Half Word Algebraic | **lha** | **r**D,d(**r**A**)** |
| Load Half Word Algebraic Indexed | **lhax** | **r**D,**r**A,**r**B |
| Load Half Word Algebraic with Update | **lhau** | **r**D,d(**r**A**)** |
| Load Half Word Algebraic with Update Indexed | **lhaux** | **r**D,**r**A,**r**B |
| Load Word and Zero | **lwz** | **r**D,d(**r**A**)** |
| Load Word and Zero Indexed | **lwzx** | **r**D,**r**A,**r**B |
| Load Word and Zero with Update | **lwzu** | **r**D,d(**r**A**)** |
| Load Word and Zero with Update Indexed | **lwzux** | **r**D,**r**A,**r**B |

**Implementation Notes**—The following notes describe the 750GX implementation of integer load instructions:

- The PowerPC Architecture cautions programmers that some implementations of the architecture might execute the load half algebraic (**lha**, **lhax**) instructions and the load word with update (**lbzu**, **lbzux**, **lhzu**, **lhzux**, **lhau**, **lhaux**, **lwu**, **lwux**) instructions with greater latency than other types of load instructions. This is not the case for the 750GX. These instructions operate with the same latency as other load instructions.
- The PowerPC Architecture cautions programmers that some implementations of the architecture might run the load/store byte-reverse (**lhbrx**, **lbrx**, **sthbrx**, **stwbrx**) instructions with greater latency than other types of load/store instructions. This is not the case for the 750GX. These instructions operate with the same latency as the other load/store instructions.
- The PowerPC Architecture describes some preferred instruction forms for load-and-store multiple instructions and integer move assist instructions that might perform better than other forms in some implementations. None of these preferred forms affect instruction performance on the 750GX.
- The PowerPC Architecture defines the load word and reserve indexed (**lwarx**) and the store word conditional indexed (**stwcx.**) instructions as a way to update memory atomically. In the 750GX, reservations are made on behalf of aligned 32-byte sections of the memory address space. Executing **lwarx** and **stwcx.** to a page marked write-through does not cause a DSI exception if the write-through (W) bit is set. However, as with other memory accesses, DSI exceptions can result for other reasons such as protection violations or page faults.
- In general, because **stwcx.** always causes an external bus transaction, it has slightly worse performance characteristics than normal store operations.

### *Integer Store Instructions*

For integer store instructions, the contents of the source register (**r**S) are stored into the byte, half word, or word in memory addressed by the EA. Many store instructions have an update form, in which **r**A is updated with the EA. For these forms, the following rules apply:

- If **r**A ≠ 0, the effective address is placed into **r**A.
- If **r**S = **r**A, the contents of register **r**S are copied to the target memory element, and then the generated EA is placed into **r**A (**r**S).

The PowerPC Architecture defines store with update instructions with **r**A = 0 as an invalid form. In addition, it defines integer store instructions with the CR update option enabled (Rc field, bit 31, in the instruction encoding = 1) to be an invalid form.

*Table 2-19* summarizes the integer store instructions.

*Table 2-19. Integer Store Instructions*

| Name | Mnemonic | Syntax |
|---|---|---|
| Store Byte | **stb** | **r**S**,d(r**A**)** |
| Store Byte Indexed | **stbx** | **r**S**,r**A**,r**B |
| Store Byte with Update | **stbu** | **r**S**,d(r**A**)** |
| Store Byte with Update Indexed | **stbux** | **r**S**,r**A**,r**B |
| Store Half Word | **sth** | **r**S**,d(r**A**)** |
| Store Half Word Indexed | **sthx** | **r**S**,r**A**,r**B |
| Store Half Word with Update | **sthu** | **r**S**,d(r**A**)** |
| Store Half Word with Update Indexed | **sthux** | **r**S**,r**A**,r**B |
| Store Word | **stw** | **r**S**,d(r**A**)** |
| Store Word Indexed | **stwx** | **r**S**,r**A**,r**B |
| Store Word with Update | **stwu** | **r**S**,d(r**A**)** |
| Store Word with Update Indexed | **stwux** | **r**S**,r**A**,r**B |

### *Integer Store Gathering*

The 750GX performs store gathering for write-through accesses to nonguarded space or to cache-inhibited stores to nonguarded space if the stores are 4 bytes and they are word-aligned. These stores are combined in the load/store unit (LSU) to form a double word that is sent out on the 60x bus as a single-beat operation. Stores are gathered only if successive, eligible stores are queued and pending. Store gathering takes place regardless of address order or endian mode. The store-gathering feature is enabled by setting the HID0[SGE] bit (bit 24).

Store gathering is not done for:

- Cacheable stores
- Stores to guarded cache-inhibited or write-through space
- Byte-reverse store
- Store Word Conditional Indexed (**stwcx.**) and External Control Out Word Indexed (**ecowx**) accesses
- Floating-point stores

If store gathering is enabled and the stores do not fall under the above categories, then an Enforce In-Order Execution of I/O (**eieio)** or Synchronize (**sync**) instruction must be used to prevent two stores from being gathered.

Store gathering is also *not* done when the MMU is busy doing a hardware table walk.

### *Integer Load-and-Store with Byte-Reverse Instructions*

*Table 2-20* describes integer load-and-store with byte-reverse instructions. When used in a PowerPC system operating with the default big-endian byte order, these instructions have the effect of loading and storing data in little-endian order. Likewise, when used in a PowerPC system operating with little-endian byte order, these instructions have the effect of loading and storing data in big-endian order. For more information about big-endian and little-endian byte ordering, see "Byte Ordering" in Chapter 3, "Operand Conventions" in the *PowerPC Microprocessor Family: The Programming Environments Manual*.

*Table 2-20. Integer Load-and-Store with Byte-Reverse Instructions*

| Name | Mnemonic | Syntax |
|---|---|---|
| Load Half Word Byte-Reverse Indexed | **lhbrx** | **r**D**,r**A**,r**B |
| Load Word Byte-Reverse Indexed | **lwbrx** | **r**D**,r**A**,r**B |
| Store Half Word Byte-Reverse Indexed | **sthbrx** | **r**S**,r**A**,r**B |
| Store Word Byte-Reverse Indexed | **stwbrx** | **r**S**,r**A**,r**B |

### *Integer Load-and-Store Multiple Instructions*

The load/store multiple instructions are used to move blocks of data to and from the GPRs. The load multiple and store multiple instructions can have operands that require memory accesses that cross a 4-KB page boundary. As a result, these instructions might be interrupted by a DSI exception associated with the address translation of the second page.

**Implementation Notes:** The following describes the 750GX implementation of the load/store multiple instruction.

- For load/store string operations, the hardware does not combine register values to reduce the number of discrete accesses. However, if store gathering is enabled and the accesses fall under the criteria for store gathering, the stores can be combined to enhance performance. At a minimum, additional cache access cycles are required.
- The 750GX supports misaligned, single-register load-and-store accesses in little-endian mode without causing an alignment exception. However, execution of misaligned load/store multiple/string operations causes an alignment exception.

The PowerPC Architecture defines the Load Multiple Word (**lmw**) instruction with **r**A in the range of registers to be loaded as an invalid form.

*Table 2-21. Integer Load-and-Store Multiple Instructions*

| Name | Mnemonic | Syntax |
|---|---|---|
| Load Multiple Word | **lmw** | **r**D**,d(r**A**)** |
| Store Multiple Word | **stmw** | **r**S**,d(r**A**)** |

### *Integer Load-and-Store String Instructions*

The integer load-and-store string instructions allow movement of data from memory to registers, or from registers to memory, without concern for alignment. These instructions can be used for a short move between arbitrary memory locations or to initiate a long move between misaligned memory fields. However, in some implementations, these instructions are likely to have greater latency and take longer to execute, perhaps much longer, than a sequence of individual load or store instructions that produce the same results. *Table 2-20* summarizes the integer load-and-store string instructions. In other PowerPC implementations operating with little-endian byte order, execution of a load or string instruction invokes the alignment error handler. See "Byte Ordering" in the *PowerPC Microprocessor Family: The Programming Environments Manual* for more information.

*Table 2-22. Integer Load-and-Store String Instructions*

| Name | Mnemonic | Syntax |
|---|---|---|
| Load String Word Immediate | **lswi** | **r**D,**r**A,NB |
| Load String Word Indexed | **lswx** | **r**D,**r**A,**r**B |
| Store String Word Immediate | **stswi** | **r**S,**r**A,NB |
| Store String Word Indexed | **stswx** | **r**S,**r**A,**r**B |

Load string and store string instructions might involve operands that are not word-aligned.

As described in *Section 4.5.6* on page 170, a misaligned string operation suffers a performance penalty compared to an aligned operation of the same type.

A non-word-aligned string operation that crosses a 4-KB boundary, or a word-aligned string operation that crosses a 256-MB boundary, always causes an alignment exception. A non-word-aligned string operation that crosses a double-word boundary is also slower than a word-aligned string operation.

**Implementation Notes:** The following describes the 750GX implementation of load/store string instructions:

- For load/store string operations, the hardware does not combine register values to reduce the number of discrete accesses. However, if store gathering is enabled and the accesses fall under the criteria for store gathering, the stores can be combined to enhance performance. At a minimum, additional cache access cycles are required.
- The 750GX supports misaligned, single-register load-and-store accesses in little-endian mode without causing an alignment exception. However, execution of misaligned load/store multiple/string operations causes an alignment exception.

### *Floating-Point Load-and-Store Address Generation*

Floating-point load-and-store operations generate effective addresses using the register indirect with immediate index addressing mode and register indirect with index addressing mode. Floating-point loads and stores are not supported for direct-store accesses. The use of floating-point loads and stores for direct-store access results in an alignment exception.

**Implementation Notes:** The 750GX treats exceptions as follows:

- The FPU can be run in two different modes—ignore-exceptions mode (MSR[FE0] = MSR[FE1] = 0) and precise-exception mode (any other settings for MSR[FE0,FE1]). For the 750GX, ignore-exceptions mode allows floating-point instructions to complete earlier and, thus, might provide better performance than precise mode.

For software compatibility, the other two mode encodings, imprecise-nonrecoverable mode and imprecise-recoverable mode, default to the precise mode.

**Note:** For the 750GX, the ignore-exceptions mode allows floating-point instructions to complete earlier and, thus, might provide better performance than the precise-exception mode.

- The floating-point load-and-store indexed instructions (**lfsx**, **lfsux**, **lfdx**, **lfdux**, **stfsx**, **stfsux**, **stfdx**, **stfdux**) are invalid when the Rc bit is one. In the 750GX, executing one of these invalid instruction forms causes CR0 to be set to an undefined value.

### *Floating-Point Load Instructions*

There are two forms of the floating-point load instruction—single-precision and double-precision. The behavior of double-precision floating-point load instructions, and the behavior of single-precision floating-point load instructions are described here. Single-precision floating-point load instructions convert single-precision data to double-precision format before loading an operand into an FPR.

The PowerPC Architecture defines a load with update instruction with **r**A = 0 as an invalid form.

*Table 2-23* summarizes the single-precision and double-precision floating-point load instructions.

*Table 2-23. Floating-Point Load Instructions*

| Name | Mnemonic | Syntax |
|---|---|---|
| Load Floating-Point Single | **lfs** | **fr**D**,d(r**A**)** |
| Load Floating-Point Single Indexed | **lfsx** | **fr**D**,r**A**,r**B |
| Load Floating-Point Single with Update | **lfsu** | **fr**D**,d(r**A**)** |
| Load Floating-Point Single with Update Indexed | **lfsux** | **fr**D**,r**A**,r**B |
| Load Floating-Point Double | **lfd** | **fr**D**,d(r**A**)** |
| Load Floating-Point Double Indexed | **lfdx** | **fr**D**,r**A**,r**B |
| Load Floating-Point Double with Update | **lfdu** | **fr**D**,d(r**A**)** |
| Load Floating-Point Double with Update Indexed | **lfdux** | **fr**D**,r**A**,r**B |

### *Floating-Point Store Instructions*

This section describes floating-point store instructions. There are three basic forms of the store instruction—single-precision, double-precision, and integer. The integer form is supported by the optional **stfiwx** instruction. The behavior of double-precision floating-point store instructions, and the behavior of single-precision floating-point store instructions are described here. Single-precision floating-point store instructions convert double-precision data to single-precision format before storing the operands.

**Programming Note:** After power-on-reset, never store data from the Floating Point Register file because the file contains unset data and might have invalid formatted floating-point data. Always initialize the Floating Point Register file with valid floating-point data before continuing after a power-on-reset,.

*Table 2-24* summarizes the single-precision and double-precision floating-point store and **stfiwx** instructions.

*Table 2-24. Floating-Point Store Instructions*

| Name | Mnemonic | Syntax |
|---|---|---|
| Store Floating-Point Single | **stfs** | **fr**S**,d(r**A**)** |
| Store Floating-Point Single Indexed | **stfsx** | **fr**S**,r**B |
| Store Floating-Point Single with Update | **stfsu** | **fr**S**,d(**rA**)** |
| Store Floating-Point Single with Update Indexed | **stfsux** | **fr**S**,r**B |
| Store Floating-Point Double | **stfd** | **fr**S**,d(r**A**)** |
| Store Floating-Point Double Indexed | **stfdx** | **fr**S**,r**B |
| Store Floating-Point Double with Update | **stfdu** | **fr**S**,d(**rA**)** |
| Store Floating-Point Double with Update Indexed | **stfdux** | **fr**S**,r**B |
| Store Floating-Point as Integer Word Indexed [1] | **stfiwx** | **fr**S**,r**B |

[1] The **stfiwx** instruction is optional in the PowerPC Architecture.

Some floating-point store instructions require conversions in the LSU. *Table 2-25* shows conversions the LSU makes when executing a Store Floating-Point Single instruction.

*Table 2-25. Store Floating-Point Single Behavior*

| FPR Precision | Data Type | Action |
|---|---|---|
| Single | Normalized | Store |
| Single | Denormalized | Store |
| Single | Zero, infinity, QNaN | Store |
| Single | SNaN | Store |
| Double | Normalized | If (exp ð ≤ 896)<br>then Denormalize and Store<br>else<br>Store |
| Double | Denormalized | Store zero |
| Double | Zero, infinity, QNaN | Store |
| Double | SNaN | Store |

**Note:** The FPRs are not initialized by $\overline{\text{HRESET}}$, and they must be initialized with some valid value after POR HRESET and before being stored.

*Table 2-26* shows the conversions made when performing a Store Floating-Point Double instruction. Most entries in the table indicate that the floating-point value is simply stored. Only in a few cases are any other actions taken.

*Table 2-26. Store Floating-Point Double Behavior* (Page 1 of 2)

| FPR Precision | Data Type | Action |
|---|---|---|
| Single | Normalized | Store |
| Single | Denormalized | Normalize and Store |
| Single | Zero, infinity, QNaN | Store |

*Table 2-26. Store Floating-Point Double Behavior* (Page 2 of 2)

| FPR Precision | Data Type | Action |
|---|---|---|
| Single | SNaN | Store |
| Double | Normalized | Store |
| Double | Denormalized | Store |
| Double | Zero, infinity, QNaN | Store |
| Double | SNaN | Store |

Architecturally, all single-precision and double-precision floating-point numbers are represented in double-precision format within the 750GX. Execution of a store floating-point single (**stfs**, **stfsu**, **stfsx**, **stfsux**) instruction requires conversion from double to single-precision format. If the exponent is not greater than 896, this conversion requires denormalization. The 750GX supports this denormalization by shifting the mantissa one bit at a time. Anywhere from 1 to 23 clock cycles are required to complete the denormalization, depending upon the value to be stored.

Because of how floating-point numbers are implemented in the 750GX, there is also a case when execution of a store floating-point double (**stfd**, **stfdu**, **stfdx**, **stfdux**) instruction can require internal shifting of the mantissa. This case occurs when the operand of a store floating-point double instruction is a denormalized single-precision value. The value could be the result of a load floating-point single instruction, a single-precision arithmetic instruction, or a floating round to single-precision instruction. In these cases, shifting the mantissa takes from 1 to 23 clock cycles, depending upon the value to be stored. These cycles are incurred during the store.

### *2.3.4.4 Branch and Flow-Control Instructions*

Some branch instructions can redirect instruction execution conditionally based on the value of bits in the CR. When the processor encounters one of these instructions, it scans the execution pipelines to determine whether an instruction in progress can affect the particular CR bit. If no interlock is found, the branch can be resolved immediately by checking the bit in the CR and taking the action defined for the branch instruction.

#### *Branch Instruction Address Calculation*

Branch instructions can alter the sequence of instruction execution. Instruction addresses are always assumed to be word aligned; the PowerPC processors ignore the two low-order bits of the generated branch target address. Branch instructions compute the EA of the next instruction address using the following addressing modes:

- Branch relative
- Branch conditional to relative address
- Branch to absolute address
- Branch conditional to absolute address
- Branch conditional to link register
- Branch conditional to count register

**Note:** In the 750GX, all branch instructions (**b**, **ba**, **bl**, **bla**, **bc**, **bca**, **bcl**, **bcla**, **bclr**, **bclrl**, **bcctr**, **bcctrl**) and condition register logical instructions (**crand**, **cror**, **crxor**, **crnand**, **crnor**, **crandc**, **creqv**, **crorc**, and **mcrf**) are executed by the branch processing unit (BPU). Some of these instructions can redirect instruction execution conditionally based on the value of bits in the CR. Whenever the CR bits resolve, the branch direction is either marked as correct or mispredicted. Correcting a mispredicted branch requires that the 750GX flush speculatively executed instructions and restore the machine state to immediately after the branch. This correction can be done immediately upon resolution of the Condition Registers bits.

### *Branch Instructions*

*Table 2-27* lists the branch instructions provided by the PowerPC processors. To simplify assembly language programming, a set of simplified mnemonics and symbols is provided for the most frequently used forms of branch conditional, compare, trap, rotate and shift, and certain other instructions. See Appendix F, “Simplified Mnemonics” in the *PowerPC Microprocessor Family: The Programming Environments Manual* for a list of simplified mnemonic examples.

*Table 2-27. Branch Instructions*

| Name | Mnemonic | Syntax |
|---|---|---|
| Branch | **b (ba bl bla)** | target_addr |
| Branch Conditional | **bc (bca bcl bcla)** | BO**,**BI,target_addr |
| Branch Conditional to Link Register | **bclr (bclrl**) | BO**,**BI |
| Branch Conditional to Count Register | **bcctr (bcctrl**) | BO**,**BI |

### *Condition Register Logical Instructions*

Condition Register logical instructions and the Move Condition Register Field (**mcrf**) instruction are also defined as flow-control instructions. *Table 2-28* shows these instructions.

*Table 2-28. Condition Register Logical Instructions*

| Name | Mnemonic | Syntax |
|---|---|---|
| Condition Register AND | **crand** | **crb**D**,crb**A**,crb**B |
| Condition Register OR | **cror** | **crb**D**,crb**A**,crb**B |
| Condition Register XOR | **crxor** | **crb**D**,crb**A**,crb**B |
| Condition Register NAND | **crnand** | **crb**D**,crb**A**,crb**B |
| Condition Register NOR | **crnor** | **crb**D**,crb**A**,crb**B |
| Condition Register Equivalent | **creqv** | **crb**D**,crb**A**,crb**B |
| Condition Register AND with Complement | **crandc** | **crb**D**,crb**A**,crb**B |
| Condition Register OR with Complement | **crorc** | **crb**D**,crb**A**,crb**B |
| Move Condition Register Field | **mcrf** | **crf**D**,crf**S |

**Note:** If the LR update option is enabled for any of these instructions, the PowerPC Architecture defines these forms of the instructions as invalid.

*Trap Instructions*

The trap instructions shown in *Table 2-29* are provided to test for a specified set of conditions. If any of the conditions tested by a trap instruction are met, the system trap type of program exception is taken. For more information, see *Section 4.5.7* on page 170. If the tested conditions are not met, instruction execution continues normally.

*Table 2-29. Trap Instructions*

| Name | Mnemonic | Syntax |
|---|---|---|
| Trap Word Immediate | **twi** | TO,rA,SIMM |
| Trap Word | **tw** | TO,**r**A,**r**B |

See Appendix F, “Simplified Mnemonics” in the *PowerPC Microprocessor Family: The Programming Environments Manual* for a complete set of simplified mnemonics.

### *2.3.4.5 System Linkage Instruction—UISA*

The System Call (**sc**) instruction permits a program to call on the system to perform a service; see *Table 2-30*. See also *Section 2.3.6.1* on page 118 for additional information.

*Table 2-30. System Linkage Instruction—UISA*

| Name | Mnemonic | Syntax |
|---|---|---|
| System Call | **sc** | — |

Executing this instruction causes the system-call exception handler to be evoked. For more information, see *Section 4.5.10* on page 171.

### *2.3.4.6 Processor Control Instructions—UISA*

Processor control instructions are used to read from and write to the Condition Register (CR), Machine State Register (MSR), and Special-Purpose Registers (SPRs).

See *Section 2.3.5.1* on page 113 for the **mftb** instruction and *Section 2.3.6.2* on page 118 for information about the instructions used for reading from and writing to the MSR and SPRs.

*Move-to/Move-from Condition Register Instructions*

*Table 2-31* summarizes the instructions for reading from or writing to the Condition Register.

*Table 2-31. Move-to/Move-from Condition Register Instructions*

| Name | Mnemonic | Syntax |
|---|---|---|
| Move-to Condition Register Fields | **mtcrf** | CRM,**r**S |
| Move-to Condition Register from XER | **mcrxr** | **crf**D |
| Move-from Condition Register | **mfcr** | **r**D |

**Implementation Note:** The PowerPC Architecture indicates that in some implementations the Move-to Condition Register Fields (**mtcrf**) instruction might perform more slowly when only a portion of the fields are updated as opposed to all of the fields. The Condition Register access latency for the 750GX is the same in both cases.

### *Move-to/Move-from Special-Purpose Register Instructions (UISA)*

*Table 2-32* lists the **mtspr** and **mfspr** instructions.

*Table 2-32. Move-to/Move-from Special-Purpose Register Instructions (UISA)*

| Name | Mnemonic | Syntax |
|---|---|---|
| Move-to Special-Purpose Register | **mtspr** | SPR,**r**S |
| Move-from Special-Purpose Register | **mfspr** | **r**D,SPR |

*Table 2-33* lists the SPR numbers for both user-level and supervisor-level accesses.

*Table 2-33. PowerPC Encodings* (Page 1 of 3)

| Register Name | SPR[1] | | | Access | **mfspr**/**mtspr** |
|---|---|---|---|---|---|
| | Decimal | SPR[5–9] | SPR[0–4] | | |
| CTR | 9 | 00000 | 01001 | User (UISA) | Both |
| DABR | 1013 | 11111 | 10101 | Supervisor (OEA) | Both |
| DAR | 19 | 00000 | 10011 | Supervisor (OEA) | Both |
| DBAT0L | 537 | 10000 | 11001 | Supervisor (OEA) | Both |
| DBAT0U | 536 | 10000 | 11000 | Supervisor (OEA) | Both |
| DBAT1L | 539 | 10000 | 11011 | Supervisor (OEA) | Both |
| DBAT1U | 538 | 10000 | 11010 | Supervisor (OEA) | Both |
| DBAT2L | 541 | 11110 | 11101 | Supervisor (OEA) | Both |
| DBAT2U | 540 | 11110 | 11100 | Supervisor (OEA) | Both |
| DBAT3L | 543 | 11110 | 11111 | Supervisor (OEA) | Both |
| DBAT3U | 542 | 11110 | 11110 | Supervisor (OEA) | Both |
| DBAT4L | 569 | 10001 | 11001 | Supervisor (OEA) | Both |
| DBAT4U | 568 | 10001 | 11000 | Supervisor (OEA) | Both |
| DBAT5L | 571 | 10001 | 11011 | Supervisor (OEA) | Both |
| DBAT5U | 570 | 10001 | 11010 | Supervisor (OEA) | Both |
| DBAT6L | 573 | 10001 | 11101 | Supervisor (OEA) | Both |
| DBAT6U | 572 | 10001 | 11100 | Supervisor (OEA) | Both |

**Note:**

1. The order of the two 5-bit halves of the SPR number is reversed compared with actual instruction coding. For **mtspr** and **mfspr** instructions, the SPR number coded in assembly language does not appear directly as a 10-bit binary number in the instruction. The number coded is split into two 5-bit halves that are reversed in the instruction, with the high-order five bits appearing in bits 16–20 of the instruction and the low-order five bits in bits 11–15.
2. The TB Registers are referred to as TBRs rather than SPRs and can be written to using the **mtspr** instruction in supervisor mode and the TBR numbers here. The TB Registers can be read in user mode using either the **mftb** or **mfspr** instruction and specifying TBR 268 for TBL and SPR 269 for TBU.

*Table 2-33. PowerPC Encodings* (Page 2 of 3)

| Register Name | SPR[1] | | | Access | **mfspr**/**mtspr** |
|---|---|---|---|---|---|
| | Decimal | SPR[5–9] | SPR[0–4] | | |
| DBAT7L | 575 | 10001 | 11111 | Supervisor (OEA) | Both |
| DBAT7U | 574 | 10001 | 11110 | Supervisor (OEA) | Both |
| DEC | 22 | 00000 | 10110 | Supervisor (OEA) | Both |
| DSISR | 18 | 00000 | 10010 | Supervisor (OEA) | Both |
| EAR | 282 | 01000 | 11010 | Supervisor (OEA) | Both |
| IBAT0L | 529 | 10000 | 10001 | Supervisor (OEA) | Both |
| IBAT0U | 528 | 10000 | 10000 | Supervisor (OEA) | Both |
| IBAT1L | 531 | 10000 | 10011 | Supervisor (OEA) | Both |
| IBAT1U | 530 | 10000 | 10010 | Supervisor (OEA) | Both |
| IBAT2L | 533 | 10000 | 10101 | Supervisor (OEA) | Both |
| IBAT2U | 532 | 10000 | 10100 | Supervisor (OEA) | Both |
| IBAT3L | 535 | 10000 | 10111 | Supervisor (OEA) | Both |
| IBAT3U | 534 | 10000 | 10110 | Supervisor (OEA) | Both |
| IBAT4L | 561 | 10001 | 10001 | Supervisor (OEA) | Both |
| IBAT4U | 560 | 10001 | 10000 | Supervisor (OEA) | Both |
| IBAT5L | 563 | 10001 | 10011 | Supervisor (OEA) | Both |
| IBAT5U | 562 | 10001 | 10010 | Supervisor (OEA) | Both |
| IBAT6L | 565 | 10001 | 10101 | Supervisor (OEA) | Both |
| IBAT6U | 564 | 10001 | 10100 | Supervisor (OEA) | Both |
| IBAT7L | 567 | 10001 | 10111 | Supervisor (OEA) | Both |
| IBAT7U | 566 | 10001 | 10110 | Supervisor (OEA) | Both |
| LR | 8 | 00000 | 01000 | User (UISA) | Both |
| PVR | 287 | 01000 | 11111 | Supervisor (OEA) | **mfspr** |
| SDR1 | 25 | 00000 | 11001 | Supervisor (OEA) | Both |
| SPRG0 | 272 | 01000 | 10000 | Supervisor (OEA) | Both |
| SPRG1 | 273 | 01000 | 10001 | Supervisor (OEA) | Both |
| SPRG2 | 274 | 01000 | 10010 | Supervisor (OEA) | Both |
| SPRG3 | 275 | 01000 | 10011 | Supervisor (OEA) | Both |
| SRR0 | 26 | 00000 | 11010 | Supervisor (OEA) | Both |
| SRR1 | 27 | 00000 | 11011 | Supervisor (OEA) | Both |

**Note:**

1. The order of the two 5-bit halves of the SPR number is reversed compared with actual instruction coding. For **mtspr** and **mfspr** instructions, the SPR number coded in assembly language does not appear directly as a 10-bit binary number in the instruction. The number coded is split into two 5-bit halves that are reversed in the instruction, with the high-order five bits appearing in bits 16–20 of the instruction and the low-order five bits in bits 11–15.
2. The TB Registers are referred to as TBRs rather than SPRs and can be written to using the **mtspr** instruction in supervisor mode and the TBR numbers here. The TB Registers can be read in user mode using either the **mftb** or **mfspr** instruction and specifying TBR 268 for TBL and SPR 269 for TBU.

*Table 2-33. PowerPC Encodings* (Page 3 of 3)

| Register Name | SPR[1] | | | Access | **mfspr**/**mtspr** |
|---|---|---|---|---|---|
| | Decimal | SPR[5–9] | SPR[0–4] | | |
| TBL[2] | 268 | 01000 | 01100 | User (VEA) | **mfspr** |
| | 284 | 01000 | 11100 | Supervisor (OEA) | **mtspr** |
| TBU[2] | 269 | 01000 | 01101 | User (VEA) | **mfspr** |
| | 285 | 01000 | 11101 | Supervisor (OEA) | **mtspr** |
| XER | 1 | 00000 | 00001 | User (UISA) | Both |

**Note:**

1. The order of the two 5-bit halves of the SPR number is reversed compared with actual instruction coding. For **mtspr** and **mfspr** instructions, the SPR number coded in assembly language does not appear directly as a 10-bit binary number in the instruction. The number coded is split into two 5-bit halves that are reversed in the instruction, with the high-order five bits appearing in bits 16–20 of the instruction and the low-order five bits in bits 11–15.
2. The TB Registers are referred to as TBRs rather than SPRs and can be written to using the **mtspr** instruction in supervisor mode and the TBR numbers here. The TB Registers can be read in user mode using either the **mftb** or **mfspr** instruction and specifying TBR 268 for TBL and SPR 269 for TBU.

Encodings for the 750GX-specific SPRs are listed in *Table 2-34*.

*Table 2-34. SPR Encodings for 750GX-Defined Registers (mfspr)*

| Register Name | SPR[1] | | | Access | **mfspr**/**mtspr** |
|---|---|---|---|---|---|
| | Decimal | SPR[5–9] | SPR[0–4] | | |
| DABR | 1013 | 11111 | 10101 | User | Both |
| HID0 | 1008 | 11111 | 10000 | Supervisor | Both |
| HID1 | 1009 | 11111 | 10001 | Supervisor | Both |
| HID2 | 1016 | 11111 | 11000 | Supervisor | Both |
| IABR | 1010 | 11111 | 10010 | Supervisor | Both |
| ICTC | 1019 | 11111 | 11011 | Supervisor | Both |
| L2CR | 1017 | 11111 | 11001 | Supervisor | Both |
| MMCR0 | 952 | 11101 | 11000 | Supervisor | Both |
| MMCR1 | 956 | 11101 | 11100 | Supervisor | Both |
| PMC1 | 953 | 11101 | 11001 | Supervisor | Both |
| PMC2 | 954 | 11101 | 11010 | Supervisor | Both |
| PMC3 | 957 | 11101 | 11101 | Supervisor | Both |
| PMC4 | 958 | 11101 | 11110 | Supervisor | Both |
| Reserved[2] | 921–924 | | | Supervisor | |
| SIA | 955 | 11101 | 11011 | Supervisor | Both |
| THRM1 | 1020 | 11111 | 11100 | Supervisor | Both |
| THRM2 | 1021 | 11111 | 11101 | Supervisor | Both |
| THRM3 | 1022 | 11111 | 11110 | Supervisor | Both |
| THRM4 | 920 | 11100 | 11000 | Supervisor | **mfspr** |
| UMMCR0 | 936 | 11101 | 01000 | User | **mfspr** |
| UMMCR1 | 940 | 11101 | 01100 | User | **mfspr** |
| UPMC1 | 937 | 11101 | 01001 | User | **mfspr** |
| UPMC2 | 938 | 11101 | 01010 | User | **mfspr** |
| UPMC3 | 941 | 11101 | 01101 | User | **mfspr** |
| UPMC4 | 942 | 11101 | 01110 | User | **mfspr** |
| USIA | 939 | 11101 | 01011 | User | **mfspr** |

**Note:**

1. The order of the two 5-bit halves of the SPR number is reversed compared with actual instruction coding.
   For **mtspr** and **mfspr** instructions, the SPR number coded in assembly language does not appear directly as a 10-bit binary number in the instruction. The number coded is split into two 5-bit halves that are reversed in the instruction, with the high-order 5 bits appearing in bits 16–20 of the instruction and the low-order 5 bits in bits 11–15.
2. These registers are reserved for future use and the contents should not be changed or reset.

### *2.3.4.7 Memory Synchronization Instructions—UISA*

Memory synchronization instructions control the order in which memory operations are completed with respect to asynchronous events, and the order in which memory operations are seen by other processors or memory-access mechanisms. See *Chapter 3, Instruction-Cache and Data-Cache Operation,* on page 121 for additional information about these instructions and about related aspects of memory synchronization. See *Table 2-35* for a summary.

*Table 2-35. Memory Synchronization Instructions—UISA*

| Name | Mnemonic | Syntax | Implementation Notes |
|---|---|---|---|
| Load Word and Reserve Indexed | **lwarx** | **r**D,**r**A,**r**B | Programmers can use **lwarx** with **stwcx.** to emulate common semaphore operations such as test and set, compare and swap, exchange memory, and fetch and add. Both instructions must use the same EA. Reservation granularity is implementation-dependent. The 750GX makes reservations on behalf of aligned 32-byte sections of the memory address space. If the W bit is set, executing **lwarx** and **stwcx.** to a page marked write-through does not cause a DSI exception, but DSI exceptions can result for other reasons. If the location is not word-aligned, an alignment exception occurs.<br>The **stwcx.** instruction is the only load/store instruction with a valid form if Rc is set. If Rc is zero, executing **stwcx.** sets CR0 to an undefined value. In general, **stwcx.** always causes a transaction on the external bus and thus operates with slightly worse performance characteristics than normal store operations. |
| Store Word Conditional Indexed | **stwcx.** | **r**S,**r**A,**r**B | |
| Synchronize | **sync** | — | Because it delays subsequent instructions until all previous instructions complete to where they cannot cause an exception, **sync** is a barrier against store gathering. Additionally, all load/store cache/bus activities initiated by prior instructions are completed. Touch load operations (**dcbt**, **dcbtst**) must complete address translation, but need not complete on the bus. If HID0[ABE] = '1', the **sync** instruction completes after a successful broadcast.<br>The latency of **sync** depends on the processor state when it is dispatched and on various system-level situations. Therefore, frequent use of **sync** might degrade performance. |

System designs with an L2 cache should take special care to recognize the hardware signaling caused by a **sync** bus operation and perform the appropriate actions to guarantee that memory references that can be queued internally to the L2 cache have been performed globally.

See *Section 2.3.5.2, Memory Synchronization Instructions—VEA,* on page 114 for details about additional memory synchronization (**eieio** and **isync**) instructions.

In the PowerPC Architecture, the Rc bit must be zero for most load-and-store instructions. If Rc is set, the instruction form is invalid for **sync** and **lwarx** instructions. If the 750GX encounters one of these invalid instruction forms, it sets CR0 to an undefined value.

### 2.3.5 PowerPC VEA Instructions

The PowerPC virtual environment architecture (VEA) describes the semantics of the memory model that can be assumed by software processes, and includes descriptions of the cache model, cache-control instructions, address aliasing, and other related issues. Implementations that conform to the VEA also adhere to the UISA, but might not necessarily adhere to the OEA.

This section describes additional instructions that are provided by the VEA.

### *2.3.5.1 Processor Control Instructions—VEA*

In addition to the Move-to Condition Register instructions (specified by the UISA), the VEA defines the **mftb** instruction (user-level instruction) for reading the contents of the Time Base Register. See *Chapter 3, Instruction-Cache and Data-Cache Operation,* on page 121 for more information.

*Table 2-36* shows the **mftb** instruction.

*Table 2-36. Move-from Time Base Instruction*

| Name | Mnemonic | Syntax |
|---|---|---|
| Move-from Time Base | **mftb** | **r**D, TBR |

Simplified mnemonics are provided for the **mftb** instruction so it can be coded with the TBR name as part of the mnemonic rather than requiring it to be coded as an operand. See Appendix F, “Simplified Mnemonics” in the *PowerPC Microprocessor Family: The Programming Environments Manual* for simplified mnemonic examples and for simplified mnemonics for Move-from Time Base (**mftb**) and Move-from Time Base Upper (**mftbu**), which are variants of the **mftb** instruction rather than of **mfspr**. The **mftb** instruction serves as both a basic and simplified mnemonic. Assemblers recognize an **mftb** mnemonic with two operands as the basic form, and an **mftb** mnemonic with one operand as the simplified form. Note that the 750GX ignores the extended opcode differences between **mftb** and **mfspr** by ignoring bit 25 and treating both instructions identically.

**Implementation Notes:** The following information relates to using the time-base implementation in the 750GX:

- The 750GX allows user-mode read access to the time-base counter through the use of the Move-from Time Base (**mftb**) and the Move-from Time Base Upper (**mftbu**) instructions. As a 32-bit PowerPC implementation, the 750GX can access TBU and TBL only separately, whereas 64-bit implementations can access the entire TB Register at once.
- The time-base counter is clocked at a frequency that is one-fourth that of the bus clock.

#### *2.3.5.2 Memory Synchronization Instructions—VEA*

Memory synchronization instructions control the order in which memory operations are completed with respect to asynchronous events, and the order in which memory operations are seen by other processors or memory-access mechanisms. See *Chapter 3, Instruction-Cache and Data-Cache Operation,* on page 121 for more information about these instructions and about related aspects of memory synchronization.

In addition to the **sync** instruction (specified by UISA), the VEA defines the Enforce In-Order Execution of I/O (**eieio**) and Instruction Synchronize (**isync**) instructions. The number of cycles required to complete an **eieio** instruction depends on system parameters and on the processor's state when the instruction is issued. As a result, frequent use of this instruction might degrade performance slightly.

*Table 2-37* describes the memory synchronization instructions defined by the VEA.

*Table 2-37. Memory Synchronization Instructions—VEA*

| Name | Mnemonic | Syntax | Implementation Notes |
|---|---|---|---|
| Enforce In-Order Execution of I/O | **eieio** | — | The **eieio** instruction is dispatched to the LSU and executes after all previous cache-inhibited or write-through accesses are performed. All subsequent instructions that generate such accesses execute after **eieio**. If HID0[ABE] = 1, an EIEIO operation is broadcast on the external bus to enforce ordering in the external memory system. The **eieio** operation bypasses the L2 cache and is forwarded to the bus unit. If HID0[ABE] = 0, the operation is not broadcast.<br>Because the 750GX does not reorder noncacheable accesses, **eieio** is not needed to force ordering. However, if store gathering is enabled and an **eieio** is detected in a store queue, stores are not gathered. If HID0[ABE] = 1, broadcasting **eieio** prevents external devices, such as a bus bridge chip, from gathering stores. |
| Instruction Synchronize | **isync** | — | The **isync** instruction is refetch serializing. That is, it causes the 750GX to purge its instruction queue and wait for all prior instructions to complete before refetching the next instruction, which is not executed until all previous instructions complete to the point where they cannot cause an exception. The **isync** instruction does not wait for all pending stores in the store queue to complete. Any instruction after an **isync** sees all effects of prior instructions. |

### *2.3.5.3 Memory Control Instructions—VEA*

Memory control instructions can be classified as follows:

- Cache-management instructions (user-level and supervisor-level)
- Segment Register manipulation instructions (OEA)
- Translation-lookaside-buffer management instructions (OEA)

This section describes the user-level cache-management instructions defined by the VEA. See *Section 2.3.6.3* on page 119 for information about supervisor-level cache, Segment Register manipulation, and translation lookaside buffer management instructions.

#### *User-Level Cache Instructions—VEA*

The instructions summarized in this section help user-level programs manage on-chip caches if they are implemented. See *Chapter 3, Instruction-Cache and Data-Cache Operation,* on page 121 for more information about cache topics. The following sections describe how these operations are treated with respect to the 750GX’s cache.

As with other memory-related instructions, the effects of cache-management instructions on memory are weakly-ordered. If the programmer must ensure that cache or other instructions have been performed with respect to all other processors and system mechanisms, a **sync** instruction must be placed after those instructions.

Note that the 750GX interprets cache-control instructions (**icbi**, **dcbi**, **dcbf**, **dcbz**, and **dcbst**) as if they pertain only to the local L1 and L2 cache. A **dcbz** (with M set) is always broadcast on the 60x bus. The **dcbi**, **dcbf**, and **dcbst** operations are broadcast if HID0[ABE] is set.

The 750GX never broadcasts an **icbi**. Of the broadcast cache operations, the 750GX snoops only **dcbz**, regardless of the HID0[ABE] setting. Any bus activity caused by other cache instructions results directly from performing the operation on the 750GX cache. All cache-control instructions to a direct-store segment (SR[T] = 1 space) are no-ops. For information on how cache-control instructions affect the L2, see *Chapter 9, L2 Cache,* on page 323.

*Table 2-38* summarizes the cache instructions defined by the VEA. Note that these instructions are accessible to user-level programs.

*Table 2-38. User-Level Cache Instructions* (Page 1 of 2)

| Name | Mnemonic | Syntax | Implementation Notes |
|---|---|---|---|
| Data Cache Block Touch[1] | **dcbt** | **r**A,**r**B | The VEA defines this instruction to allow for potential system performance enhancements through the use of software-initiated prefetch hints. Implementations are not required to take any action based on execution of this instruction, but they may prefetch the cache block corresponding to the EA into their cache.<br>When **dcbt** executes, the 750GX checks for protection violations (as for a load instruction). This instruction is treated as a no-op for the following cases:<br>• A valid translation is not found either in BAT or TLB.<br>• The access causes a protection violation.<br>• The page is mapped cache-inhibited, G = 1 (guarded), or T = 1.<br>• The cache is locked or disabled.<br>• HID0[NOOPTI] = 1.<br>Otherwise, if no data is in the cache location, the 750GX requests a cache-line fill (with intent to modify). Data brought into the cache is validated as if it were a load instruction. The memory reference of a **dcbt** sets the reference bit. |
| Data Cache Block Touch for Store[1] | **dcbtst** | **r**A,**r**B | This instruction behaves like **dcbt**. |
| Data Cache Block Set to Zero | **dcbz** | **r**A,**r**B | The EA is computed, translated, and checked for protection violations. For cache hits, four beats of zeros are written to the cache block, and the tag is marked M. For cache misses with the replacement block marked exclusive unmodified (E), the zero line fill is performed, and the cache block is marked M. However, if the replacement block is marked M, the contents are written back to memory first. The instruction executes regardless of whether the cache is locked. If the cache is disabled, an alignment exception occurs. If M = 1 (coherency enforced), the address is broadcast to the bus before the zero line fill.<br>The exception priorities (from highest to lowest) are as follows:<br>1 Cache disabled—Alignment exception<br>2 Page marked write-through or cache Inhibited—Alignment exception<br>3 BAT protection violation—DSI exception<br>4 TLB protection violation—DSI exception<br>**dcbz** is the only cache instruction that broadcasts even if HID0[ABE] = 0. This is done to maintain coherency with other cache devices in the system. |

1. A program that uses **dcbt** and **dcbtst** instructions improperly performs less efficiently. To improve performance, HID0[NOOPTI] can be set, which causes **dcbt** and **dcbtst** to be no-oped at the cache. These instructions do not cause bus activity and cause only a 1-clock execution latency. The default state of this bit is zero, which enables the use of these instructions.

*Table 2-38. User-Level Cache Instructions* (Page 2 of 2)

| Name | Mnemonic | Syntax | Implementation Notes |
|---|---|---|---|
| Data Cache Block Store | **dcbst** | **r**A,**r**B | The EA is computed, translated, and checked for protection violations.<br>• For cache hits with the tag marked exclusive unmodified (E), no further action is taken.<br>• For cache hits with the tag marked M, the cache block is written back to memory and marked exclusive unmodified (E).<br>A **dcbst** is not broadcast unless HID0[ABE] = 1 regardless of WIMG settings. The instruction acts like a load with respect to address translation and memory protection. It executes regardless of whether the cache is disabled or locked.<br>The exception priorities (from highest to lowest) for **dcbst** are as follows:<br>1 BAT protection violation–DSI exception<br>2 TLB protection violation–DSI exception. |
| Data Cache Block Flush | **dcbf** | **r**A,**r**B | The EA is computed, translated, and checked for protection violations.<br>• For cache hits with the tag marked exclusive modified (M), the cache block is written back to memory and the cache entry is invalidated.<br>• For cache hits with the tag marked exclusive unmodified (E), the entry is invalidated.<br>• For cache misses, no further action is taken.<br>A **dcbf** is not broadcast unless HID0[ABE] = 1 regardless of WIMG settings. The instruction acts like a load with respect to address translation and memory protection. It executes regardless of whether the cache is disabled or locked.<br>The exception priorities (from highest to lowest) for **dcbf** are as follows:<br>1 BAT protection violation—DSI exception<br>2 TLB protection violation—DSI exception. |
| Instruction Cache Block Invalidate | **icbi** | **r**A,**r**B | This instruction performs a virtual lookup into the instruction cache (index only). The address is not translated, so it cannot cause an exception. All ways of a selected set are invalidated regardless of whether the cache is disabled or locked. The 750GX never broadcasts **icbi** onto the 60x bus. |

1. A program that uses **dcbt** and **dcbtst** instructions improperly performs less efficiently. To improve performance, HID0[NOOPTI] can be set, which causes **dcbt** and **dcbtst** to be no-oped at the cache. These instructions do not cause bus activity and cause only a 1-clock execution latency. The default state of this bit is zero, which enables the use of these instructions.

### 2.3.5.4 Optional External Control Instructions

The PowerPC Architecture defines an optional external control feature that, if implemented, is supported by the two external control instructions, **eciwx** and **ecowx**. These instructions allow a user-level program to communicate with a special-purpose device. The instructions provided are summarized in *Table 2-39*.

*Table 2-39. External Control Instructions*

| Name | Mnemonic | Syntax | Implementation Notes |
|---|---|---|---|
| External Control In Word Indexed | **eciwx** | **r**D,**r**A,**r**B | A transfer size of 4 bytes is implied. The $\overline{\text{TBST}}$ and TSIZ[0–2] signals are redefined to specify the Resource ID (RID), copied from bits EAR[28–31]. For these operations, $\overline{\text{TBST}}$ carries the EAR[28] data. Misaligned operands for these instructions cause an alignment exception. Addressing a location where SR[T] = 1 causes a DSI exception. If MSR[DR] = 0, a programming error occurs and the physical address on the bus is undefined.<br>**Note**: These instructions are optional in the PowerPC Architecture. |
| External Control Out Word Indexed | **ecowx** | **r**S,**r**A,**r**B | |

The **eciwx** and **ecowx** instructions let a system designer map special devices in an alternative way. The MMU translation of the EA is not used to select the special device, as it is used in most instructions such as loads and stores. Rather, it is used as an address operand that is passed to the device over the address bus. Four other signals (the burst and size signals on the 60x bus) are used to select the device; these four signals output the 4-bit resource ID (RID) field located in the EAR. The **eciwx** instruction also loads a word from the data bus that is output by the special device. For more information about the relationship between these instructions and the system interface, see *Chapter 7, Signal Descriptions,* on page 249.

### 2.3.6 PowerPC OEA Instructions

The PowerPC operating environment architecture (OEA) includes the structure of the memory-management model, supervisor-level registers, and the exception model. Implementations that conform to the OEA also adhere to the UISA and the VEA. This section describes the instructions provided by the OEA.

#### *2.3.6.1 System Linkage Instructions—OEA*

This section describes the system linkage instructions (see *Table 2-40*). The user-level **sc** instruction lets a user program call on the system to perform a service and causes the processor to take a system-call exception. The supervisor-level **rfi** instruction is used for returning from an exception handler.

*Table 2-40. System Linkage Instructions—OEA*

| Name | Mnemonic | Syntax | Implementation Notes |
|---|---|---|---|
| System Call | **sc** | — | The **sc** instruction is context-synchronizing. |
| Return from Interrupt | **rfi** | — | The **rfi** instruction is privileged and context-synchronizing. For the 750GX, this means the **rfi** instruction works its way to the final stage of the execution pipeline, updates architected registers, and redirects the instruction flow. |

#### *2.3.6.2 Processor Control Instructions—OEA*

This section describes the processor control instructions used to access the MSR and the SPRs. *Table 2-41* lists instructions for accessing the MSR.

*Table 2-41. Move-to/Move-from Machine State Register Instructions*

| Name | Mnemonic | Syntax |
|---|---|---|
| Move-to Machine State Register | **mtmsr** | **r**S |
| Move-from Machine State Register | **mfmsr** | **r**D |

The OEA defines encodings of **mtspr** and **mfspr** to provide access to supervisor-level registers. The instructions are listed in *Table 2-42*.

*Table 2-42. Move-to/Move-from Special-Purpose Register Instructions (OEA)*

| Name | Mnemonic | Syntax |
|---|---|---|
| Move-to Special-Purpose Register | **mtspr** | SPR,**r**S |
| Move-from Special-Purpose Register | **mfspr** | **r**D,SPR |

Encodings for the architecture-defined SPRs are listed in *Table 2-33* on page 109. Encodings for 750GX-specific, supervisor-level SPRs are listed in *Table 2-34* on page 112. Simplified mnemonics are provided for **mtspr** and **mfspr** in Appendix F, “Simplified Mnemonics” in the *PowerPC Microprocessor Family: The Programming Environments Manual*. For a discussion of context synchronization requirements when altering certain SPRs, see Appendix E, “Synchronization Programming Examples” in the *PowerPC Microprocessor Family: The Programming Environments Manual*.

### *2.3.6.3 Memory Control Instructions—OEA*

Memory control instructions include the following.

- Cache-management instructions (supervisor-level and user-level).
- Segment register manipulation instructions.
- Translation-lookaside-buffer management instructions.

This section describes supervisor-level memory control instructions. *Section 2.3.5.3, Memory Control Instructions—VEA,* on page 115 describes user-level memory control instructions.

#### *Supervisor-Level Cache-Management Instruction—(OEA)*

*Table 2-43* lists the only supervisor-level cache-management instruction.

*Table 2-43. Supervisor-Level Cache-Management Instruction*

| Name | Mnemonic | Syntax | Implementation Notes |
|---|---|---|---|
| Data Cache Block Invalidate | **dcbi** | **r**A,**r**B | The EA is computed, translated, and checked for protection violations. For cache hits, the cache block is marked invalid (I) regardless of whether it was marked exclusive unmodified (E) or exclusive modified (M). A **dcbi** is not broadcast unless HID0[ABE] = 1, regardless of WIMG settings. The instruction acts like a store with respect to address translation and memory protection. It executes regardless of whether the cache is disabled or locked.<br>The exception priorities (from highest to lowest) for **dcbi** are as follows:<br>1 BAT protection violation—DSI exception.<br>2 TLB protection violation—DSI exception. |

See *User-Level Cache Instructions—VEA* on page 115 for cache instructions that provide user-level programs the ability to manage the on-chip caches. If the effective address references a direct-store segment, then the instruction is treated as a no-op.

#### *Segment Register Manipulation Instructions (OEA)*

The instructions listed in *Table 2-44* provide access to the Segment Registers for 32-bit implementations. These instructions operate completely independently of the MSR[IR] and MSR[DR] bit settings. See "Synchronization Requirements for Special Registers and for Lookaside Buffers" in Chapter 2, "PowerPC Register Set" of the *PowerPC Microprocessor Family: The Programming Environments Manual* for serialization requirements and other recommended precautions to observe when manipulating the Segment Registers. Be sure to execute an **isync** after execution of an **mtsr** instruction.

*Table 2-44. Segment Register Manipulation Instructions*

| Name | Mnemonic | Syntax | Implementation Notes |
|---|---|---|---|
| Move-to Segment Register | **mtsr** | SR,**r**S | Execute **isync** after **mtsr.** |
| Move-to Segment Register Indirect | **mtsrin** | **r**S,**r**B | Execute **isync** after **mtsrin.** |
| Move-from Segment Register | **mfsr** | **r**D,SR | The shadow SRs in the instruction MMU can be read by setting HID0[RISEG] before executing **mfsr**. |
| Move-from Segment Register Indirect | **mfsrin** | **r**D,**r**B | — |

*Translation Lookaside Buffer Management Instructions—(OEA)*

The address-translation mechanism is defined in terms of the segment descriptors and page table entries (PTEs) PowerPC processors use to locate the logical-to-physical address mapping for a particular access. These segment descriptors and PTEs reside in Segment Registers and page tables in memory, respectively.

See *Chapter 7, Signal Descriptions,* on page 249 for more information about TLB operations. *Table 2-45* summarizes the operation of the TLB instructions in the 750GX.

*Table 2-45. Translation Lookaside Buffer Management Instruction*

| Name | Mnemonic | Syntax | Implementation Notes |
|---|---|---|---|
| TLB Invalidate Entry | **tlbie** | **r**B | Invalidates both ways in both instruction and data TLB entries at the index provided by EA[14–19]. It executes regardless of the MSR[DR] and MSR[IR] settings. To invalidate all entries in both TLBs, the programmer should issue 64 **tlbie** instructions; each should successively increment this field. |
| TLB Synchronize | **tlbsync** | — | On the 750GX, the only function **tlbsync** serves is to wait for the TLB Invalidate Synchronize (TLBISYNC) signal to go inactive. |

**Implementation Note:** The **tlbia** instruction is optional for an implementation if its effects can be achieved through some other mechanism. Therefore, it is not implemented on the 750GX. As described above, **tlbie** can be used to invalidate a particular index of the TLB based on EA[14–19]—a sequence of 64 **tlbie** instructions followed by a **tlbsync** instruction invalidates all the TLB structures (for EA[14–19] = 0, 1, 2,..., 63). Attempting to execute **tlbia** causes an illegal instruction program exception.

The presence and exact semantics of the TLB management instructions are implementation-dependent. To minimize compatibility problems, system software should incorporate uses of these instructions into subroutines.

### 2.3.7 Recommended Simplified Mnemonics

To simplify assembly language coding, a set of alternative mnemonics is provided for some frequently used operations (such as no-op, load immediate, load address, move register, and complement register). Programs written to be portable across the various assemblers for the PowerPC Architecture should not assume the existence of mnemonics not described in this document.

For a complete list of simplified mnemonics, see Appendix F, “Simplified Mnemonics” in the *PowerPC Microprocessor Family: The Programming Environments Manual*.

# 3. Instruction-Cache and Data-Cache Operation

The 750GX microprocessor contains separate 32-KB, 8-way set-associative instruction and data caches to allow the execution units and registers rapid access to instructions and data. This chapter describes the organization of the on-chip instruction and data caches, the modified, exclusive, invalid (MEI) cache-coherency protocol, cache-control instructions, various cache operations, and the interaction between the caches, the load/store unit (LSU), the instruction unit, and the bus interface unit (BIU).

Note that in this chapter, the term ‘multiprocessor’ is used in the context of maintaining cache coherency. These multiprocessor devices could be actual processors or other devices that can access system memory, maintain their own caches, and function as bus masters requiring cache coherency. If the L2 cache is enabled, read *Chapter 9, L2 Cache,* on page 323 before reading this chapter.

The 750GX L1 cache implementation has the following characteristics.

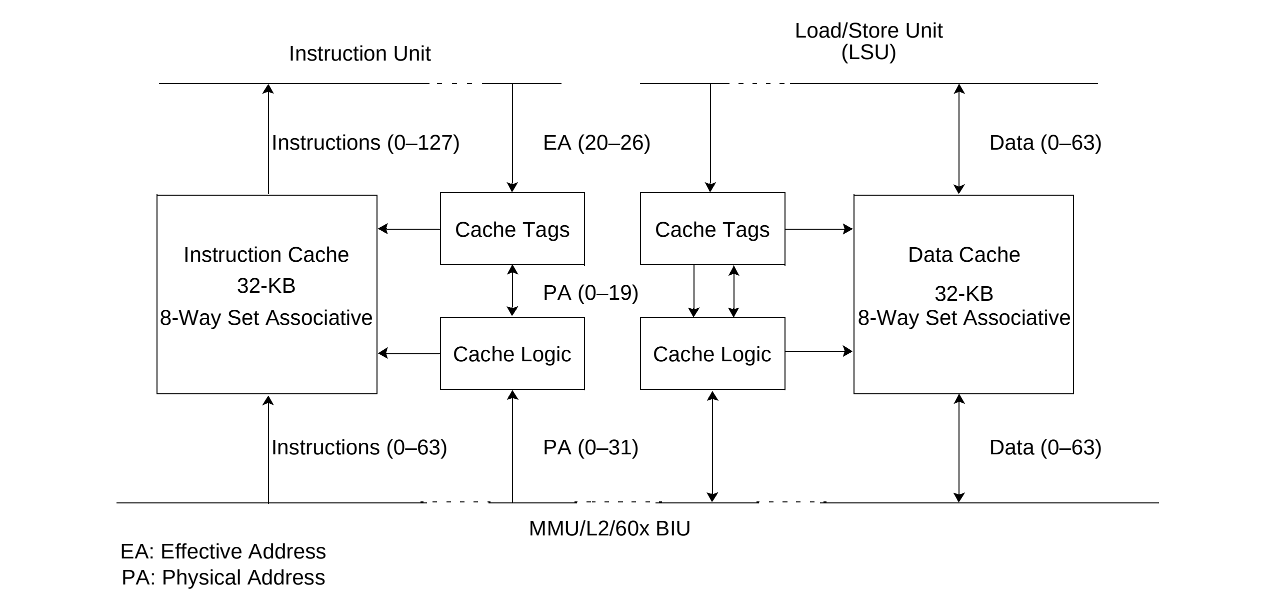
- There are two separate 32-KB instruction and data caches (Harvard architecture).
- Both instruction and data caches are 8-way set-associative.
- The caches implement a pseudo least-recently-used (PLRU) replacement algorithm within each set.
- The cache directories are physically addressed. The physical (real) address tag is stored in the cache directory.
- Both the instruction and data caches have 32-byte cache blocks. A cache block is the block of memory that a coherency state describes, also referred to as a cache line.
- Two coherency state bits for each data-cache block allow encoding for three states:
  - Exclusive Modified (M)
  - Exclusive Unmodified (E)
  - Invalid (I)
- A single coherency state bit for each instruction-cache block allows encoding for two possible states:
  - Invalid (INV)
  - Valid (VAL)
- Each cache can be invalidated or locked by setting the appropriate bits in the Hardware-Implementation-Dependent Register 0 (HID0), a Special-Purpose Register (SPR) specific to the 750GX.

The 750GX supports a fully-coherent 4-GB physical memory address space. Bus snooping is used to drive the MEI 3-state cache-coherency protocol that ensures the coherency of global memory with respect to the processor’s data cache. The MEI protocol is described in *Section 3.3.2* on page 126.

On a cache miss, the 750GX’s cache blocks are filled in four beats of 64 bits each. The burst fill is performed as a critical-double-word-first operation. The critical double word is simultaneously written to the cache and forwarded to the requesting unit, thus minimizing stalls due to cache fill latency. The data-cache line is first loaded into a 32-byte reload buffer, and, when it is full, it is written into the data cache in one cycle. This minimizes the contention between the load-store unit and the line reload function. See *Figure 9-1* on page 327.

The instruction and data caches are integrated into the 750GX as shown in *Figure 3-1*.

*Figure 3-1. Cache Integration*



Both caches are tightly coupled into the 750GX’s bus interface unit (BIU) to allow efficient access to the system memory controller and other bus masters. The bus interface unit receives requests for bus operations from the instruction and data caches, and executes the operations per the 60x bus protocol. The BIU provides address queues, prioritizing logic, and bus control logic. The BIU captures snoop addresses for data cache, address queue, and memory reservation (**lwarx** and **stwcx.** instruction) operations. In the 750GX, an L1 cache miss first accesses the L2 cache to find the desired cache block before accessing the BIU.

The data cache provides buffers for load-and-store bus operations. All the data for the corresponding address queues (load-and-store data queues) is located in the data cache. The data queues are considered temporary storage for the cache and not part of the BIU. The data cache also provides storage for the cache tags required for memory coherency and performs the cache-block-replacement PLRU function. The data cache is supported by two cache-block reload/write-back buffers. This allows a cache block to be loaded or unloaded from the cache in a single cycle. See *Figure 9-1* on page 327.

The data cache supplies data to the General Purpose Registers (GPRs) and Floating Point Registers (FPRs) by means of the load/store unit (LSU). The 750GX’s LSU is directly coupled to the data cache to allow efficient movement of data to and from the GPRs and FPRs. The LSU provides all logic required to calculate effective addresses, handles data alignment to and from the data cache, and provides sequencing for load-and-store string and multiple operations. Write operations to the data cache can be performed on a byte, half-word, word, or double-word basis.

The instruction cache provides a 128-bit interface to the instruction unit, so four instructions can be made available to the instruction unit in a single clock cycle. The instruction unit accesses the instruction cache frequently in order to sustain the high throughput provided by the 6-entry instruction queue.

## 3.1 Data-Cache Organization

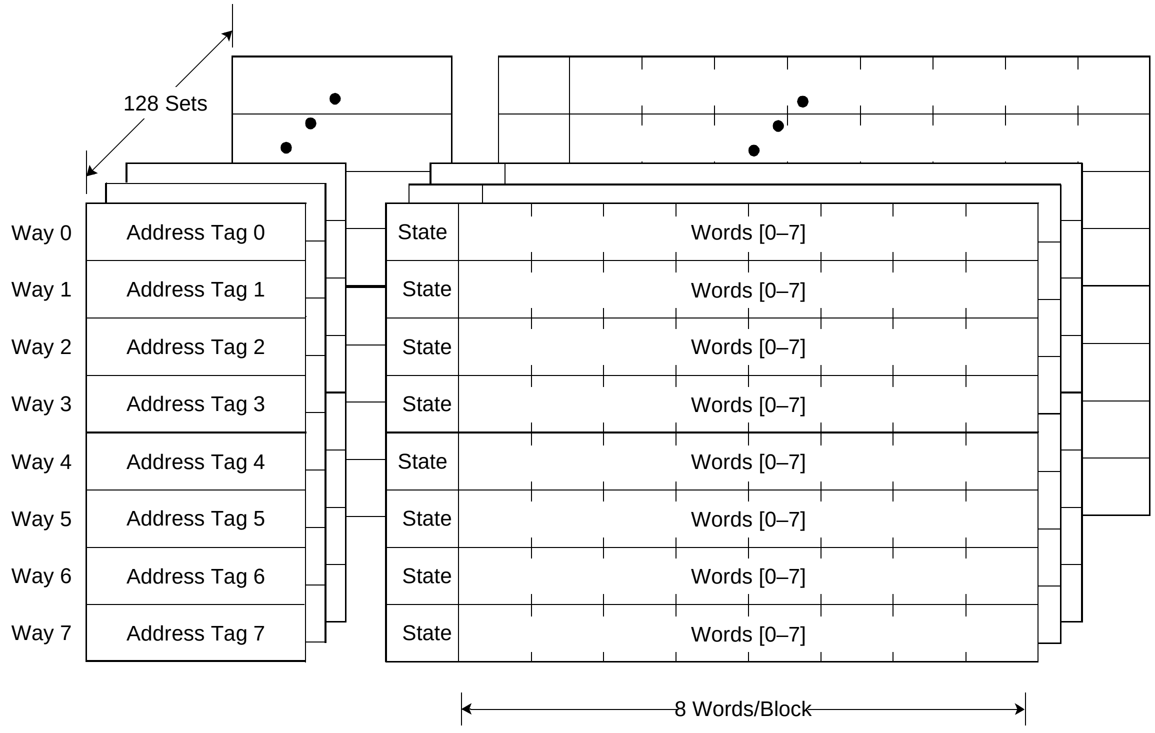
The data cache is organized as 128 sets of eight ways as shown in *Figure 3-2*. Each way consists of 32 bytes, two state bits, and an address tag. Note that in the PowerPC Architecture, the term 'cache block,' or simply 'block,' when used in the context of cache implementations, refers to the unit of memory at which coherency is maintained. For the 750GX, this is the 8-word (32-byte) cache line. This value might be different for other PowerPC implementations.

Each cache block contains eight contiguous words from memory that are loaded from an 8-word boundary (that is, bits A[27–31] of the logical (effective) addresses are zero). As a result, cache blocks are aligned with page boundaries. Note that address bits A[20–26] provide the index to select a cache set. Bits A[27–31] select a byte within a block. The two state bits implement a 3-state MEI protocol, a coherent subset of the standard 4-state modified/exclusive/shared/invalid (MESI) protocol. The MEI protocol is described in *Section 3.3.2* on page 126.

The tags consist of physical address bits PA[0–19]. Address translation occurs in parallel with set selection (from A[20–26]), and the higher-order address bits (the tag bits in the cache) are physical.

The 750GX's on-chip data-cache tags are single-ported, and load or store operations must be arbitrated with snoop accesses to the data-cache tags. Load or store operations can be performed to the cache on the clock cycle immediately following a snoop access if the snoop misses; snoop hits might block the data cache for two or more cycles, depending on whether a copy-back to main memory is required.

*Figure 3-2. Data-Cache Organization*



## 3.2 Instruction-Cache Organization

The instruction cache also consists of 128 sets of eight ways, as shown in *Figure 3-3* on page 125. Each way consists of 32 bytes, a single state bit, and an address tag. As with the data cache, each instruction-cache block contains eight contiguous words from memory that are loaded from an 8-word boundary (that is, bits A[27–31] of the logical [effective] addresses are zero). As a result, cache blocks are aligned with page boundaries. Also, address bits A[20–26] provide the index to select a set, and bits A[27–29] select a word within a block.

The tags consist of bits PA[0–19]. Address translation occurs in parallel with set selection (from A[20–26]), and the higher order address bits (the tag bits in the cache) are physical.

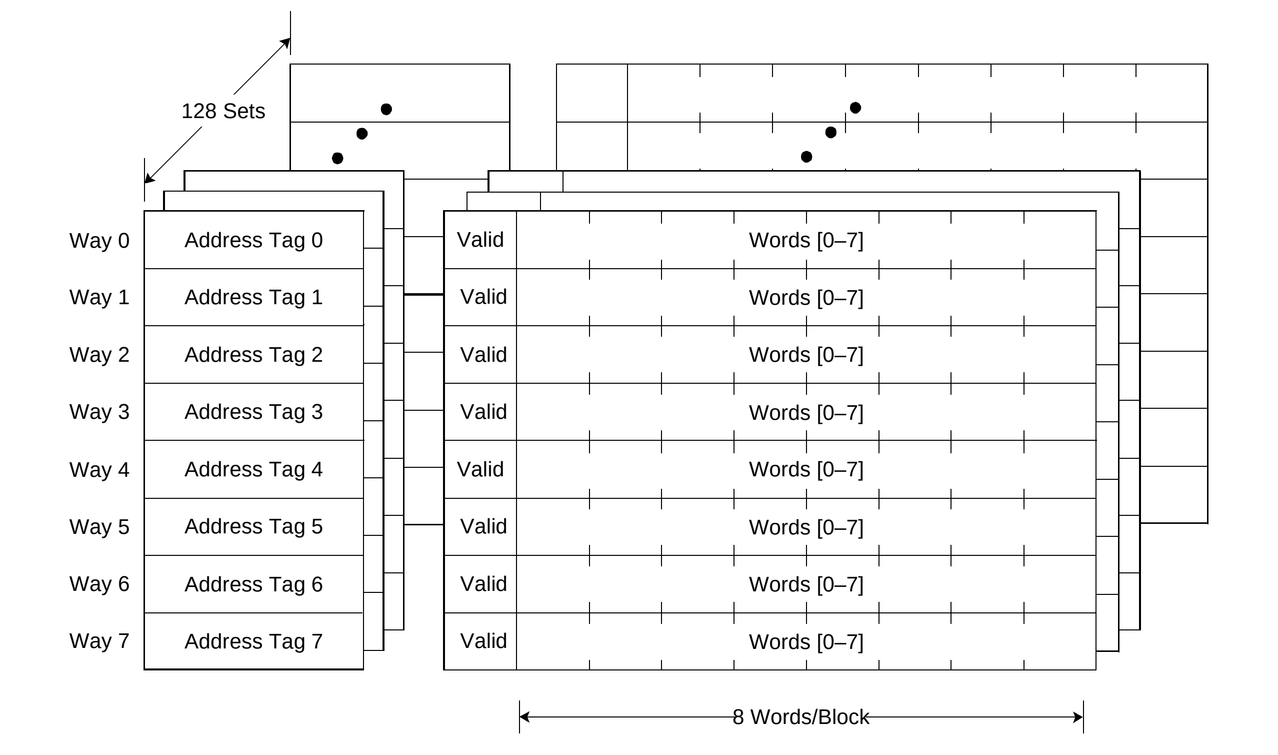
The instruction cache differs from the data cache in that it does not implement MEI cache-coherency protocol, and a single state bit is implemented that indicates only whether a cache block is valid or invalid. The instruction cache is not snooped, so if a processor modifies a memory location that might be contained in the instruction cache, software must ensure that such memory updates are visible to the instruction-fetching mechanism. This can be achieved with the following instruction sequence:

```
dcbst    # update memory

sync     # wait for update

icbi     # remove (invalidate) copy in instruction cache

sync     # wait for the Instruction Cache Block Invalidate (ICBI) operation to be globally performed

isync    # remove copy in own instruction buffer
```

These operations are necessary because the processor does not maintain instruction memory coherent with data memory. Software is responsible for enforcing coherency of instruction caches and data memory.

Since instruction fetching might bypass the data cache, changes made to items in the data cache might not be reflected in memory until after the instruction fetch completes.

*Figure 3-3. Instruction-Cache Organization*



## 3.3 Memory and Cache Coherency

The primary objective of a coherent memory system is to provide the same image of memory to all devices using the system. Coherency allows synchronization and cooperative use of shared resources. Otherwise, multiple copies of a memory location, some containing stale values, could exist in a system resulting in errors when the stale values are used. Each potential bus master must follow rules for managing the state of its cache. This section describes the coherency mechanisms of the PowerPC Architecture and the 3-state cache-coherency protocol of the 750GX's data cache.

Note that unless specifically noted, the discussion of coherency in this section applies to the 750GX's data cache only. The instruction cache is not snooped. Instruction-cache coherency must be maintained by software. However, the 750GX does support a fast instruction-cache invalidate capability as described in *Section 3.4.1.4* on page 133.

### 3.3.1 Memory/Cache Access Attributes (WIMG Bits)

Some memory characteristics can be set on either a block or page basis by using the WIMG bits in the block-address-translation (BAT) registers or page table entry (PTE), respectively. The WIMG attributes control the following functionality:

- Write-through (W bit)
- Caching-inhibited (I bit)
- Memory coherency (M bit)
- Guarded memory (G bit)

These bits allow both uniprocessor and multiprocessor system designs to exploit numerous system-level performance optimizations.

The WIMG attributes are programmed by the operating system for each page and block. The write-through (W) and caching-inhibited (I) attributes control how the processor performing an access uses its own cache. The memory coherency (M) attribute ensures that coherency is maintained for all copies of the addressed memory location. The guarded memory (G) attribute prevents out-of-order loading and prefetching from the addressed memory location.

The WIMG attributes occupy four bits in the BAT registers for block-address translation and in the PTEs for page-address translation. The WIMG bits are programmed as follows.

- The operating system uses the Move-to Special Purpose Register (**mtspr)** instruction to program the WIMG bits in the BAT registers for block-address translation. The instruction BAT (IBAT) register pairs do not have a G bit, and all accesses that use the IBAT register pairs are considered not guarded.
- The operating system writes the WIMG bits for each page into the PTEs in system memory as it sets up the page tables.

When an access requires coherency, the processor performing the access must inform the coherency mechanisms throughout the system that the access requires memory coherency. The M attribute determines the kind of access performed on the bus (global or local).

Software must exercise care with respect to the use of these bits if coherent memory support is desired. Careless specification of these bits might create situations that present coherency paradoxes to the processor. In particular, this can happen when the state of these bits is changed without appropriate precautions (such as flushing the pages that correspond to the changed bits from the caches of all processors in the system) or when the address translations of aliased real addresses specify different values for any of the WIMG bits. These coherency paradoxes can occur within a single processor or across several processors. It is important to note that in the presence of a paradox, the operating system software is responsible for correctness.

For data accesses in real-addressing mode (MSR[DR] = 0), the WIMG bits default to '0011' (write back, caching enabled, memory coherent, guarded). For instruction accesses in real-addressing mode (MSR[IR] = 0), the WIMG bits default to '0001' (write back, caching enabled, memory not coherent, guarded).

### 3.3.2 MEI Protocol

The 750GX data-cache-coherency protocol is a coherent subset of the standard MESI 4-state cache protocol that omits the shared state. The 750GX’s data cache characterizes each 32-byte block it contains as being in one of three MEI states. Addresses presented to the cache are indexed into the cache directory with bits A[20–26], and the upper-order 20 bits from the physical address translation (PA[0–19]) are compared against the indexed cache directory tags. If neither of the indexed tags matches, the result is a cache miss. If a tag matches, a cache hit occurred, and the directory indicates the state of the cache block through two state bits kept with the tag. The three possible states for a cache block in the cache are the modified state (M), the exclusive state (E), and the invalid state (I). The three MEI states are defined in *Table 3-1* on page 127.

*Table 3-1. MEI State Definitions*

| MEI State | Definition |
|---|---|
| Modified (M) | The addressed cache block is present in the cache, and is modified with respect to system memory. That is, the modified data in the cache block has not been written back to memory. The cache block might be present in 750GX's L2 cache, but it is not present in any other coherent cache. |
| Exclusive (E) | The addressed cache block is present in the cache, and this cache has exclusive ownership of the addressed block. The addressed block might be present in 750GX's L2 cache, but it is not present in any other processor's cache. The data in this cache block is consistent with system memory. |
| Invalid (I) | This state indicates that the address block does not contain valid data, or that the addressed cache block is not resident in the cache. |

The 750GX provides dedicated hardware to provide memory coherency by snooping bus transactions. *Figure 3-4* on page 128 shows the MEI cache-coherency protocol, as enforced by the 750GX. The information in this figure assumes that the WIM bits for the page or block are set to 001; that is, write-back, caching-not-inhibited, and memory coherency enforced.

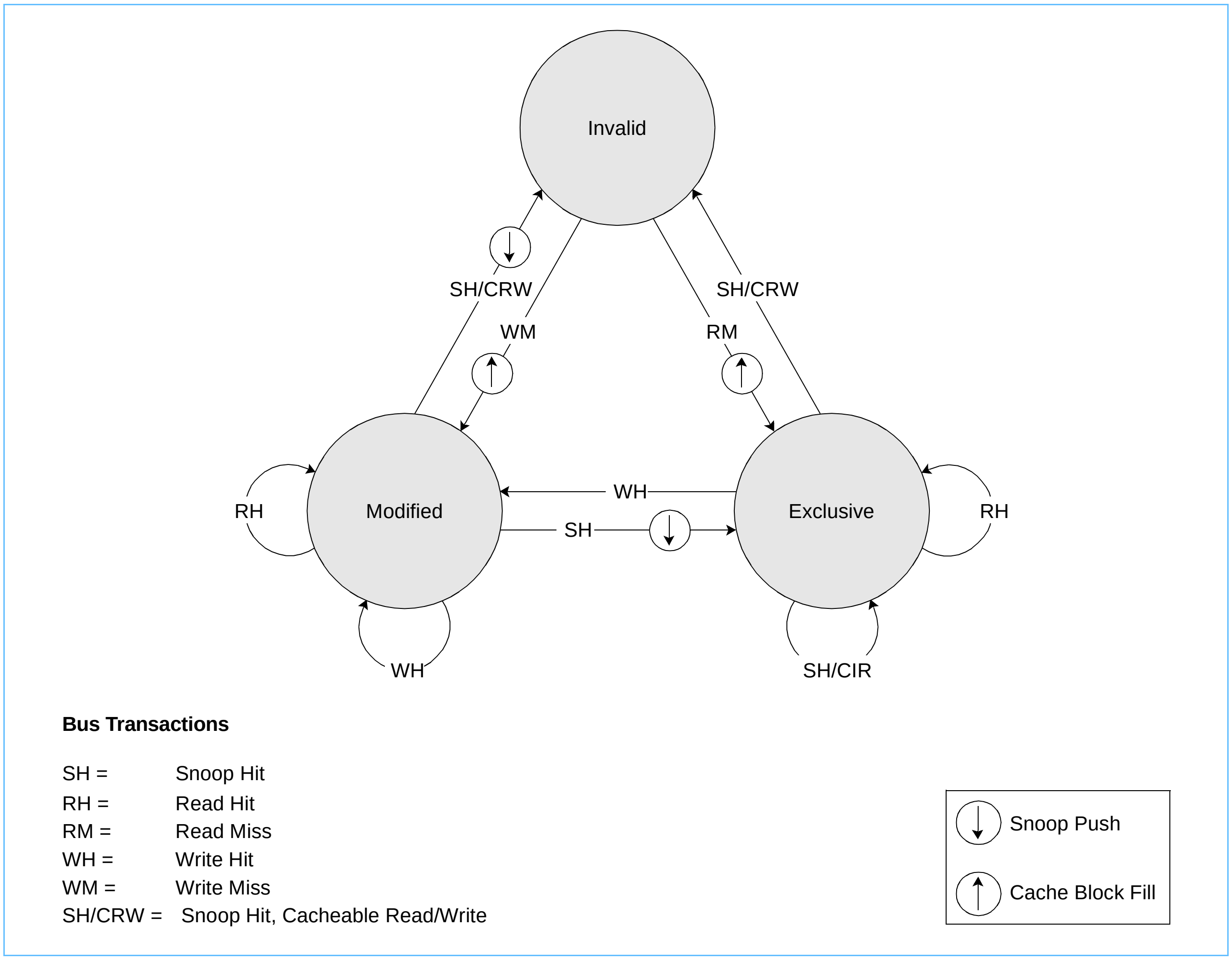
Since data cannot be shared, the 750GX signals all cache block fills as if they were write misses (read-with-intent-to-modify), which flushes the corresponding copies of the data in all caches external to the 750GX prior to the cache-block-fill operation. Following the cache-block load, the 750GX is the exclusive owner of the data and can write to it without a bus broadcast transaction.

To maintain the 3-state coherency, all global reads observed on the bus by the 750GX are snooped as if they were writes, causing the 750GX to flush the cache block (write the cache block back to memory and invalidate the cache block if it is modified, or simply invalidate the cache block if it is unmodified). The exception to this rule occurs when a snooped transaction is a caching-inhibited read[1], in which case the 750GX does not invalidate the snooped cache block. If the cache block is modified, the block is written back to memory, and the cache block is marked exclusive. If the cache block is marked exclusive, no bus action is taken, and the cache block remains in the exclusive state.

This treatment of caching-inhibited reads decreases the possibility of data thrashing by allowing noncaching devices to read data without invalidating the entry from the 750GX's data cache.

[1] Either burst or single-beat, where the transfer type (TT[0–4]) = X1010. See *Table 7-1* on page 256 for clarification.

*Figure 3-4. MEI Cache-Coherency Protocol—State Diagram (WIM = 001)*



*Section 3.7, MEI State Transactions,* on page 147 provides a detailed list of MEI transitions for various operations and WIM bit settings.

### *3.3.2.1 MEI Hardware Considerations*

While the 750GX provides the hardware required to monitor bus traffic for coherency, the 750GX’s datacache tags are single-ported, and a simultaneous load/store and snoop access represent a resource conflict. In general, the snoop access has the highest priority and is given first access to the tags. The load or store access will then occur on the clock following the snoop. The snoop is not given priority into the tags when the snoop coincides with a tag write (for example, validation after a cache-block load). In these situations, the snoop is retried and must rearbitrate before the lookup is possible.

Occasionally, cache snoops cannot be serviced and must be retried. These retries occur if the cache is busy with a burst read or write when the snoop operation takes place.

Note that it is possible for a snoop to hit a modified cache block that is already in the process of being written to the copy-back buffer for replacement purposes. If this happens, the 750GX retries the snoop, and raises the priority of the castout operation to allow it to go to the bus before the cache-block fill.

Another consideration is page table aliasing. If a store hits to a modified cache block but the page table entry is marked write-through (WIMG = 1xxx), then the page has probably been aliased through another page table entry which is marked write-back (WIMG = 0xxx). If this occurs, the 750GX ignores the modified bit in the cache tag. The cache block is updated during the write-through operation, and the block remains in the modified state.

The global ($\overline{\text{GBL}}$) signal, asserted as part of the address attribute field during a bus transaction, enables the snooping hardware of the 750GX. Address-bus masters assert $\overline{\text{GBL}}$ to indicate that the current transaction is a global access (that is, an access to memory shared by more than one device). If $\overline{\text{GBL}}$ is not asserted for the transaction, that transaction is not snooped by the 750GX. Note that the $\overline{\text{GBL}}$ signal is not asserted for instruction fetches, and that $\overline{\text{GBL}}$ is asserted for all data read or write operations when using real-addressing mode (that is, address translation is disabled).

Normally, $\overline{\text{GBL}}$ reflects the M-bit value specified for the memory reference in the corresponding translation descriptors. Care should be taken to minimize the number of pages marked as global, because the retry protocol enforces coherency and can use considerable bus bandwidth if much data is shared. Therefore, available bus bandwidth decreases as more memory is marked as global.

The 750GX snoops a transaction if the transfer start ($\overline{\text{TS}}$) and $\overline{\text{GBL}}$ signals are asserted together in the same bus clock (this is a qualified snooping condition). No snoop update to the 750GX cache occurs if the snooped transaction is not marked global. Also, because cache-block castouts and snoop pushes do not require snooping, the $\overline{\text{GBL}}$ signal is not asserted for these operations.

When the 750GX detects a qualified snoop condition, the address associated with the $\overline{\text{TS}}$ signal is compared with the cache tags. Snooping finishes if no hit is detected. If, however, the address hits in the cache, the 750GX reacts according to the MEI protocol shown in *Figure 3-4* on page 128.

### 3.3.3 Coherency Precautions in Single-Processor Systems

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 750GX ignores any hits to a cache block in a memory space marked caching-inhibited (WIMG = x1xx). The access is performed on the external bus as if there were no hit. The data in the cache is not pushed, and the cache block is not invalidated.
- Store to a page marked write-through (WIMG = 1xxx) and a cache hit occurs to a modified cache block.
  The 750GX ignores 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.

### 3.3.4 Coherency Precautions in Multiprocessor Systems

The 750GX’s 3-state coherency protocol permits no data sharing between the 750GX and other caches. All burst reads initiated by the 750GX are performed as read with intent to modify. Burst snoops are interpreted as read with intent to modify or read with no intent to cache. This effectively places all caches in the system into a 3-state coherency scheme. The 4-state caches can share data amongst themselves but not with the 750GX.

### 3.3.5 PowerPC 750GX-Initiated Load/Store Operations

Load-and-store operations are assumed to be weakly ordered on the 750GX. The load/store unit (LSU) can perform load operations that occur later in the program ahead of store operations, even when the data cache is disabled (see *Section 3.3.5.2*). However, strongly ordered load-and-store operations can be enforced by setting the invalid (I) bit (of the page WIMG bits) when address translation is enabled. Note that when address translation is disabled (real-addressing mode), the default WIMG bits cause the I bit to be cleared (accesses are assumed to be cacheable), and thus the accesses are weakly ordered. See *Section 5.2* on page 195 for a description of the WIMG bits when address translation is disabled.

The 750GX does not provide support for direct-store segments. Operations attempting to access a direct-store segment will invoke a data-storage interrupt (DSI) exception. For additional information about DSI exceptions, see *Section 4.5.3, DSI Exception (0x00300),* on page 169.

#### *3.3.5.1 Performed Loads and Stores*

The PowerPC Architecture defines a performed load operation as one that has the addressed memory location bound to the target register of the load instruction. The architecture defines a performed store operation as one where the stored value is the value that any other processor will receive when executing a load operation (that is of course, until it is changed again). With respect to the 750GX, caching-enabled (WIMG = x0xx) loads and caching-enabled, write-back (WIMG = 00xx) stores are performed when they have arbitrated to address the cache block. Note that in the event of a cache miss, these storage operations might place a memory request into the processor’s memory queue, but such operations are considered an extension to the state of the cache with respect to snooping bus operations. Caching-inhibited (WIMG = x1xx) loads, caching-inhibited (WIMG = x1xx) stores, and write-through (WIMG = 1xxx) stores are performed when they have been successfully presented to the external 60x bus.

#### *3.3.5.2 Sequential Consistency of Memory Accesses*

The PowerPC Architecture requires that all memory operations executed by a single processor be sequentially consistent with respect to that processor. This means that all memory accesses appear to be executed in program order with respect to exceptions and data dependencies.

The 750GX achieves sequential consistency by operating a single pipeline to the cache/MMU. All memory accesses are presented to the MMU in exact program order. Therefore, exceptions are determined in order. Loads are allowed to bypass stores once exception checking has been performed for the store, but data dependency checking is handled in the load/store unit so that a load will not bypass a store with an address match. Note that, although memory accesses that miss in the cache are forwarded to the memory queue for future arbitration for the external bus, all potential synchronous exceptions have been resolved before the cache. In addition, although subsequent memory accesses can address the cache, full coherency checking between the cache and the memory queue is provided to avoid dependency conflicts.

#### *3.3.5.3 Atomic Memory References*

The PowerPC Architecture defines the Load Word and Reserve Indexed (**lwarx**) and the Store Word Conditional Indexed (**stwcx.**) instructions to provide an atomic update function for a single, aligned word of memory. These instructions can be used to develop a rich set of multiprocessor synchronization primitives.

**Note:** Atomic memory references constructed using **lwarx** and **stwcx.** instructions depend on the presence of a coherent memory system for correct operation. These instructions should not be expected to provide atomic access to noncoherent memory. For detailed information on these instructions, see *Chapter 2, Programming Model,* on page 57.

The **lwarx** instruction performs a load word from memory operation and creates a reservation for the 32-byte section of memory that contains the accessed word. The reservation granularity is 32 bytes. The **lwarx** instruction makes a nonspecific reservation with respect to the executing processor and a specific reservation with respect to other masters. This means that any subsequent **stwcx.** executed by the same processor, regardless of address, will cancel the reservation. Also, any bus write or invalidate operation from another processor to an address that matches the reservation address will cancel the reservation.

The **stwcx.** instruction does not check the reservation for a matching address. The **stwcx.** instruction is only required to determine whether a reservation exists. The **stwcx.** instruction performs a store word operation only if the reservation exists. If the reservation has been cancelled for any reason, then the **stwcx.** instruction fails and clears the CR0[EQ] bit in the Condition Register. The architectural intent is to follow the **lwarx** and **stwcx.** instruction pair with a conditional branch, which checks to see whether the **stwcx.** instruction failed.

If the page table entry is marked caching-enabled (WIMG = x0xx), and an **lwarx** access misses in the cache, then the 750GX performs a cache-block fill. If the page is marked caching-inhibited (WIMG = x1xx) or the cache is locked, and the access misses, then the **lwarx** instruction appears on the bus as a single-beat load. All bus operations that are a direct result of either an **lwarx** instruction or an **stwcx.** instruction are placed on the bus with a special encoding. Note that this does not force all **lwarx** instructions to generate bus transactions, but rather provides a means for identifying when an **lwarx** instruction does generate a bus transaction. If an implementation requires that all **lwarx** instructions generate bus transactions, then the associated pages should be marked as caching-inhibited.

The 750GX’s data cache treats all **stwcx.** operations as write-through independent of the WIMG settings. However, if the **stwcx.** operation hits in the 750GX’s L2 cache, then the operation completes with the reservation intact in the L2 cache. See *Chapter 9, L2 Cache,* on page 323 for more information. Otherwise, the **stwcx.** operation continues to the bus interface unit for completion. When the write-through operation completes successfully, either in the L2 cache or on the 60x bus, then the data-cache entry is updated (assuming it hits), and CR0[EQ] is modified to reflect the success of the operation. If the reservation is not intact, the **stwcx.** completes in the bus interface unit without performing a bus transaction, and without modifying either of the caches.

## 3.4 Cache Control

The 750GX’s L1 caches are controlled by programming specific bits in the HID0 Special-Purpose Register and by issuing dedicated cache-control instructions. *Section 3.4.1* describes the HID0 cache-control bits, and *Section 3.4.2* on page 133 describes the cache-control instructions.

### 3.4.1 Cache-Control Parameters in HID0

The HID0 Special-Purpose Register contains several bits that invalidate, disable, and lock the instruction and data caches. The following sections describe these facilities.

### *3.4.1.1 Data-Cache Flash Invalidation*

The data cache is automatically invalidated when the 750GX is powered up and during a hard reset. However, a soft reset does not automatically invalidate the data cache. Software must use the HID0 data-cache flash invalidate bit (HID0[DCFI]) if data cache invalidation is desired after a soft reset. Once HID0[DCFI] is set through an **mtspr** operation, the 750GX automatically clears this bit in the next clock cycle (provided that the data cache is enabled in the HID0 Register).

Note that some PowerPC microprocessors accomplish data-cache flash invalidation by setting and clearing HID0[DCFI] with two consecutive **mtspr** instructions (that is, the bit is not automatically cleared by the microprocessor). Software that has this sequence of operations does not need to be changed to run on the 750GX.

### *3.4.1.2 Enabling and Disabling the Data Cache*

The data cache can be enabled or disabled by using the data-cache enable bit, HID0[DCE]. HID0[DCE] is cleared on power-up, disabling the data cache.

When the data cache is in the disabled state (HID0[DCE] = 0), the cache tag state bits are ignored, and all accesses are propagated to the L2 cache or 60x bus as single-beat transactions. Note that the $\overline{\text{CI}}$ (cache inhibit) signal always reflects the state of the caching-inhibited memory/cache access attribute (the I bit) independent of the state of HID0[DCE]. Also note that disabling the data cache does not affect the translation logic; translation for data accesses is controlled by MSR[DR].

The setting of the DCE bit must be preceded by a synchronization (**sync**) instruction to prevent the cache from being enabled or disabled in the middle of a data access. In addition, the cache must be globally flushed before it is disabled to prevent coherency problems when it is re-enabled.

Snooping is not performed when the data cache is disabled.

The Data Cache Block Set to Zero (**dcbz)** instruction will cause an alignment exception when the data cache is disabled. The touch load (Data Cache Block Touch [**dcbt**] and Data Cache Block Touch for Store [**dcbtst**] instructions are no-ops when the data cache is disabled. Other cache operations (caused by the Data Cache Block Flush (**dcbf)**, Data Cache Block Store [**dcbst**], and Data Cache Block Invalidate [**dcbi**] instructions) are not affected by disabling the cache. This can potentially cause coherency errors. For example, a **dcbf** instruction that hits a modified cache block in the disabled cache will cause a copyback to memory of potentially stale data.

### *3.4.1.3 Locking the Data Cache*

The contents of the data cache can be locked by setting the data-cache lock bit, HID0[DLOCK]. A data access that hits in a locked data cache is serviced by the cache. However, all accesses that miss in the locked cache are propagated to the L2 cache or 60x bus as single-beat transactions. Note that the $\overline{\text{CI}}$ signal always reflects the state of the caching-inhibited memory/cache access attribute (the I bit) independent of the state of HID0[DLOCK].

The 750GX treats snoop hits to a locked data cache the same as snoop hits to an unlocked data cache. However, any cache block invalidated by a snoop hit remains invalid until the cache is unlocked.

The setting of the DLOCK bit must be preceded by a **sync** instruction to prevent the data cache from being locked during a data access.

#### *3.4.1.4 Instruction-Cache Flash Invalidation*

The instruction cache is automatically invalidated when the 750GX is powered up and during a hard reset. However, a soft reset does not automatically invalidate the instruction cache. Software must use the HID0 instruction-cache flash invalidate bit (HID0[ICFI]) if instruction-cache invalidation is desired after a soft reset. Once HID0[ICFI] is set through an **mtspr** operation, the 750GX automatically clears this bit in the next clock cycle (provided that the instruction cache is enabled in the HID0 Register).

**Note:** Some PowerPC microprocessors accomplish instruction-cache flash invalidation by setting and clearing HID0[ICFI] with two consecutive **mtspr** instructions (that is, the bit is not automatically cleared by the microprocessor). Software that has this sequence of operations does not need to be changed to run on the 750GX.

#### *3.4.1.5 Enabling and Disabling the Instruction Cache*

The instruction cache can be enabled or disabled through the use of the instruction-cache enable bit, HID0[ICE]. HID0[ICE] is cleared on power-up, disabling the instruction cache.

When the instruction cache is in the disabled state (HID[ICE] = 0), the cache tag state bits are ignored, and all instruction fetches are propagated to the L2 cache or 60x bus as single-beat transactions. Note that the $\overline{\text{CI}}$ signal always reflects the state of the caching-inhibited memory/cache access attribute (the I bit) independent of the state of HID0[ICE]. Also note that disabling the instruction cache does not affect the translation logic; translation for instruction accesses is controlled by MSR[IR].

The setting of the ICE bit must be preceded by an instruction sync (**isync**) instruction to prevent the cache from being enabled or disabled in the middle of an instruction fetch. In addition, the cache must be globally flushed before it is disabled to prevent coherency problems when it is re-enabled. The Instruction Cache Block Invalidate (**icbi**) instruction is not affected by disabling the instruction cache.

#### *3.4.1.6 Locking the Instruction Cache*

The contents of the instruction cache can be locked by setting the instruction-cache lock bit, HID0[ILOCK]. An instruction fetch that hits in a locked instruction cache is serviced by the cache. However, all accesses that miss in the locked cache are propagated to the L2 cache or 60x bus as single-beat transactions. Note that the $\overline{\text{CI}}$ signal always reflects the state of the caching-inhibited memory/cache access attribute (the I bit) independent of the state of HID0[ILOCK].

The setting of the ILOCK bit must be preceded by an **isync** instruction to prevent the instruction cache from being locked during an instruction fetch.

### 3.4.2 Cache-Control Instructions

The PowerPC Architecture defines instructions for controlling both the instruction and data caches (when they exist). The cache-control instructions, **dcbt**, **dcbtst**, **dcbz**, **dcbst**, **dcbf**, **dcbi**, and **icbi**, are intended for the management of the local L1 and L2 caches. The 750GX interprets the cache-control instructions as if they pertain only to its own L1 or L2 caches. These instructions are not intended for managing other caches in the system (except to the extent necessary to maintain coherency).

The 750GX does not snoop cache-control instruction broadcasts, except for Data Cache Block Zero (**dcbz**) when M = 1. The **dcbz** instruction is the only cache-control instruction that causes a broadcast on the 60x bus (when M = 1) to maintain coherency. All other data cache-control instructions (**dcbi**, **dcbf**, **dcbst**, and **dcbz**) are not broadcast, unless broadcast is enabled through the HID0[ABE] configuration bit. Note that **dcbi**, **dcbf**, **dcbst**, and **dcbz** do broadcast to the 750GX’s L2 cache, regardless of HID0[ABE]. The **icbi** instruction is never broadcast.

### *3.4.2.1 Data Cache Block Touch (dcbt) and Data Cache Block Touch for Store (dcbtst)*

The **dcbt** and **dcbtst** instructions provide potential system performance improvement through the use of software-initiated prefetch hints. The 750GX treats these instructions identically (that is, a **dcbtst** instruction behaves exactly the same as a **dcbt** instruction on the 750GX). Note that PowerPC implementations are not required to take any action based on the execution of these instructions, but they might choose to prefetch the cache block corresponding to the effective address into their cache.

The 750GX loads the data into the cache when the address hits in the translation lookaside buffer (TLB) or the BAT, is permitted load access from the addressed page, is not directed to a direct-store segment, and is directed at a cacheable page. Otherwise, the 750GX treats these instructions as no-ops. The data brought into the cache as a result of this instruction is validated in the same manner that a load instruction would be (that is, it is marked as exclusive). The memory reference of a **dcbt** (or **dcbtst**) instruction causes the reference bit to be set. Note also that the successful execution of the **dcbt** (or **dcbtst**) instruction affects the state of the TLB and cache LRU bits as defined by the PLRU algorithm.

### *3.4.2.2 Data Cache Block Zero (dcbz)*

The effective address is computed, translated, and checked for protection violations as defined in the PowerPC Architecture. The **dcbz** instruction is treated as a store to the addressed byte with respect to address translation and protection.

If the block containing the byte addressed by the EA is in the data cache, all bytes are cleared, and the tag is marked as modified (M). If the block containing the byte addressed by the EA is not in the data cache and the corresponding page is caching-enabled, the block is established in the data cache without fetching the block from main memory, and all bytes of the block are cleared, and the tag is marked as modified (M).

If the contents of the cache block are from a page marked memory coherence required (M=1), an address-only bus transaction is run prior to clearing the cache block. The **dcbz** instruction is the only cache-control instruction that causes a broadcast on the 60x bus (when M = 1) to maintain coherency. The other cache-control instructions are not broadcast unless broadcasting is specifically enabled through the HID0[ABE] configuration bit. The **dcbz** instruction executes regardless of whether the cache is locked, but if the cache is disabled, an alignment exception is generated. If the page containing the byte addressed by the EA is caching-inhibited or write-through, then the system alignment exception handler is invoked. BAT and TLB protection violations generate DSI exceptions.

**Note:** If the target address of a **dcbz** instruction hits in the L1 cache, the 750GX requires four internal clock cycles to rewrite the cache block to zeros. On the first clock, the block is remarked as valid-unmodified, and on the last clock the block is marked as valid-modified. If a snoop request to that address is received during the middle two clocks of the **dcbz** operation, the 750GX does not properly react to the snoop operation or generate an address retry (by an $\overline{\text{ARTRY}}$ assertion) to the other master. The other bus master continues reading the data from system memory, and both the 750GX and the other bus master end up with different copies of the data. In addition, if the other bus master has a cache, the cache block is marked valid in both caches, which is not allowed in the 750GX’s 3-state cache environment.

For this reason, avoid using **dcbz** for data that is shared in real time and that is not protected during writing through higher-level software synchronization protocols (such as semaphores). Use of **dcbz** must be avoided for managing semaphores themselves. An alternative solution could be to prevent **dcbz** from hitting in the L1 cache by performing a **dcbf** to that address beforehand.

#### *3.4.2.3 Data Cache Block Store (dcbst)*

The effective address is computed, translated, and checked for protection violations as defined in the PowerPC Architecture. This instruction is treated as a load with respect to address translation and memory protection.

If the address hits in the cache and the cache block is in the exclusive (E) state, no action is taken. If the address hits in the cache and the cache block is in the modified (M) state, the modified block is written back to memory and the cache block is placed in the exclusive (E) state.

The execution of a **dcbst** instruction does not broadcast on the 60x bus unless broadcast is enabled through the HID0[ABE] bit. The function of this instruction is independent of the WIMG bit settings of the block containing the effective address. The **dcbst** instruction executes regardless of whether the cache is disabled or locked; however, a BAT or TLB protection violation generates a DSI exception.

#### *3.4.2.4 Data Cache Block Flush (dcbf)*

The effective address is computed, translated, and checked for protection violations as defined in the PowerPC Architecture. This instruction is treated as a load with respect to address translation and memory protection.

If the address hits in the cache, and the block is in the modified (M) state, the modified block is written back to memory and the cache block is placed in the invalid (I) state. If the address hits in the cache, and the cache block is in the exclusive (E) state, the cache block is placed in the invalid (I) state. If the address misses in the cache, no action is taken.

The execution of **dcbf** does not broadcast on the 60x bus unless broadcast is enabled through the HID0[ABE] bit. The function of this instruction is independent of the WIMG bit settings of the block containing the effective address. The **dcbf** instruction executes regardless of whether the cache is disabled or locked. However, a BAT or TLB protection violation generates a DSI exception.

#### *3.4.2.5 Data Cache Block Invalidate (dcbi)*

The effective address is computed, translated, and checked for protection violations as defined in the PowerPC Architecture. This instruction is treated as a store with respect to address translation and memory protection.

If the address hits in the cache, the cache block is placed in the invalid (I) state, regardless of whether the data is modified. Because this instruction can effectively destroy modified data, it is privileged (that is, **dcbi** is available to programs at the supervisor privilege level, MSR[PR] = 0). The execution of **dcbi** does not broadcast on the 60x bus unless broadcast is enabled through the HID0[ABE] bit. The function of this instruction is independent of the WIMG bit settings of the block containing the effective address. The **dcbi** instruction executes regardless of whether the cache is disabled or locked. However, a BAT or TLB protection violation generates a DSI exception.

### *3.4.2.6 Instruction Cache Block Invalidate (icbi)*

For the **icbi** instruction, the effective address is not computed or translated, so it cannot generate a protection violation or exception. This instruction performs a virtual lookup into the instruction cache (index only). All ways of the selected instruction cache set are invalidated.

The **icbi** instruction is not broadcast on the 60x bus. The **icbi** instruction invalidates the cache blocks independent of whether the cache is disabled or locked.

## 3.5 Cache Operations

This section describes the 750GX's cache operations.

### 3.5.1 Cache-Block-Replacement/Castout Operations

Both the instruction and data cache use a pseudo least-recently-used (PLRU) replacement algorithm when a new block needs to be placed in the cache. When the data to be replaced is in the modified (M) state, that data is written into a castout buffer while the missed data is being accessed on the bus. When the load completes, the 750GX then pushes the replaced cache block from the castout buffer to the L2 cache (if L2 is enabled) or to main memory (if L2 is disabled).

The replacement logic first checks to see if there are any invalid blocks in the set and chooses the lowest-order, invalid block (L[0–7]) as the replacement target. If all eight blocks in the set are valid, the PLRU algorithm is used to determine which block should be replaced. The PLRU algorithm is shown in *Figure 3-5* on page 137.

Each cache is organized as eight blocks per set by 128 sets. There is a valid bit for each block in the cache, L[0–7]. When all eight blocks in the set are valid, the PLRU algorithm is used to select the replacement target. There are seven PLRU bits, B[0–6] for each set in the cache. For every hit in the cache, the PLRU bits are updated using the rules specified in *Table 3-2* on page 138.

*Figure 3-5. PLRU Replacement Algorithm*

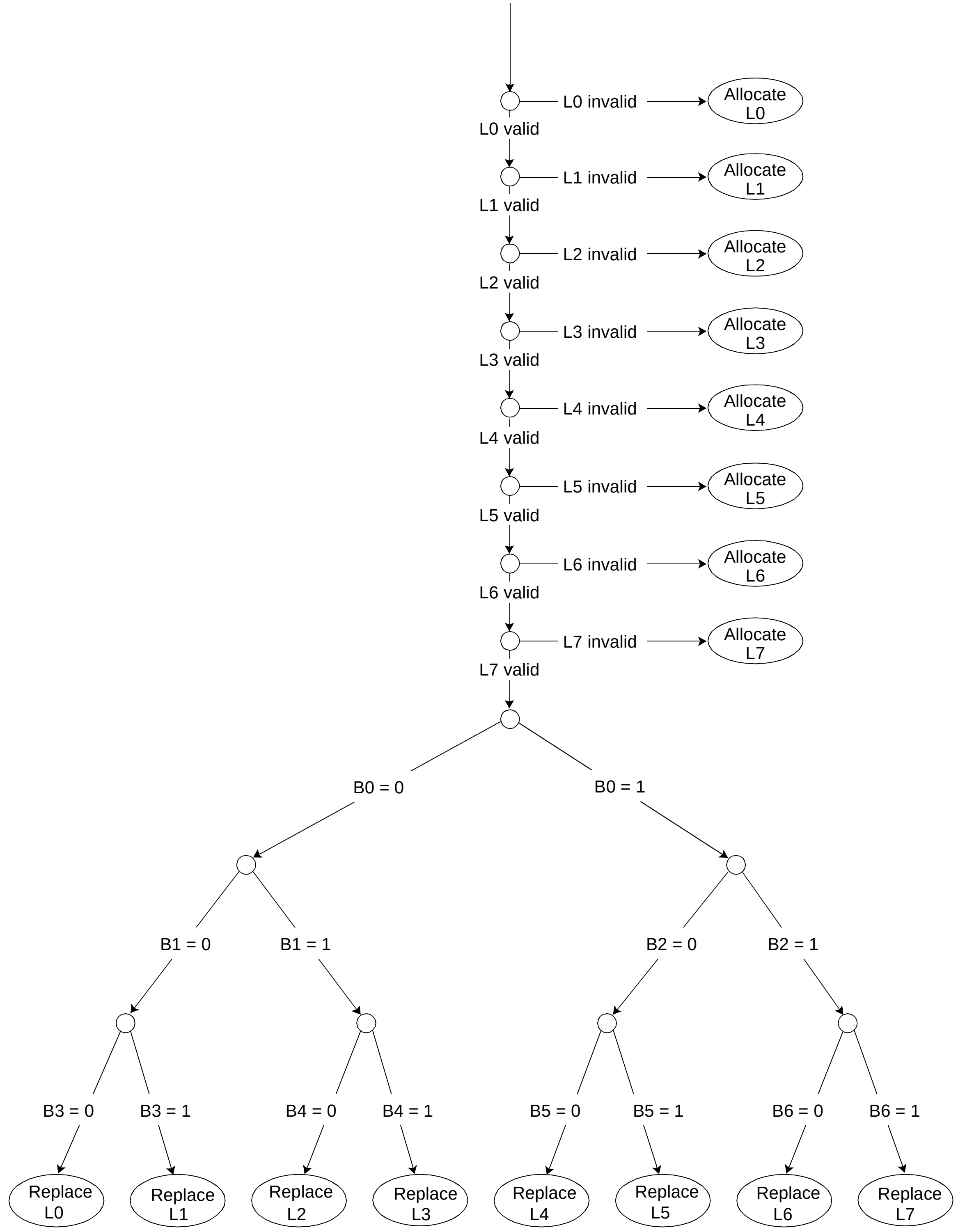


*Table 3-2. PLRU Bit Update Rules*

| If the current access is to: | Then the PLRU bits are changed to:[1] | | | | | | |
|---|---|---|---|---|---|---|---|
| | B0 | B1 | B2 | B3 | B4 | B5 | B6 |
| L0 | 1 | 1 | x | 1 | x | x | x |
| L1 | 1 | 1 | x | 0 | x | x | x |
| L2 | 1 | 0 | x | x | 1 | x | x |
| L3 | 1 | 0 | x | x | 0 | x | x |
| L4 | 0 | x | 1 | x | x | 1 | x |
| L5 | 0 | x | 1 | x | x | 0 | x |
| L6 | 0 | x | 0 | x | x | x | 1 |
| L7 | 0 | x | 0 | x | x | x | 0 |

[1] x = Does not change

If all eight blocks are valid, then a block is selected for replacement according to the PLRU bit encodings shown in *Table 3-3*.

*Table 3-3. PLRU Replacement Block Selection*

| If the PLRU bits are: | | | | | | Then the block selected for replacement is: |
|---|---|---|---|---|---|---|
| B0 | 0 | B1 | 0 | B3 | 0 | L0 |
| | 0 | | 0 | | 1 | L1 |
| | 0 | | 1 | B4 | 0 | L2 |
| | 0 | | 1 | | 1 | L3 |
| | 1 | B2 | 0 | B5 | 0 | L4 |
| | 1 | | 0 | | 1 | L5 |
| | 1 | | 1 | B6 | 0 | L6 |
| | 1 | | 1 | | 1 | L7 |

During power-up or hard reset, all the valid bits of the blocks are cleared, and the PLRU bits cleared to point to block L0 of each set. Note that this is also the state of the data or instruction cache after setting their respective flash invalidate bit, HID0[DCFI] or HID0[ICFI].

### 3.5.2 Cache Flush Operations

The instruction cache can be invalidated by executing a series of **icbi** instructions or by setting HID0[ICFI]. The data cache can be invalidated by executing a series of **dcbi** instructions or by setting HID0[DCFI].

Any modified entries in the data cache can be copied back to memory (flushed) by using the **dcbf** instruction or by executing a series of 12 uniquely addressed load or **dcbz** instructions to each of the 128 sets. The address space should not be shared with any other process to prevent snoop hit invalidations during the flushing routine. Exceptions should be disabled during this time so that the PLRU algorithm is not disturbed.

The data-cache flush assist bit, HID0[DCFA], simplifies the software flushing process. When set, HID0[DCFA] forces the PLRU replacement algorithm to ignore the invalid entries and follow the replacement sequence defined by the PLRU bits. This reduces the series of uniquely addressed load or **dcbz** instructions to eight per set. HID0[DCFA] should be set just prior to the beginning of the cache flush routine and cleared after the series of instructions is complete.

The L2 flush mechanism is similar to the L1 data-cache flush mechanism. The L2 flush requires that the entire data cache be flushed prior to flushing the L2 cache. Also, exceptions must be disabled during the L2 flush so that the LR and PLRU algorithms are not disturbed. The L2 cache can be flushed by executing uniquely addressed load instructions to each of the 32-byte blocks of the L2 cache. This can be done by loading a contiguous 1-MB block of memory. The loads must not hit in the L1 cache in order to effect a flush of the L2 cache.

### 3.5.3 Data-Cache Block-Fill Operations

The 750GX's data-cache blocks are filled in four beats of 64 bits each, with the critical double word loaded first. The data cache is not blocked to internal accesses while the load (caused by a cache miss) completes. This functionality is sometimes referred to as 'hits under misses,' because the cache can service a hit while a cache miss fill is waiting to complete. The critical-double-word read from memory is simultaneously written to the data cache and forwarded to the requesting unit, thus minimizing stalls due to cache fill latency.

A cache block is filled after a read miss or write miss (read-with-intent-to-modify) occurs in the cache. The cache block that corresponds to the missed address is updated by a burst transfer of the data from the L2 or system memory. Note that if a read miss occurs in a system with multiple bus masters, and the data is modified in another cache, the modified data is first written to external memory before the cache fill occurs.

### 3.5.4 Instruction-Cache Block-Fill Operations

The 750GX's instruction-cache blocks are loaded in four beats of 64 bits each, with the critical double word loaded first. The instruction cache is not blocked to internal accesses while the fetch (caused by a cache miss) completes. On a cache miss, the critical and following double words read from memory are simultaneously written to the instruction cache and forwarded to the instruction queue, thus minimizing stalls due to cache fill latency. There is no snooping of the instruction cache.

### 3.5.5 Data-Cache Block-Push Operations

When a cache block in the 750GX is snooped and hit by another bus master and the data is modified, the cache block must be written to memory and made available to the snooping device. The cache block is said to be pushed out onto the 60x bus.

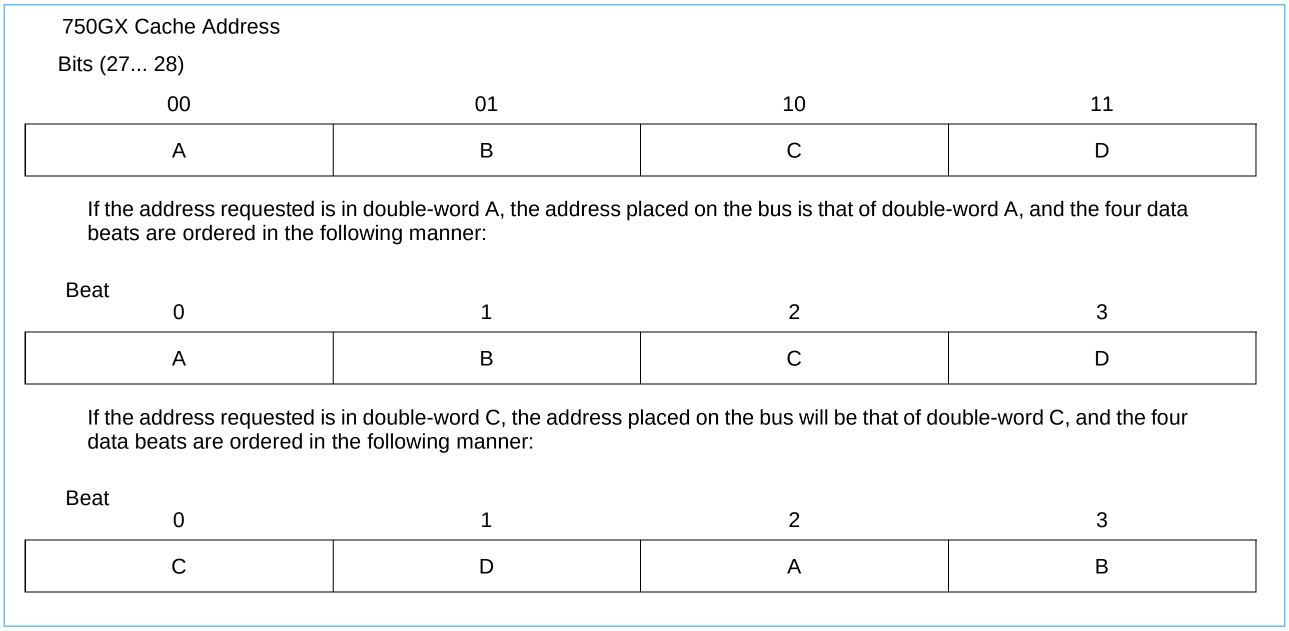
## 3.6 L1 Caches and 60x Bus Transactions

The 750GX transfers data to and from the cache in single-beat transactions of two words, or in 4-beat transactions of eight words which fill a cache block. Single-beat bus transactions can transfer from one to eight bytes to or from the 750GX, and can be misaligned. Single-beat transactions can be caused by cache write-through accesses, caching-inhibited accesses (WIMG = x1xx), accesses when the cache is disabled (HID0[DCE] bit is cleared), or accesses when the cache is locked (HID0[DLOCK] bit is cleared).

Burst transactions on the 750GX always transfer eight words of data at a time, and are aligned to a double-word boundary. The 750GX transfer burst ($\overline{\text{TBST}}$) output signal indicates to the system whether the current transaction is a single-beat transaction or 4-beat burst transfer. Burst transactions have an assumed address order. For cacheable read operations, instruction fetches, or cacheable, non-write-through write operations that miss the cache, the 750GX presents the double-word-aligned address associated with the load/store instruction or instruction fetch that initiated the transaction.

As shown in *Figure 3-6*, the first quadword contains the address of the load/store or instruction fetch that missed the cache. This minimizes latency by allowing the critical code or data to be forwarded to the processor before the rest of the block is filled. For all other burst operations, however, the entire block is transferred in order. Critical-double-word-first fetching on a cache miss applies to both the data and instruction cache.

*Figure 3-6. 750GX Cache Addresses*



750GX Cache Address

Bits (27... 28)

| 00 | 01 | 10 | 11 |
|---|---|---|---|
| A | B | C | D |

If the address requested is in double-word A, the address placed on the bus is that of double-word A, and the four data beats are ordered in the following manner:

Beat

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| A | B | C | D |

If the address requested is in double-word C, the address placed on the bus will be that of double-word C, and the four data beats are ordered in the following manner:

Beat

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| C | D | A | B |

### 3.6.1 Read Operations and the MEI Protocol

The MEI coherency protocol affects how the 750GX data cache performs read operations on the 60x bus. All reads (except for caching-inhibited reads) are encoded on the bus as read-with-intent-to-modify (RWITM) to force flushing of the addressed cache block from other caches in the system.

The MEI coherency protocol also affects how the 750GX snoops read operations on the 60x bus. All reads snooped from the 60x bus (except for caching-inhibited reads) are interpreted as RWITM to cause flushing from the 750GX's cache. Single-beat reads ($\overline{\text{TBST}}$ negated) are interpreted by the 750GX as caching inhibited.

These actions for read operations allow the 750GX to operate successfully (coherently) on the bus with other bus masters that implement either the 3-state MEI or a 4-state MESI cache-coherency protocol.

### 3.6.2 Bus Operations Caused by Cache-Control Instructions

The cache-control, TLB management, and synchronization instructions supported by the 750GX can affect or be affected by the operation of the 60x bus. The operation of the instructions can also indirectly cause bus transactions to be performed, or their completion can be linked to the bus.

The **dcbz** instruction is the only cache-control instruction that causes an address-only broadcast on the 60x bus. All other data cache-control instructions (**dcbi**, **dcbf**, **dcbst**, and **dcbz**) are not broadcast unless specifically enabled through the HID0[ABE] configuration bit. Note that **dcbi**, **dcbf**, **dcbst**, and **dcbz** do broadcast to the 750GX's L2 cache, regardless of HID0[ABE]. HID0[ABE] also controls the broadcast of the **sync** and Enforce In-Order Execution of I/O (**eieio**) instructions.

The **icbi** instruction is never broadcast. No broadcasts by other masters are snooped by the 750GX (except for **dcbz** kill block transactions). For detailed information on the cache-control instructions, see *Chapter 2, Programming Model,* on page 57.

*Table 3-4* provides an overview of the bus operations initiated by cache-control instructions. Note that the information in this table assumes that the WIM bits are set to 001; that is, the cache is operating in write-back mode, caching is enabled and coherency is enforced.

*Table 3-4. Bus Operations Caused by Cache-Control Instructions (WIM = 001)*

| Instruction | Current Cache State | Next Cache State | Bus Operation | Comment |
|---|---|---|---|---|
| **sync** | Don't care | No change | **sync**<br>(if enabled in HID0[ABE]) | Waits for memory queues to complete bus activity. |
| **tlbie** | — | — | None | TLB invalidate entry. |
| **tlbsync** | — | — | None | TLB synchronization. Waits for the negation of the $\overline{\text{TLBSYNC}}$ input signal to complete. |
| **eieio** | Don't care | No change | **eieio**<br>(if enabled in HID0[ABE]) | Address-only bus operation. |
| **icbi** | Don't care | I | None | — |
| **dcbi** | Don't care | I | Kill block<br>(if enabled in HID0[ABE]) | Address-only bus operation. |
| **dcbf** | I, E | I | Flush block<br>(if enabled in HID0[ABE]) | Address-only bus operation. |
| **dcbf** | M | I | Write with kill | Block is pushed. |
| **dcbst** | I, E | No change | Clean block<br>(if enabled in HID0[ABE]) | Address-only bus operation. |
| **dcbst** | M | E | Write with kill | Block is pushed. |
| **dcbz** | I | M | Write with kill | — |
| **dcbz** | E, M | M | Kill block | Writes over modified data. |
| **dcbt** | I | E | Read-with-intent-to-modify | Fetched cache block is stored in the cache. |
| **dcbt** | E, M | No change | None | — |
| **dcbtst** | I | E | Read-with-intent-to-modify | Fetched cache block is stored in the cache. |
| **dcbtst** | E,M | No change | None | — |

For additional details about the specific bus operations performed by the 750GX, see *Chapter 8, Bus Interface Operation,* on page 279 in this manual.

### 3.6.3 Snooping

The 750GX maintains data-cache coherency in hardware by coordinating activity between the data cache, the bus interface logic, the L2 cache, and the memory system. The 750GX has a copy-back cache which relies on bus snooping to maintain cache coherency with other caches in the system. For the 750GX, the coherency size of the bus is the size of a cache block, 32 bytes. This means that any bus transactions that cross an aligned 32-byte boundary must present a new address onto the bus at that boundary for proper snoop operation by the 750GX, or they must operate noncoherently with respect to the 750GX.

As bus operations are performed on the bus by other bus masters, the 750GX’s bus snooping logic monitors the addresses and transfer attributes that are referenced. The 750GX snoops the bus transactions during the cycle that $\overline{\text{TS}}$ is asserted for any of the following qualified snoop conditions:

- The global signal ($\overline{\text{GBL}}$) is asserted indicating that coherency enforcement is required.
- A reservation is currently active in the 750GX as the result of an **lwarx** instruction, and the transfer type attributes ($\overline{\text{TT}}$[0–4]) indicate a write or kill operation. These transactions are snooped regardless of whether $\overline{\text{GBL}}$ is asserted to support reservations in the MEI cache protocol.

All transactions snooped by the 750GX are checked for correct address-bus parity. Every assertion of $\overline{\text{TS}}$ detected by the 750GX (whether snooped or not) must be followed by an accompanying assertion of address acknowledge ($\overline{\text{AACK}}$).

Once a qualified snoop condition is detected on the bus, the snooped address associated with $\overline{\text{TS}}$ is compared against the data-cache tags, memory queues, and/or other storage elements as appropriate. The L1 data-cache tags and L2 cache tags are snooped for standard data-cache-coherency support. No snooping is done in the instruction cache for coherency.

The memory queues are snooped for pipeline collisions and memory coherency collisions. A pipeline collision is detected when another bus master addresses any portion of a line that this 750GX’s data cache is currently in the process of loading (L1 loading from L2, or L1/L2 loading from memory). A memory coherency collision occurs when another bus master addresses any portion of a line that the 750GX has currently queued to write to memory from the data cache (castout or copy-back), but has not yet been granted bus access to perform.

If a snooped transaction results in a cache hit or pipeline collision or memory queue collision, the 750GX asserts $\overline{\text{ARTRY}}$ on the 60x bus. The current bus master, detecting the assertion of the $\overline{\text{ARTRY}}$ signal, should cancel the transaction and retry it at a later time, so that the 750GX can first perform a write operation back to memory from its cache or memory queues. The 750GX might also retry a bus transaction if it is unable to snoop the transaction on that cycle due to internal resource conflicts. Additional snoop action might be forwarded to the cache as a result of a snoop hit in some cases (a cache push of modified data, or a cache-block invalidation). There is no immediate way for another CPU bus agent to determine the cause of the 750GX $\overline{\text{ARTRY}}$.

**Implementation Note:** Snooping of the memory queues for pipeline collisions, as described above, is performed for burst read operations in progress only. In this case, the read address has completed on the bus; however, the data tenure might be either in-progress or not yet started by the processor. During this time the 750GX will retry any other global access to that line by another bus master until all data has been received in its L1 cache. Pipeline collisions, however, do not apply for burst write operations in progress. If the 750GX has completed an address tenure for a burst write, and is currently waiting for a data-bus grant or is currently transferring data to memory, it will not generate an address retry to another bus master that addresses the line. It is the responsibility of the memory system to handle this collision (usually by keeping the data transactions to memory in order). Note also that all burst writes by the 750GX are performed as nonglobal, and hence do not normally enable snooping, even for address collision purposes. (Snooping might still occur for reservation cancelling purposes.)

### 3.6.4 Snoop Response to 60x Bus Transactions

There are several bus transaction types defined for the 60x bus. The transactions in *Table 3-5* correspond to the transfer type signals TT[0–4], which are described in *Section 7.2.4.1, Transfer Type (TT[0–4]),* on page 256.

The 750GX never retries a transaction in which $\overline{\text{GBL}}$ is not asserted, even if the tags are busy or there is a tag hit. Reservations are snooped regardless of the state of $\overline{\text{GBL}}$.

*Table 3-5. Response to Snooped Bus Transactions* (Page 1 of 3)

| Snooped Transaction | TT[0–4] | 750GX Response |
|---|---|---|
| Clean block | 00000 | No action is taken. |
| Flush block | 00100 | No action is taken. |
| SYNC | 01000 | No action is taken. |
| Kill block | 01100 | The kill block operation is an address-only bus transaction initiated when a **dcbz** or **dcbi** instruction is executed.<br>• If the addressed cache block is in the exclusive (E) state, the cache block is placed in the invalid (I) state.<br>• If the addressed cache block is in the modified (M) state, the 750GX asserts $\overline{\text{ARTRY}}$ and initiates a push of the modified block out of the cache and the cache block is placed in the invalid (I) state.<br>• If the address misses in the cache, no action is taken.<br>Any reservation associated with the address is canceled. |
| EIEIO | 10000 | No action is taken. |
| External control word write | 10100 | No action is taken. |
| TLB invalidate | 11000 | No action is taken. |
| External control word read | 11100 | No action is taken. |
| **lwarx** reservation set | 00001 | No action is taken. |
| Reserved | 00101 | — |
| TLBSYNC | 01001 | No action is taken. |
| ICBI | 01101 | No action is taken. |
| Reserved | 1XX01 | — |
| Write-with-flush | 00010 | A write-with-flush operation is a single-beat or burst transaction initiated when a caching-inhibited or write-through store instruction is executed.<br>• If the addressed cache block is in the exclusive (E) state, the cache block is placed in the invalid (I) state.<br>• If the addressed cache block is in the modified (M) state, the 750GX asserts $\overline{\text{ARTRY}}$ and initiates a push of the modified block out of the cache, and the cache block is placed in the invalid (I) state.<br>• If the address misses in the cache, no action is taken.<br>Any reservation associated with the address is canceled. |

*Table 3-5. Response to Snooped Bus Transactions* (Page 2 of 3)

| Snooped Transaction | TT[0–4] | 750GX Response |
|---|---|---|
| Write-with-kill | 00110 | A write-with-kill operation is a burst transaction initiated due to a castout, caching-enabled push, or snoop copy-back.<br>• If the address hits in the cache, the cache block is placed in the invalid (I) state (killing modified data that might have been in the block).<br>• If the address misses in the cache, no action is taken.<br>Any reservation associated with the address is canceled. |
| Read | 01010 | A read operation is used by most single-beat and burst load transactions on the bus.<br>For single-beat, caching-inhibited read transaction:<br>• If the addressed cache block is in the exclusive (E) state, the cache block remains in the exclusive (E) state.<br>• If the addressed cache block is in the modified (M) state, the 750GX asserts $\overline{\text{ARTRY}}$ and initiates a push of the modified block out of the cache, and the cache block is placed in the exclusive (E) state.<br>• If the address misses in the cache, no action is taken.<br>For burst read transactions:<br>• If the addressed cache block is in the exclusive (E) state, the cache block is placed in the invalid (I) state.<br>• If the addressed cache block is in the modified (M) state, the 750GX asserts $\overline{\text{ARTRY}}$ and initiates a push of the modified block out of the cache, and the cache block is placed in the invalid (I) state.<br>• If the address misses in the cache, no action is taken. |
| Read-with-intent-to-modify (RWITM) | 01110 | A RWITM operation is issued to acquire exclusive use of a memory location for the purpose of modifying it.<br>• If the addressed cache block is in the exclusive (E) state, the cache block is placed in the invalid (I) state.<br>• If the addressed cache block is in the modified (M) state, the 750GX asserts $\overline{\text{ARTRY}}$ and initiates a push of the modified block out of the cache, and the cache block is placed in the invalid (I) state.<br>• If the address misses in the cache, no action is taken. |
| Write-with-flush-atomic | 10010 | Write-with-flush-atomic operations occur after the processor issues an **stwcx.** instruction.<br>• If the addressed cache block is in the exclusive (E) state, the cache block is placed in the invalid (I) state.<br>• If the addressed cache block is in the modified (M) state, the 750GX asserts $\overline{\text{ARTRY}}$ and initiates a push of the modified block out of the cache, and the cache block is placed in the invalid (I) state.<br>• If the address misses in the cache, no action is taken.<br>Any reservation is canceled, regardless of the address. |
| Reserved | 10110 | — |
| Read-atomic | 11010 | Read atomic operations appear on the bus in response to **lwarx** instructions and generate the same snooping responses as read operations. |
| Read-with-intent-to-modify-atomic | 11110 | The RWITM atomic operations appear on the bus in response to **stwcx.** instructions and generate the same snooping responses as RWITM operations. |
| Reserved | 00011 | — |
| Reserved | 00111 | — |

*Table 3-5. Response to Snooped Bus Transactions* (Page 3 of 3)

| Snooped Transaction | TT[0–4] | 750GX Response |
|---|---|---|
| Read-with-no-intent-to-cache (RWNITC) | 01011 | A RWNITC operation is issued to acquire exclusive use of a memory location with no intention of modifying the location.<br>• If the addressed cache block is in the exclusive (E) state, the cache block remains in the exclusive (E) state.<br>• If the addressed cache block is in the modified (M) state, the 750GX asserts $\overline{\text{ARTRY}}$ and initiates a push of the modified block out of the cache, and the cache block is placed in the exclusive (E) state.<br>• If the address misses in the cache, no action is taken. |
| Reserved | 01111 | — |
| Reserved | 1XX11 | — |

### 3.6.5 Transfer Attributes

In addition to the address and transfer type signals, the 750GX supports the transfer attribute signals: $\overline{\text{TBST}}$, TSIZ[0–2], $\overline{\text{WT}}$, $\overline{\text{CI}}$, and $\overline{\text{GBL}}$. The $\overline{\text{TBST}}$ and TSIZ[0–2] signals indicate the data-transfer size for the bus transaction.

The $\overline{\text{WT}}$ signal reflects the write-through status (the complement of the W bit) for the transaction as determined by the MMU address translation during write operations. $\overline{\text{WT}}$ is asserted for burst writes due to **dcbf** (flush) and **dcbst** (clean) instructions, and for snoop pushes; $\overline{\text{WT}}$ is negated for External Control Out Word Indexed (**ecowx**) transactions. Since the write-through status is not meaningful for reads, the 750GX uses the $\overline{\text{WT}}$ signal during read transactions to indicate that the transaction is an instruction fetch ($\overline{\text{WT}}$ negated), or not an instruction fetch ($\overline{\text{WT}}$ asserted).

The $\overline{\text{CI}}$ signal reflects the caching-inhibited/enabled status (the complement of the I bit) of the transaction as determined by the MMU address translation even if the L1 caches are disabled or locked. $\overline{\text{CI}}$ is always asserted for External Control In Word Indexed (**eciwx**) and **ecowx** bus transactions independent of the address translation.

The $\overline{\text{GBL}}$ signal reflects the memory coherency requirements (the complement of the M bit) of the transaction as determined by the MMU address translation. Castout and snoop copy-back operations (TT[0–4] = 00110) are generally marked as nonglobal ($\overline{\text{GBL}}$ negated) and are not snooped (except for reservation monitoring). Other masters, however, might perform direct memory access (DMA) write operations with this encoding but marked global ($\overline{\text{GBL}}$ asserted); thus, the operation must be snooped.

*Table 3-6* summarizes the address and transfer attribute information presented on the bus by the 750GX for various master or snoop-related transactions.

*Table 3-6. Address/Transfer Attribute Summary*

| Bus Transaction | A[0–31] | TT[0–4] | TBST | TSIZ[0–2] | GBL | WT | CI |
|---|---|---|---|---|---|---|---|
| Instruction fetch operations: | | | | | | | |
| Burst (caching-enabled) | PA[0–28] \|\| 0b000 | 0 1 1 1 0 | 0 | 0 1 0 | ¬ M | 1 | 1* |
| Single-beat read (caching-inhibited or cache disabled) | PA[0–28] \|\| 0b000 | 0 1 0 1 0 | 1 | 0 0 0 | ¬ M | 1 | ¬ I |
| Single-beat read (caching-inhibited or cache disabled, 32-bit bus) | PA(0:29) \|\| 00 | 0 1 0 1 0 | 1 | 1 0 0 | ¬ M | 1 | ¬ I |
| Data-cache operations: | | | | | | | |
| Cache-block fill (due to load or store miss) | PA[0–28] \|\| 0b000 | A 1 1 1 0 | 0 | 0 1 0 | ¬ M | 0 | 1* |
| Castout<br>(normal replacement) | CA[0–26] \|\| 0b00000 | 0 0 1 1 0 | 0 | 0 1 0 | 1 | 1 | 1* |
| Push (cache-block push due to **dcbf** or **dcbst**) | PA[0–26] \|\| 0b00000 | 0 0 1 1 0 | 0 | 0 1 0 | 1 | 0 | 1* |
| Snoop copyback | CA[0–26] \|\| 0b00000 | 0 0 1 1 0 | 0 | 0 1 0 | 1 | 0 | 1* |
| Data-cache bypass operations: | | | | | | | |
| Single-beat read (caching-inhibited or cache disabled) | PA[0–31] | A 1 0 1 0 | 1 | S S S | ¬ M | 0 | ¬ I |
| Single-beat write (caching-inhibited, write-through, or cache disabled) | PA[0–31] | 0 0 0 1 0 | 1 | S S S | ¬ M | ¬W | ¬ I |
| Special instructions: | | | | | | | |
| **dcbz** (address-only) | PA[0–28] \|\| 0b000 | 0 1 1 0 0 | 0 | 0 1 0 | 0* | 0 | 1* |
| **dcbi** (if HID0[ABE] = 1, address-only) | PA[0–26] \|\| 0b00000 | 0 1 1 0 0 | 0 | 0 1 0 | ¬ M | 0 | 1* |
| **dcbf** (if HID0[ABE] = 1, address-only) | PA[0–26] \|\| 0b00000 | 0 0 1 0 0 | 0 | 0 1 0 | ¬ M | 0 | 1* |
| **dcbst** (if HID0[ABE] = 1, address-only) | PA[0–26] \|\| 0b00000 | 0 0 0 0 0 | 0 | 0 1 0 | ¬ M | 0 | 1* |
| **sync** (if HID0[ABE] = 1, address-only) | 0x0000_0000 | 0 1 0 0 0 | 0 | 0 1 0 | 0 | 0 | 0 |
| **eieio** (if HID0[ABE] = 1, address-only) | 0x0000_0000 | 1 0 0 0 0 | 0 | 0 1 0 | 0 | 0 | 0 |
| **stwcx.** (always single-beat write) | PA[0–29] \|\| 0b00 | 1 0 0 1 0 | 1 | 1 0 0 | ¬ M | ¬ W | ¬ I |
| **eciwx** | PA[0–29] \|\| 0b00 | 1 1 1 0 0 | EAR[28–31] | | 1 | 0 | 0 |
| **ecowx** | PA[0–29] \|\| 0b00 | 1 0 1 0 0 | EAR[28–31] | | 1 | 1 | 0 |

**Note:**

PA = Physical address, CA = Cache address.

W,I,M = WIM state from address translation; ¬ = complement; 0*or 1* = WIM state implied by transaction type in table

For instruction fetches, reflection of the M bit must be enabled through HID0[IFEM].

A = Atomic; high if **lwarx**, low otherwise

S = Transfer size

Special instructions listed might not generate bus transactions depending on cache state.

## 3.7 MEI State Transactions

*Table 3-7* shows MEI state transitions for various operations. Bus operations are described in *Table 3-4* on page 141.

*Table 3-7. MEI State Transitions* (Page 1 of 3)

| Operation | Cache Operation | Bus Sync | WIM | Current Cache State | Next Cache State | Cache Actions | Bus Operation |
|---|---|---|---|---|---|---|---|
| Load (T = 0) | Read | No | x0x | I | Same | Cast out of modified block (as required). | Write-with-kill |
| | | | | | | Pass 4-beat read to memory queue. | Read |
| Load (T = 0) | Read | No | x0x | E,M | Same | Read data from cache. | — |
| Load (T = 0) | Read | No | x1x | I | Same | Pass single-beat read to memory queue. | Read |
| Load (T = 0) | Read | No | x1x | E | Same | Pass single-beat read to memory queue. | Read |
| Load (T = 0) | Read | No | x1x | M | Same | Pass single-beat read to memory queue. | Read |
| **lwarx** | Read | Acts like other reads but bus operation uses special encoding. | | | | | |
| Store (T = 0) | Write | No | 00x | I | Same | Cast out of modified block (if necessary). | Write-with-kill |
| | | | | | | Pass RWITM to memory queue. | RWITM |
| Store (T = 0) | Write | No | 00x | E,M | M | Write data to cache. | — |
| Store ¦ **stwcx.** (T = 0) | Write | No | 10x | I | Same | Pass single-beat write to memory queue. | Write-with-flush |
| Store ¦ **stwcx.** (T = 0) | Write | No | 10x | E | Same | Write data to cache. | — |
| | | | | | | Pass single-beat write to memory queue. | Write-with-flush |
| Store ¦ **stwcx.** (T = 0) | Write | No | 10x | M | Same | Push block to write queue. | Write-with-kill |
| Store (T = 0) or **stwcx.** | Write | No | x1x | I | Same | Pass single-beat write to memory queue. | Write-with-flush |
| Store (T = 0) or **stwcx.** | Write | No | x1x | E | Same | Pass single-beat write to memory queue. | Write-with-flush |
| Store (T = 0) or **stwcx.** | Write | No | x1x | M | Same | Pass single-beat write to memory queue. | Write-with-flush |
| | | | | | | Push block to write queue | Write-with-kill |
| **stwcx.** | Conditional write | If the reserved bit is set, this operation is like other writes except the bus operation uses a special encoding. | | | | | |
| **dcbf** | Data-cache-block flush | No | xxx | I,E | Same | Pass flush. | Flush |
| | | | | Same | I | State change only. | — |
| **dcbf** | Data-cache-block flush | No | xxx | M | I | Push block to write queue. | Write-with-kill |
| **Note:** Single-beat writes are not snooped in the write queue. | | | | | | | |

*Table 3-7. MEI State Transitions* (Page 2 of 3)

| Operation | Cache Operation | Bus Sync | WIM | Current Cache State | Next Cache State | Cache Actions | Bus Operation |
|---|---|---|---|---|---|---|---|
| **dcbst** | Data-cache-block store | No | xxx | I,E | Same | **dcbst.** | — |
| | | | | | | Pass clean. | Clean |
| | | | | Same | Same | No action. | — |
| **dcbst** | Data-cache-block store | No | xxx | M | E | Push block to write queue. | Write-with-kill |
| **dcbz** | Data-cache-block set to zero | No | x1x | x | x | Alignment trap. | — |
| **dcbz** | Data-cache-block set to zero | No | 10x | x | x | Alignment trap. | — |
| **dcbz** | Data-cache-block set to zero | Yes | 00x | I | Same | Cast out of modified block. | Write-with-kill |
| | | | | | | Pass kill. | Kill |
| | | | | Same | M | Clear block. | — |
| **dcbz** | Data-cache-block set to zero | No | 00x | E,M | M | Clear block. | — |
| **dcbt** | Data-cache-block touch | No | x1x | I | Same | Pass single-beat read to memory queue. | Read |
| **dcbt** | Data-cache-block touch | No | x1x | M | I | Push block to write queue. | Write-with-kill |
| **dcbt** | Data-cache-block touch | No | x0x | I | Same | Cast out of modified block (as required). | Write-with-kill |
| | | | | | | Pass 4-beat read to memory queue. | Read |
| **dcbt** | Data-cache-block touch | No | x0x | E,M | Same | No action. | — |
| Single-beat read | Reload dump 1 | No | xxx | I | Same | Forward data_in. | — |
| 4-beat read (double-word-aligned) | Reload dump | No | xxx | I | E | Write data_in to cache. | — |
| 4-beat write (double-word-aligned) | Reload dump | No | xxx | I | M | Write data_in to cache. | — |
| E⟶I | Snoop write or kill | No | xxx | E | I | State change only (committed). | — |
| M⟶I | Snoop kill | No | xxx | M | I | State change only (committed). | — |
| Push M⟶I | Snoop flush | No | xxx | M | I | Conditionally push. | Write-with-kill |
| Push M⟶E | Snoop clean | No | xxx | M | E | Conditionally push. | Write-with-kill |

**Note:** Single-beat writes are not snooped in the write queue.

*Table 3-7. MEI State Transitions* (Page 3 of 3)

| Operation | Cache Operation | Bus Sync | WIM | Current Cache State | Next Cache State | Cache Actions | Bus Operation |
|---|---|---|---|---|---|---|---|
| **tlbie** | TLB invalidate | No | xxx | x | x | Pass TLBI. | — |
| | | | | | | No action. | — |
| **sync** | Synchronization | No | xxx | x | x | Pass **sync.** | — |
| | | | | | | No action. | — |
| **Note:** Single-beat writes are not snooped in the write queue. | | | | | | | |

# 4. Exceptions

The operating environment architecture (OEA) portion of the PowerPC Architecture defines the mechanism by which PowerPC processors implement exceptions (referred to as interrupts in the architecture specification). Exception conditions can be defined at other levels of the architecture. For example, the user instruction set architecture (UISA) defines conditions that can cause floating-point exceptions; the OEA defines the mechanism by which the exception is taken.

The PowerPC exception mechanism allows the processor to change to supervisor state as a result of unusual conditions arising in the execution of instructions and from external signals, bus errors, or various internal conditions. When exceptions occur, information about the state of the processor is saved to certain registers, and the processor begins execution at an address (exception vector) predetermined for each exception. Processing of exceptions begins in supervisor mode.

Although multiple exception conditions can map to a single exception vector, often a more specific condition can be determined by examining a register associated with the exception—for example, the Data Storage Interrupt Status Register (DSISR) and the Floating-Point Status and Control Register (FPSCR). The high-order bits of the Machine State Register (MSR) are also set for some exceptions. Software can explicitly enable or disable some exception conditions.

The PowerPC Architecture requires that exceptions be taken in program order. Therefore, although a particular implementation might recognize exception conditions out of order, they are handled strictly in order with respect to the instruction stream. When an instruction-caused exception is recognized, any unexecuted instructions that appear earlier in the instruction stream, including any that have not yet entered the execute state, are required to complete before the exception is taken. For example, if a single instruction encounters multiple exception conditions, those exceptions are taken and handled based on the priority of the exception. Likewise, exceptions that are asynchronous and precise are recognized when they occur, but are not handled until all instructions currently in the execute stage successfully complete execution and report their results.

To prevent loss of state information, exception handlers must save the information stored in the Machine Status Save/Restore Registers, SRR0 and SRR1, soon after the exception is taken to prevent this information from being lost due to another exception being taken. Because exceptions can occur while an exception handler routine is executing, multiple exceptions can become nested. It is up to the exception handler to save the necessary state information if control is to return to the excepting program.

In many cases, after the exception handler returns, there is an attempt to execute the instruction that caused the exception (for example, a page fault). Instruction execution continues until the next exception condition is encountered. Recognizing and handling exception conditions sequentially guarantees that the machine state is recoverable and processing can resume without losing instruction results.

In this book, the following terms are used to describe the stages of exception processing.

Recognition — Exception recognition occurs when the condition that can cause an exception is identified by the processor.

Taken — An exception is said to be taken when control of instruction execution is passed to the exception handler. That is, the context is saved, the instruction at the appropriate vector offset is fetched, and the exception handler routine is begun in supervisor mode.

Handling — Exception handling is performed by the software linked to the appropriate vector offset. Exception handling is begun in supervisor mode (referred to as privileged state in the architecture specification).

**Note:** The PowerPC Architecture documentation refers to exceptions as interrupts. In this book, the term “interrupt” is reserved to refer to asynchronous exceptions and sometimes to the event that causes the exception. The PowerPC Architecture also uses the word “exception” to refer to IEEE-defined floating-point exception conditions that can cause a program exception to be taken (see *Section 4.5.7, Program Exception (0x00700),* on page 170). The occurrence of these IEEE exceptions might not cause an exception to be taken. IEEE-defined exceptions are referred to as IEEE floating-point exceptions or floating-point exceptions.

## 4.1 PowerPC 750GX Microprocessor Exceptions

As specified by the PowerPC Architecture, exceptions can be either precise or imprecise and either synchronous or asynchronous. Asynchronous exceptions are caused by events external to the processor’s execution; synchronous exceptions are caused by instructions. The types of exceptions are shown in *Table 4-1*.

**Note:** All exceptions except for the system management interrupt, thermal management, and performance-monitor exception are defined, at least to some extent, by the PowerPC Architecture.

*Table 4-1. PowerPC 750GX Microprocessor Exception Classifications*

| Synchronous/Asynchronous | Precise/Imprecise | Exception Types |
|---|---|---|
| Asynchronous, nonmaskable | Imprecise | Machine check, system reset |
| Asynchronous, maskable | Precise | External interrupt, decrementer, system management interrupt, performance-monitor interrupt, thermal-management interrupt |
| Synchronous | Precise | Instruction-caused exceptions |

These classifications are discussed in greater detail in *Section 4.2, Exception Recognition and Priorities,* on page 153. For a better understanding of how the 750GX implements precise exceptions, see Chapter 6, “Exceptions” of the *PowerPC Microprocessor Family: The Programming Environments Manual*. Exceptions implemented in 750GX, and conditions that cause them, are listed in *Table 4-2*.

*Table 4-2. Exceptions and Conditions* (Page 1 of 2)

| Exception Type | Vector Offset (hex) | Causing Conditions |
|---|---|---|
| Reserved | 00000 | — |
| System reset | 00100 | Assertion of either hard reset ($\overline{\text{HRESET}}$) or soft reset ($\overline{\text{SRESET}}$) at power-on reset. |
| Machine check | 00200 | Assertion of transfer error acknowledge ($\overline{\text{TEA}}$) during a data-bus transaction; assertion of machine-check interrupt ($\overline{\text{MCP}}$); an address, data or L2 double-bit error. MSR[ME] must be set. |
| DSI | 00300 | As specified in the PowerPC Architecture, if a page fault occurs. |
| ISI | 00400 | As defined by the PowerPC Architecture, if a page fault occurs. |
| External interrupt | 00500 | MSR[EE] = 1, and interrupt ($\overline{\text{INT}}$) is asserted. |
| Alignment | 00600 | • A floating-point load/store, Store Multiple Word (**stmw**), Store Word Conditional Indexed (**stwcx.**), Load Multiple Word (**lmw**), Load String Word Indexed (**lwarx**), External Control In Word Indexed (**eciwx**), or External Control Out Word Indexed (**ecowx**) instruction operand is not word-aligned.<br>• A multiple/string load/store operation is attempted in little-endian mode.<br>• An operand of a Data Cache Block Set to Zero (**dcbz**) instruction is on a page that is write-through or cache-inhibited for a virtual mode access.<br>• An attempt to execute a **dcbz** instruction occurs when the cache is disabled. |

*Table 4-2. Exceptions and Conditions* (Page 2 of 2)

| Exception Type | Vector Offset (hex) | Causing Conditions |
|---|---|---|
| Program | 00700 | As defined by the PowerPC Architecture (for example, an instruction opcode error). |
| Floating-point unavailable | 00800 | As defined by the PowerPC Architecture. MSR[FP] = 0, and a floating-point instruction is executed. |
| Decrementer | 00900 | As defined by the PowerPC Architecture, when the most-significant bit of the Decrementer Register (DEC) changes from 0 to 1, and MSR[EE] = 1. |
| Reserved | 00A00–00BFF | — |
| System call | 00C00 | Execution of the System Call (**sc**) instruction. |
| Trace | 00D00 | MSR[SE] = 1, or a branch instruction is completing and MSR[BE] = 1. The 750GX differs from the OEA by not taking this exception on an Instruction Synchronize (**isync**) instruction. |
| Reserved | 00E00 | The 750GX does not generate an exception to this vector. Other PowerPC processors might use this vector for floating-point assist exceptions. |
| Reserved | 00E10–00EFF | — |
| Performance monitor | 00F00 | The limit specified in PMC*n* is met and MMCR0[ENINT] = 1 (750GX-specific). |
| Instruction address breakpoint | 01300 | IABR[0–29] matches EA[0–29] of the next instruction to complete, IABR[TE] matches MSR[IR], and IABR[BE] = 1 (750GX-specific). |
| System management exception | 01400 | A system management exception is enabled if MSR[EE] = 1, and is signaled to the 750GX by the assertion of an input signal pin, the system management interrupt ($\overline{\text{SMI}}$). |
| Reserved | 01500–016FF | — |
| Thermal-management interrupt | 01700 | Thermal management is enabled, junction temperature exceeds the threshold specified in THRM1 or THRM2, and MSR[EE] = 1 (750GX-specific). |
| Reserved | 01800–02FFF | — |

## 4.2 Exception Recognition and Priorities

Exceptions are roughly prioritized by exception class, as follows.

1. Nonmaskable, asynchronous exceptions have priority over all other exceptions. These are system reset and machine-check exceptions (although the machine-check exception condition can be disabled so the condition causes the processor to go directly into the checkstop state). These exceptions cannot be delayed and do not wait for completion of any precise-exception handling.
2. Synchronous, precise exceptions are caused by instructions and are taken in strict program order.
3. Imprecise exceptions (imprecise mode floating-point enabled exceptions) are caused by instructions, and they are delayed until higher-priority exceptions are taken. Note that the 750GX does not implement an exception of this type.
4. Maskable asynchronous exceptions (external, decrementer, thermal-management, system-management, performance-monitor, and interrupt exceptions) are delayed if higher-priority exceptions are taken.

The following list of exception categories describes how the 750GX handles exceptions up to the point of signaling the appropriate interrupt to occur. Note that a recoverable state is reached if the completed store queue is empty (drained, not cancelled), and the instruction that is next in program order has been signaled to complete and has completed. If MSR[RI] = 0, the 750GX is in a nonrecoverable state. Also, instruction completion is defined as updating all architectural registers associated with that instruction, and then removing that instruction from the completion buffer.

- Exceptions caused by asynchronous events (interrupts). These exceptions are further distinguished by whether they are maskable and recoverable.
  - Asynchronous, nonmaskable, nonrecoverable
    System reset for assertion of $\overline{\text{HRESET}}$—Has highest priority and is taken immediately regardless of other pending exceptions or recoverability (includes power-on reset).
  - Asynchronous, maskable, nonrecoverable
    Machine-check exception—Has priority over any other pending exception except system reset for assertion of $\overline{\text{HRESET}}$. Taken immediately regardless of recoverability.
  - Asynchronous, nonmaskable, recoverable
    System reset for assertion of $\overline{\text{SRESET}}$—Has priority over any other pending exception except system reset for $\overline{\text{HRESET}}$ (or power-on reset), or machine check. Taken immediately when a recoverable state is reached.
  - Asynchronous, maskable, recoverable
    System management, performance monitor, thermal-management, external, and decrementer interrupts—Before handling this type of exception, the next instruction in program order must complete. If that instruction causes another type of exception, that exception is taken and the asynchronous, maskable recoverable exception remains pending, until the instruction completes. Further instruction completion is halted. The asynchronous, maskable recoverable exception is taken when a recoverable state is reached.
- Instruction-related exceptions. These exceptions are further organized based on the point in instruction processing at which they generate an exception.
  - Instruction fetch
    Instruction storage interrupt (ISI) exceptions—Once this type of exception is detected, dispatching stops, and the current instruction stream is allowed to drain out of the machine. If completing any of the instructions in this stream causes an exception, that exception is taken and the instruction fetch exception is discarded (but might be encountered again when instruction processing resumes). Otherwise, once all pending instructions have executed and a recoverable state is reached, the ISI exception is taken.
  - Instruction dispatch/execution
    Program, data-storage interrupt (DSI), alignment, floating-point unavailable, system call, and instruction address breakpoint—This type of exception is determined during dispatch or execution of an instruction. The exception remains pending until all instructions before the exception-causing instruction in program order complete. The exception is then taken without completing the exception-causing instruction. If completing these previous instructions causes an exception, that exception takes priority over the pending instruction dispatch/execution exception, which is then discarded (but might be encountered again when instruction processing resumes).
  - Post-instruction execution
    Trace—Trace exceptions are generated following execution and completion of an instruction while trace mode is enabled. If executing the instruction produces conditions for another type of exception, that exception is taken and the post-instruction exception is forgotten for that instruction.

**Note:** These exception classifications correspond to how exceptions are prioritized, as described in *Table 4-3, Exception Priorities*, on page 155.

*Table 4-3. Exception Priorities*

| Priority | Exception | Cause |
|---|---|---|
| **Asynchronous Exceptions (Interrupts)** | | |
| 0 | System Reset | $\overline{\text{HRESET}}$, POR |
| 1 | Machine Check | $\overline{\text{TEA}}$, 60x address-parity error, 60x data-parity error, L2 ECC double-bit error, $\overline{\text{MCP}}$, L2-tag parity error, data-tag parity error, instruction-tag parity error, instruction-cache parity error, data-cache parity error, or locked L2 snoop hit[1] |
| 2 | System Reset | $\overline{\text{SRESET}}$ |
| 3 | SMI | $\overline{\text{SMI}}$ (system management exception) |
| 4 | EI | $\overline{\text{INT}}$ (External Exception) |
| 5 | PFM | Performance-monitor exception |
| 6 | DEC | Decrementer exception |
| 7 | TMI | Thermal-management exception |
| **Instruction Fetch Exceptions** | | |
| 0 | ISI | Instruction storage exception |
| **Instruction Dispatch/Execution Exceptions** | | |
| 0 | IABR | Instruction address breakpoint exception |
| 1 | PI | Program exception due to:<br>1. Illegal instruction<br>2. Privileged instruction<br>3. Trap |
| 2 | SC | System call |
| 3 | FPA | Floating-point unavailable exception |
| 4 | PI | Program exception due to floating-point enabled exception |
| 5 | DSI | Data-storage exception due to **eciwx**, **ecowx** with the enable bit of the External Access Register cleared (EAR[E] = 0) (bit 11 of DSISR) |
| 6 | Alignment | Alignment exception due to:<br>• Floating point not word aligned<br>• **lmw**, **stmw**, **lwarx**, or **stwcx** not word-aligned<br>• Either **eciwx** or **ecowx** not word-aligned<br>• Multiple or string access with the little-endian bit set.<br>• **dcbz** to write-through or cache-inhibited page or cache is disabled. |
| 7 | DSI | Data-storage exception due to a block-address-translation (BAT) page-protection violation |
| 8 | DSI | Data-storage exception due to:<br>• Any access except cache operations to a segment where SR[T] = 1<br>• An access that crosses from an SR[T] = 0 segment to an SR[T} = 1 segment<br>These exceptions are indicated by DSISR[5] = 1. |
| 9 | DSI | Data-storage exception due to a translation lookaside buffer (TLB) page-protection violation |
| 10 | DSI | Data-storage exception due to a Data Address Breakpoint Register (DABR) address match |
| **Post Instruction Execution Exceptions** | | |
| 11 | Trace | Trace exception due to:<br>MSR[SE] = 1 or (MSR[BE] = 1 for branches |

[1] Even though DSISR(5) and DSISR(11) are set by different priority exceptions, both bits can be set at the same time.

System reset and machine-check exceptions can occur at any time and are not delayed even if an exception is being handled. As a result, state information for an interrupted exception might be lost. Therefore, these exceptions are typically nonrecoverable. An exception might not be taken immediately when it is recognized.

## 4.3 Exception Processing

When an exception is taken, the processor uses Machine Status Save/Restore Register 0 (SRR0) to determine where instruction processing should resume, and uses Machine Status Save/Restore Register 1 (SRR1) to save the contents of the Machine State Register (MSR) for the current context.

### 4.3.1 Machine Status Save/Restore Register 0 (SRR0)

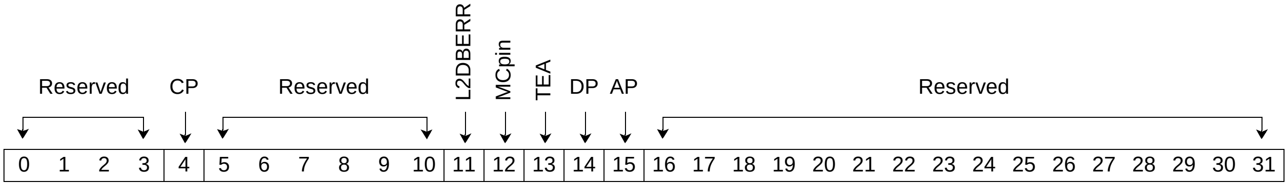
When an exception occurs, the address saved in SRR0 determines where instruction processing should resume when the exception handler returns control to the interrupted process. Depending on the exception, this might be the address in SRR0 or the next address in the program flow. All instructions in the program flow preceding this one will have completed execution, and no subsequent instruction will have begun execution. This might be the address of the instruction that caused the exception or the next one (as in the case of a system call, trace, or trap exception).

| SRR0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |

### 4.3.2 Machine Status Save/Restore Register 1 (SRR1)

SRR1 is used to save machine status (selected MSR bits and possibly other status bits as well) on exceptions and to restore those values when a Return from Interrupt (**rfi**) instruction is executed.

When the 750GX takes a machine-check exception, it will set one or more error bits in SRR1, in Hardware-Implementation-Dependent Register 2 (HID2), or in the L2 Cache Control Register (L2CR). A parity error in either the internal L2 tag array or instruction-cache or data-cache tag arrays is indicated by the CP bit. A data-parity error on the 60x bus is indicated by the DP bit. The MCpin bit indicates that the machine-check pin was activated. The transfer error acknowledge (TEA) bit indicates the machine check was caused by a TEA response on the 60x bus. An address-parity error on the 60x bus will set the AP bit.



| Bits | Field Name | Description |
|---|---|---|
| 0:3 | Reserved | |
| 4 | CP | Set when an internal cache parity error is detected. |
| 5:10 | Reserved | |
| 11 | L2DBERR | Set when an L2 data-cache ECC double-bit error is detected. |
| 12 | MCpin | Set when the machine-check pin is asserted. |
| 13 | TEA | Set when a transfer error acknowledge ($\overline{\text{TEA}}$) error is detected. |
| 14 | DP | Set when a data-bus parity error is detected. |
| 15 | AP | Set when an address-bus parity error is detected. |
| 16:31 | Reserved | |

### 4.3.3 Machine State Register (MSR)

| Reserved | | | | | | | | | | | | | POW | Reserved | ILE | EE | PR | FP | ME | FE0 | SE | BE | FE1 | Reserved | IP | IR | DR | Reserved | PM | RI | LE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |

| Bits | Field Name | Description |
|---|---|---|
| 0:12 | Reserved | Reserved[1]<br>Bits Description<br>0 Full function<br>1:4 Partial function<br>5:9 Full function<br>10:12 Partial function |
| 13 | POW | Power management enable<br>0 Power management disabled (normal operation mode).<br>1 Power management enabled (reduced power mode).<br>Power management functions are implementation-dependent. See *Chapter 10, Power and Thermal Management,* on page 335. |
| 14 | Reserved | Reserved. Implementation-specific |
| 15 | ILE | Exception little-endian mode. When an exception occurs, this bit is copied into MSR[LE] to select the endian mode for the context established by the exception. |
| 16 | EE | External interrupt enable<br>0 The processor delays recognition of external interrupts and decrementer exception conditions.<br>1 The processor is enabled to take an external interrupt or the decrementer exception. |
| 17 | PR | Privilege level<br>0 The processor can execute both user- and supervisor-level instructions.<br>1 The processor can only execute user-level instructions. |
| 18 | FP | Floating-point available<br>0 The processor prevents dispatch of floating-point instructions, including floating-point loads, stores, and moves.<br>1 The processor can execute floating-point instructions and can take floating-point enabled program exceptions. |
| 19 | ME | Machine check enable<br>0 Machine-check exceptions are disabled. If one occurs, the system enters a checkstop.<br>1 Machine-check exceptions are enabled. |
| 20 | FE0 | IEEE floating-point exception mode 0 (see *Table 4-4* on page 160). |
| 21 | SE | Single-step trace enable<br>0 The processor executes instructions normally.<br>1 The processor generates a single-step trace exception upon the successful execution of every instruction except **rfi**, **isync**, and **sc**. Successful execution means that the instruction caused no other exception. |

[1] Full function reserved bits are saved in SRR1 when an exception occurs; they are saved in the same bit locations in SRR1 that they occupy in MSR. Partial function reserved bits are not saved.

| Bits | Field Name | Description |
|---|---|---|
| 22 | BE | Branch trace enable<br>0 The processor executes branch instructions normally.<br>1 The processor generates a branch-type trace exception when a branch instruction executes successfully. |
| 23 | FE1 | IEEE floating-point exception mode 1 (see *Table 4-4* on page 160). |
| 24 | Reserved | Reserved. |
| 25 | IP | Exception prefix. The setting of this bit specifies whether an exception vector offset is prefaced with Fs or 0s. In the following description, *nnnnn* is the offset of the exception.<br>0 Exceptions are vectored to the physical address 0x000*n_nnnn*.<br>1 Exceptions are vectored to the physical address 0xFFF*n_nnnn*. |
| 26 | IR | Instruction address translation<br>0 Instruction address translation is disabled.<br>1 Instruction address translation is enabled.<br>For more information, see *Chapter 5, Memory Management,* on page 179. |
| 27 | DR | Data address translation<br>0 Data address translation is disabled.<br>1 Data address translation is enabled.<br>For more information, see *Chapter 5, Memory Management,* on page 179. |
| 28 | Reserved | Reserved. Full function[1] |
| 29 | PM | Performance-monitor marked mode<br>0 Process is not a marked process.<br>1 Process is a marked process.<br>750GX–specific; defined as reserved by the PowerPC Architecture. For more information about the performance monitor, see *Section 4.5.13, Performance-Monitor Interrupt (0x00F00),* on page 172. |
| 30 | RI | Indicates whether a system reset or machine-check exception is recoverable.<br>0 Exception is not recoverable.<br>1 Exception is recoverable.<br>The RI bit indicates whether, from the perspective of the processor, it is safe to continue (that is, the processor state data such as that saved to SRR0 is valid), but it does not guarantee that the interrupted process is recoverable. Exception handlers must look at bit 30 in SRR1 to determine if the interrupted process is recoverable. |
| 31 | LE | Little-endian mode enable<br>0 The processor runs in big-endian mode.<br>1 The processor runs in little-endian mode. |

[1] Full function reserved bits are saved in SRR1 when an exception occurs; they are saved in the same bit locations in SRR1 that they occupy in MSR. Partial function reserved bits are not saved.

The IEEE floating-point exception mode bits (FE0 and FE1) together define whether floating-point exceptions are handled precisely, imprecisely, or whether they are taken at all. As shown in *Table 4-4*, if either FE0 or FE1 is set, the 750GX treats exceptions as precise. MSR bits are guaranteed to be written to SRR1 when the first instruction of the exception handler is encountered. For further details, see Chapter 6, “Exceptions” of the *PowerPC Microprocessor Family: The Programming Environments Manual*.

*Table 4-4. IEEE Floating-Point Exception Mode Bits*

| FE0 | FE1 | Mode |
|---|---|---|
| 0 | 0 | Floating-point exceptions disabled. |
| 0 | 1 | Imprecise nonrecoverable. For this setting, the 750GX operates in floating-point precise mode. |
| 1 | 0 | Imprecise recoverable. For this setting, the 750GX operates in floating-point precise mode. |
| 1 | 1 | Floating-point precise mode. |

### 4.3.4 Enabling and Disabling Exceptions

When a condition exists that might cause an exception to be generated, it must be determined whether the exception is enabled for that condition.

- IEEE floating-point enabled exceptions (a type of program exception) are ignored when both MSR[FE0] and MSR[FE1] are cleared. If either bit is set, all IEEE enabled floating-point exceptions are taken and cause a program exception.
- Asynchronous, maskable exceptions (such as the external and decrementer interrupts) are enabled by setting MSR[EE]. When MSR[EE] = 0, recognition of these exception conditions is delayed. MSR[EE] is cleared automatically when an exception is taken, to delay recognition of conditions causing those exceptions.
- A machine-check exception can occur only if the machine-check enable bit, MSR[ME], is set. If MSR[ME] is cleared, the processor goes directly into checkstop state when a machine-check exception condition occurs. Individual machine-check exceptions can be enabled and disabled through bits in the HID0 Register, which is described in *Table 4-8* on page 167.
- System reset exceptions cannot be masked.

### 4.3.5 Steps for Exception Processing

After it is determined that the exception can be taken (by confirming that any instruction-caused exceptions occurring earlier in the instruction stream have been handled, and by confirming that the exception is enabled for the exception condition), the processor does the following:

1. SRR0 is loaded with an instruction address that depends on the type of exception. Normally, this is the instruction that would have completed next had the exception not been taken. See the individual exception description for details about how this register is used for specific exceptions.
2. SRR1[1:4, 10:15] are loaded with information specific to the exception type.
3. SRR1[5:9, 16:31] are loaded with a copy of the corresponding MSR bits. Depending on the implementation, reserved bits might not be copied.
4. The MSR is set as described in *Section 4.3.6*. The new values take effect as the first instruction of the exception-handler routine is fetched.
5. Note that MSR[IR] and MSR[DR] are cleared for all exception types. Therefore, address translation is disabled for both instruction fetches and data accesses beginning with the first instruction of the exception-handler routine.
6. Instruction fetch and execution resumes, using the new MSR value, at a location specific to the exception type. The location is determined by adding the exception's vector (see *Table 4-2* on page 152) to the base address determined by MSR[IP]. If IP is cleared, exceptions are vectored to the physical address

0x000*n_nnnn*. If IP is set, exceptions are vectored to the physical address 0xFFF*n_nnnn*. For a machine-check exception that occurs when MSR[ME] = 0 (machine-check exceptions are disabled), the checkstop state is entered (the machine stops executing instructions).

### 4.3.6 Setting MSR[RI]

The RI bit in the MSR was designed to indicate to the exception handler whether the exception is recoverable. When an exception occurs, the RI bit is copied from the MSR to SRR1 and cleared in the MSR. All interrupts are disabled except machine checks. If a machine-check exception occurs while MSR[RI] is clear, a zero value is found in SRR1[RI] to indicate that the machine state is definitely not recoverable. When this bit is a one, the exception is recoverable as far as the current state of the machine and all programs are concerned including noncritical machine checks. An operating system might handle MSR[RI] as follows:

- In all exceptions, if SRR1[RI] is cleared, the machine state is not recoverable. If it is set, the exception is recoverable with respect to the processor and all programs.
- Use the general-purpose SPRs (SPRG0-SPRG3) registers to aid in saving the machine state. The following procedure is suggested:
  - Point SPRG0 to a stack-saved area in memory
  - Save three GPRs in SPRG1-SPRG3.
  - Move SPRG0 into one of the GRPs that was saved. This GPR now points to the save area in memory.
  - Move the GPRs, SRR0, SRR1, SPRG1-3 and other registers to be used by the exception routine into the stack-saved area.
  - Update SPGR0 to point to a new save area.
  - Set MSR[RI] to indicate that machine state has been saved. Also set MSR[EE] if you want to re-enable external interrupts.
- When exception processing is complete, clear MSR[EE] and MSR[RI]. Adjust SPRG0 to point to the stack-saved area, restore the GPRs, SRR0 and SRR1, and any other register that you might have saved. Execute **rfi**. This returns the processor to the interrupted program.

### 4.3.7 Returning from an Exception Handler

The **rfi** instruction performs context synchronization by allowing previously-issued instructions to complete before returning to the interrupted process. In general, execution of the **rfi** instruction ensures the following:

- All previous instructions have completed to a point where they can no longer cause an exception. If a previous instruction causes a direct-store interface error exception, the results must be determined before this instruction is executed.
- Previous instructions complete execution in the context (privilege, protection, and address translation) under which they were issued.
- The **rfi** instruction copies SRR1 bits back into the MSR.
- Instructions fetched after this instruction execute in the context established by this instruction.
- Program execution resumes at the instruction indicated by SRR0

For a complete description of context synchronization, see Chapter 6, “Exceptions,” of the *PowerPC Microprocessor Family: The Programming Environments Manual.*

## 4.4 Process Switching

The following instructions are useful for restoring proper context during process switching:

- The Synchronization (**sync**) instruction orders the effects of instruction execution. All instructions previously initiated appear to have completed before the **sync** instruction completes, and no subsequent instructions appear to be initiated until the **sync** instruction completes. For an example using **sync**, see Chapter 2, “PowerPC Register Set” of the *PowerPC Microprocessor Family: The Programming Environments Manual.*
- The Instruction Synchronization (**isync**) instruction waits for all previous instructions to complete and then discards any fetched instructions, causing subsequent instructions to be fetched (or refetched) from memory and to execute in the context (privilege, translation, and protection) established by the previous instructions.
- The **stwcx.** instruction clears any outstanding reservations, ensuring that an **lwarx** instruction in an old process is not paired with an **stwcx.** instruction in a new one.

The operating system should set MSR[RI] as described in *Section 4.3.6, Setting MSR[RI],*.

## 4.5 Exception Definitions

*Table 4-5* shows all the types of exceptions that can occur with the 750GX and MSR settings when the processor goes into supervisor mode due to an exception. Depending on the exception, certain of these bits are stored in SRR1 when an exception is taken.

*Table 4-5. MSR Setting Due to Exception* (Page 1 of 2)

| Exception Type | MSR Bit[2] | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | POW | ILE | EE | PR | FP | ME | FE0 | SE | BE | FE1 | IP | IR | DR | PM | RI | LE |
| System reset | 0 | — | 0 | 0 | 0 | — | 0 | 0 | 0 | 0 | — | 0 | 0 | 0 | 0 | ILE |
| Machine check | 0 | — | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | — | 0 | 0 | 0 | 0 | ILE |
| DSI | 0 | — | 0 | 0 | 0 | — | 0 | 0 | 0 | 0 | — | 0 | 0 | 0 | 0 | ILE |
| ISI | 0 | — | 0 | 0 | 0 | — | 0 | 0 | 0 | 0 | — | 0 | 0 | 0 | 0 | ILE |
| External interrupt | 0 | — | 0 | 0 | 0 | — | 0 | 0 | 0 | 0 | — | 0 | 0 | 0 | 0 | ILE |
| Alignment | 0 | — | 0 | 0 | 0 | — | 0 | 0 | 0 | 0 | — | 0 | 0 | 0 | 0 | ILE |
| Program | 0 | — | 0 | 0 | 0 | — | 0 | 0 | 0 | 0 | — | 0 | 0 | 0 | 0 | ILE |
| Floating-point unavailable | 0 | — | 0 | 0 | 0 | — | 0 | 0 | 0 | 0 | — | 0 | 0 | 0 | 0 | ILE |
| Decrementer interrupt | 0 | — | 0 | 0 | 0 | — | 0 | 0 | 0 | 0 | — | 0 | 0 | 0 | 0 | ILE |
| System call | 0 | — | 0 | 0 | 0 | — | 0 | 0 | 0 | 0 | — | 0 | 0 | 0 | 0 | ILE |
| Trace exception | 0 | — | 0 | 0 | 0 | — | 0 | 0 | 0 | 0 | — | 0 | 0 | 0 | 0 | ILE |

**Note:**

1. A zero indicates that the bit is cleared.
2. The ILE bit is copied from the MSR[ILE].
3. A dash indicates that the bit is not altered.
4. Reserved bits are read as if written as 0.

*Table 4-5. MSR Setting Due to Exception* (Page 2 of 2)

| Exception Type | MSR Bit[2] | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | POW | ILE | EE | PR | FP | ME | FE0 | SE | BE | FE1 | IP | IR | DR | PM | RI | LE |
| System management | 0 | — | 0 | 0 | 0 | — | 0 | 0 | 0 | 0 | — | 0 | 0 | 0 | 0 | ILE |
| Performance monitor | 0 | — | 0 | 0 | 0 | — | 0 | 0 | 0 | 0 | — | 0 | 0 | 0 | 0 | ILE |
| Thermal management | 0 | — | 0 | 0 | 0 | — | 0 | 0 | 0 | 0 | — | 0 | 0 | 0 | 0 | ILE |

**Note:**
1. A zero indicates that the bit is cleared.
2. The ILE bit is copied from the MSR[ILE].
3. A dash indicates that the bit is not altered.
4. Reserved bits are read as if written as 0.

The setting of the exception prefix bit (IP) determines how exceptions are vectored. If the bit is cleared, exceptions are vectored to the physical address 0x000*n_nnnn* (where *nnnnn* is the vector offset). If IP is set, exceptions are vectored to physical address 0xFFF*n_nnnn*. *Table 4-2* on page 152 shows the exception vector offset of the first instruction of the exception handler routine for each exception type.

### 4.5.1 System Reset Exception (0x00100)

The 750GX implements the system reset exception as defined in the PowerPC Architecture (OEA). The system reset exception is a nonmaskable, asynchronous exception signaled to the processor through the assertion of system-defined signals. In the 750GX, the exception is signaled by the assertion of either the soft reset ($\overline{\text{SRESET}}$) or hard reset ($\overline{\text{HRESET}}$) inputs, described more fully in *Chapter 7, Signal Descriptions,* on page 249

The 750GX implements HID0[NHR], which helps software distinguish a hard reset from a soft reset. Because this bit is cleared by a hard reset, but not by a soft reset, software can set this bit after a hard reset and tell whether a subsequent reset is a hard or soft reset by examining whether this bit is still set.

The first bus operation following the negation of $\overline{\text{HRESET}}$ or the assertion of $\overline{\text{SRESET}}$ will be a single-beat instruction fetch (caching will be inhibited) to x00100.

*Table 4-6* lists register settings when a system reset exception is taken.

*Table 4-6. System Reset Exception–Register Settings*

| Register | Setting Description | | | |
|---|---|---|---|---|
| SRR0 | Set to the effective address of the instruction that the processor would have attempted to execute next if no exception conditions were present. | | | |
| SRR1 | 0 Loaded with equivalent MSR bits<br>1:4 Cleared<br>5:9 Loaded with equivalent MSR bits<br>10:15 Cleared<br>16:31 Loaded with equivalent MSR bits<br>**Note:** If the processor state is corrupted to the extent that execution cannot resume reliably, MSR[RI] (SRR1[30]) is cleared. | | | |
| MSR | POW 0<br>ILE —<br>EE 0<br>PR 0 | FP 0<br>ME —<br>FE0 0<br>SE 0 | BE 0<br>FE1 0<br>IP —<br>IR 0 | DR 0<br>PM 0<br>RI 0<br>LE Set to value of ILE |

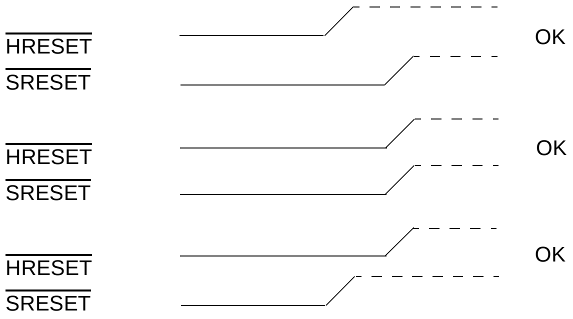
#### 4.5.1.1 Soft Reset

If SRESET is asserted, the processor is first put in a recoverable state. To do this, the 750GX allows any instruction at the point of completion to either complete or take an exception, blocks completion of any subsequent instructions, and allows the completion queue to drain. The state before the exception occurred is then saved as specified in the PowerPC Architecture, and instruction fetching begins at the system reset interrupt vector offset, 0x00100. The vector address on a soft reset depends on the setting of MSR[IP] (either 0x0000_0100 or 0xFFF0_0100). Soft resets are third in priority, after hard resets and machine checks. This exception is recoverable provided attaining a recoverable state does not generate a machine check.

SRESET is an effectively edge-sensitive signal that can be asserted and deasserted asynchronously, provided the minimum pulse width specified in the *PowerPC 750GX RISC Microprocessor Datasheet* is met. Asserting SRESET causes the 750GX to take a system reset exception. This exception modifies the MSR, SRR0, and SRR1, as described in the *PowerPC Microprocessor Family: The Programming Environments Manual*. Unlike a hard reset, a soft reset does not directly affect the states of output signals. Attempts to use SRESET during a hard reset sequence or while the Joint Test Action Group (JTAG) logic is non-idle cause unpredictable results (see *Section 7.2.10.2, Soft Reset (SRESET)—Input,* on page 272 for more information).

SRESET can be asserted during HRESET assertion (see *Figure 4-1*). In all three cases shown in *Table 4-1*, the SRESET assertion and deassertion have no effect on the operation or state of the machine. SRESET asserted coincident to, or after the assertion of, HRESET will also have no effect on the operation or state of the machine.

*Figure 4-1. SRESET Asserted During HRESET*



#### 4.5.1.2 Hard Reset

A hard reset is initiated by asserting HRESET. A hard reset is used primarily for power-on reset (POR) (in which case test reset (TRST) must also be asserted), but it can also be used to restart a running processor. The HRESET signal must be asserted during power up and must remain asserted for a period that allows the phase-locked loop (PLL) to achieve lock and the internal logic to be reset. This period is specified in the *PowerPC 750GX RISC Microprocessor Datasheet*. The 750GX tristates all I/O drivers within five clocks of HRESET assertion. If HRESET is asserted for less than this amount of time, the results are not predictable.

The 750GX’s internal state after the hard reset interval is defined in *Table 4-7*. If HRESET is asserted during normal operation, all operations cease, and the machine state is lost (see *Section 7.2.10.1, Hard Reset (HRESET)—Input,* on page 272 for more information on a hard reset).

The hard reset exception is a nonrecoverable, nonmaskable, asynchronous exception. When $\overline{\text{HRESET}}$ is asserted or at power-on reset (POR), the 750GX immediately branches to 0xFFF0_0100 without attempting to reach a recoverable state. A hard reset has the highest priority of any exception. It is always nonrecoverable.

*Table 4-7* on page 166 shows the state of the machine just before it fetches the first instruction of the system reset handler after a hard reset. In *Table 4-7*, the term “Unknown” means that the content might have been disordered. In particular, the Floating Point Registers (FPRs), BATs, and TLBs might have been disordered. These facilities must be properly initialized before use. To initialize the BATs, first set them all to zero, then to the correct values before any address translation occurs. FPR registers should also be initialized before processing continues.

*Table 4-7. Settings Caused by Hard Reset*

| Register | Setting |
|---|---|
| BATs | Unknown |
| Cache, instruction cache, and data cache | All blocks are unchanged from before HRESET. |
| CR | All zeros |
| CTR | 00000000 |
| DABR | Breakpoint is disabled.<br>Address is unknown. |
| DAR | 00000000 |
| DEC | FFFFFFFF |
| DSISR | 00000000 |
| FPRs | Unknown |
| FPSCR | 00000000 |
| GPRs | Unknown |
| HID0 | 00000000 |
| HID1 | 00000000 |
| IABR | All zeros (break point disabled) |
| ICTC | 00000000 |
| L2CR | 00000000 |
| LR | 00000000 |
| MMCR*n* | 00000000 |
| MSR | 00000040 (only IP set) |
| PMC*n* | Unknown |
| PVR | See the *PowerPC* 750GX *Datasheet* |
| Reservation Address | Unknown (reservation flag -cleared) |
| SDR1 | 00000000 |
| SPRGs | 00000000 |
| SRR0 | 00000000 |
| SRR1 | 00000000 |
| SRs | Unknown |
| Tag directory, instruction cache, and data cache | All entries are marked invalid, all LRU bits are set to zero, and caches are disabled. |
| TBL | 00000000 |
| TBU | 00000000 |
| THRM*n* | 00000000 |
| TLBs | Unknown |
| UMMCR*n* | 00000000 |
| UPMC*n* | 00000000 |
| USIA | 00000000 |
| XER | 00000000 |

The following is also true after a hard reset operation:

- External checkstops are enabled.
- The on-chip test interface has given control of the I/Os to the rest of the chip for functional use.
- Since the reset exception has data and instruction translation disabled (MSR[DR] and MSR[IR] both cleared), the chip operates in direct address-translation mode (referred to as the real-addressing mode in the architecture specification).
- Time from HRESET deassertion until the 750GX asserts the first transfer start (TS) (bus parked on the 750GX) or BG is 8 to 12 bus clocks (SYSCLK).

### 4.5.2 Machine-Check Exception (0x00200)

The 750GX implements the machine-check exception as defined in the PowerPC Architecture (OEA). It conditionally initiates a machine-check exception after an address or data-parity error occurred on the bus or in either the L1 or L2 cache, after receiving a qualified transfer error acknowledge (TEA) indication on the 750GX bus, or after the machine-check interrupt (MCP) signal had been asserted. As defined in the OEA, the exception is not taken if MSR[ME] is cleared, in which case the processor enters the checkstop state.

Certain machine-check conditions can be enabled and disabled using HID0 bits, as described in *Table 4-8*.

*Table 4-8. HID0 Machine-Check Enable Bits*

| Bits | Field Name | Description |
|---|---|---|
| 0 | EMCP | Enable MCP. The primary purpose of this bit is to mask out further machine-check exceptions caused by assertion of MCP, similar to how MSR[EE] can mask external interrupts.<br>0 Masks MCP. Asserting MCP does not generate a machine-check exception or a checkstop.<br>1 Asserting MCP causes a checkstop if MSR[ME] = 0 or a machine-check exception if MSR[ME] = 1. |
| 1 | DBP | Disable 60x bus address-parity and data-parity generation.<br>0 Parity generation is enabled.<br>1 Disable parity generation. If the system does not use address or data parity and the respective parity checking is disabled (HID0[EBA] or HID0[EBD] = 0), input receivers for those signals are disabled, do not require pull-up resistors, and therefore should be left unconnected. If all parity generation is disabled, all parity checking should also be disabled and parity signals need not be connected. |
| 2 | EBA | Enable/disable 60x bus address-parity checking.<br>0 Prevents address-parity checking.<br>1 Allows an address-parity error to cause a checkstop if MSR[ME] = 0 or a machine-check exception if MSR[ME] = 1.<br>EBA and EBD allow the processor to operate with memory subsystems that do not generate parity. |
| 3 | EBD | Enable 60x bus data-parity checking.<br>0 Parity checking is disabled.<br>1 Allows a data-parity error to cause a checkstop if MSR[ME] = 0 or a machine-check exception if MSR[ME] = 1.<br>EBA and EBD allow the processor to operate with memory subsystems that do not generate parity. |
| 15 | NHR | Not hard reset (software use only)<br>0 A hard reset occurred if software previously set this bit<br>1 A hard reset has not occurred. |

A $\overline{\text{TEA}}$ indication on the bus can result from any load or store operation initiated by the processor. In general, $\overline{\text{TEA}}$ is expected to be used by a memory controller to indicate that a memory parity error or an uncorrectable memory ECC error has occurred. Note that the resulting machine-check exception is imprecise and unordered with respect to the instruction that originated the bus operation.

If MSR[ME] and the appropriate HID0 bits are set, the exception is recognized and handled; otherwise, the processor generates an internal checkstop condition. When the exception is recognized, all incomplete stores are discarded. The bus protocol operates normally.

A machine-check exception might result from referencing a nonexistent physical address, either directly (with MSR[DR] = 0) or through an invalid translation. If a **dcbz** instruction introduces a block into the cache associated with a nonexistent physical address, a machine-check exception can be delayed until an attempt is made to store that block to main memory. Not all PowerPC processors provide the same level of error checking. Checkstop sources are implementation-dependent.

Machine-check exceptions are enabled when MSR[ME] = 1; this is described in the next section. If MSR[ME] = 0 and a machine check occurs, the processor enters the checkstop state. The checkstop state is described in *Section 4.5.2.2, Checkstop State (MSR[ME] = 0),* on page 169.

### *4.5.2.1 Machine-Check Exception Enabled (MSR[ME] = 1)*

Machine-check exceptions are enabled when MSR[ME] = 1. When a machine-check exception is taken, registers are updated as shown in *Table 4-9*.

*Table 4-9. Machine-Check Exception—Register Settings*

| Register | Setting Description |
|---|---|
| SRR0 | On a best-effort basis, the 750GX can set this to an EA of some instruction that was executing or about to be executing when the machine-check condition occurred. |
| SRR1 | 0:10 Cleared.<br>11 Set when an L2 data-cache ECC double-bit error is detected; otherwise, zero.<br>12 Set when an $\overline{\text{MCP}}$ signal is asserted; otherwise, zero.<br>13 Set when a $\overline{\text{TEA}}$ signal is asserted; otherwise, zero.<br>14 Set when a data-bus parity error is detected; otherwise, zero.<br>15 Set when an address-bus parity error is detected; otherwise, zero.<br>16:31 MSR[16–31]. |
| MSR | POW 0 FP 0 BE 0 DR 0<br>ILE — ME 0 FE1 0 PM 0<br>EE 0 FE0 0 IP — RI 0<br>PR 0 SE 0 IR 0 LE Set to value of ILE |
| **Note:** To handle another machine-check exception, the exception handler should set MSR[ME] as soon as it is practical after a machine-check exception is taken. Otherwise, subsequent machine-check exceptions cause the processor to enter the checkstop state. | |

The machine-check exception is usually unrecoverable in the sense that execution cannot resume in the context that existed before the exception (see *Section 4.3.6, Setting MSR[RI],*). If the condition that caused the machine check does not otherwise prevent continued execution, MSR[ME] is set to allow the processor to continue execution at the machine-check exception vector address and prevent the processor from entering checkstop state if another machine check occurs. Typically, earlier processes cannot resume. However, operating systems can use the machine-check exception handler to try to identify and log the cause of the machine-check condition.

When a machine-check exception is taken, instruction fetching resumes at offset 0x00200 from the physical base address indicated by MSR[IP].

#### *4.5.2.2 Checkstop State (MSR[ME] = 0)*

If MSR[ME] = 0 and a machine check occurs, the processor enters the checkstop state. The 750GX processor can also be forced into the checkstop state by the assertion of checkstop input (CKSTP_IN), the primary input signal.

When a processor is in checkstop state, instruction processing is suspended and generally cannot resume without the processor being reset. The contents of all latches are frozen within two cycles upon entering checkstop state.

### 4.5.3 DSI Exception (0x00300)

A DSI exception occurs when no higher-priority exception exists and an error condition related to a data memory access occurs. The DSI exception is implemented as it is defined in the PowerPC Architecture (OEA). In case of a TLB miss for a load, store, or cache operation, a DSI exception is taken if the resulting hardware table search causes a page fault.

On the 750GX, a DSI exception is taken when a load or store is attempted to a direct-store segment (SR[T] = 1). In the 750GX, a floating-point load or store to a direct-store segment causes a DSI exception rather than an alignment exception, as specified by the PowerPC Architecture.

The 750GX also implements the data address breakpoint facility, which is defined as optional in the PowerPC Architecture and is supported by the optional Data Address Breakpoint Register (DABR). Although the architecture does not strictly prescribe how this facility must be implemented, the 750GX follows the recommendations provided by the architecture and described in the Chapter 2, “Programming Model” and Chapter 6, “Exceptions” in the *PowerPC Microprocessor Family: The Programming Environments Manual*.

### 4.5.4 ISI Exception (0x00400)

An ISI exception occurs when no higher-priority exception exists and an attempt to fetch the next instruction fails. This exception is implemented as it is defined by the PowerPC Architecture (OEA), and is taken for the following conditions:

- The effective address cannot be translated.
- The fetch access is to a no-execute segment (SR[N] = 1).
- The fetch access is to guarded storage and MSR[IR] = 1.
- The fetch access is to a segment for which SR[T] is set.
- The fetch access violates memory protection.

When an ISI exception is taken, instruction fetching resumes at offset 0x00400 from the physical base address indicated by MSR[IP].

### 4.5.5 External Interrupt Exception (0x00500)

An external interrupt is signaled to the processor by the assertion of the external interrupt signal (INT). The INT signal is expected to remain asserted until the 750GX takes the external interrupt exception. If INT is negated early, recognition of the interrupt request is not guaranteed. After the 750GX begins execution of the external interrupt handler, the system can safely negate the INT. When the 750GX detects assertion of INT, it stops dispatching and waits for all pending instructions to complete. This allows any instructions in progress that need to take an exception to do so before the external interrupt is taken. After all instructions have vacated the completion buffer, the 750GX takes the external interrupt exception as defined in the PowerPC Architecture (OEA).

An external interrupt might be delayed by other higher-priority exceptions or if MSR[EE] is cleared when the exception occurs. Register settings for this exception are described in Chapter 6, “Exceptions” in the *PowerPC Microprocessor Family: The Programming Environments Manual.*

When an external interrupt exception is taken, instruction fetching resumes at offset 0x00500 from the physical base address indicated by MSR[IP].

### 4.5.6 Alignment Exception (0x00600)

The 750GX implements the alignment exception as defined by the PowerPC Architecture (OEA). An alignment exception is initiated when any of the following occurs:

- The operand of a floating-point load or store is not word-aligned.
- The operand of **lmw**, **stmw**, **lwarx**, or **stwcx.** is not word-aligned.
- The operand of **dcbz** is in a page which is write-through or caching-inhibited.
- An attempt is made to execute **dcbz** when the data cache is disabled.
- An **eciwx** or **ecowx** is not word-aligned.
- A multiple or string access is attempted with MSR[LE] set.

**Note:** In the 750GX, a floating-point load or store to a direct-store segment causes a DSI exception rather than an alignment exception, as specified by the PowerPC Architecture. For more information, see *Section 4.5.3, DSI Exception (0x00300),* on page 169.

### 4.5.7 Program Exception (0x00700)

The 750GX implements the program exception as it is defined by the PowerPC Architecture (OEA). A program exception occurs when no higher-priority exception exists and one or more of the exception conditions defined in the OEA occur.

The 750GX invokes the system illegal instruction program exception when it detects any instruction from the illegal instruction class. The 750GX fully decodes the Special Purpose Register (SPR) field of the instruction. If an undefined SPR is specified, a program exception is taken.

The UISA defines **mtspr** and **mfspr** with the record bit (Rc) set as causing a program exception or giving a boundedly-undefined result (see *Section 2.3.1.1, Definition of Boundedly Undefined,* on page 87 for more information). In the 750GX, the appropriate Condition Register (CR) should be treated as undefined. Likewise, the PowerPC Architecture states that the Floating Compared Unordered (**fcmpu**) or Floating Compared Ordered (**fcmpo**) instruction with the record bit set can either cause a program exception or provide a boundedly-undefined result. In the 750GX, the condition register field destination (the BF field) in an instruction encoding for these cases is considered undefined.

The 750GX does not support either of the two floating-point imprecise modes defined by the PowerPC Architecture. Unless exceptions are disabled (MSR[FE0] = MSR[FE1] = 0), all floating-point exceptions are treated as precise.

When a program exception is taken, instruction fetching resumes at offset 0x00700 from the physical base address indicated by MSR[IP]. Chapter 6, “Exceptions” in the *PowerPC Microprocessor Family: The Programming Environments Manual* describes register settings for this exception.

### 4.5.8 Floating-Point Unavailable Exception (0x00800)

The floating-point unavailable exception is implemented as defined in the PowerPC Architecture. A floating-point unavailable exception occurs when no higher-priority exception exists, an attempt is made to execute a floating-point instruction (including floating-point load, store, or move instructions), and the floating-point available bit in the MSR is disabled, (MSR[FP] = 0). Register settings for this exception are described in Chapter 6, “Exceptions” in the *PowerPC Microprocessor Family: The Programming Environments Manual.*

When a floating-point unavailable exception is taken, instruction fetching resumes at offset 0x00800 from the physical base address indicated by MSR[IP].

### 4.5.9 Decrementer Exception (0x00900)

The decrementer exception is implemented in the 750GX as it is defined by the PowerPC Architecture. The decrementer exception occurs when no higher-priority exception exists, a decrementer exception condition occurs (for example, the Decrementer Register has completed decrementing), and MSR[EE] = 1. In the 750GX, the Decrementer Register is decremented at one fourth the bus clock rate. Register settings for this exception are described in Chapter 6, “Exceptions” in the *PowerPC Microprocessor Family: The Programming Environments Manual.*

When a decrementer exception is taken, instruction fetching resumes at offset 0x00900 from the physical base address indicated by MSR[IP].

### 4.5.10 System Call Exception (0x00C00)

A system-call exception occurs when a System Call (**sc**) instruction is executed. In the 750GX, the system call exception is implemented as it is defined in the PowerPC Architecture. Register settings for this exception are described in Chapter 6, “Exceptions” in the *PowerPC Microprocessor Family: The Programming Environments Manual.*

When a system call exception is taken, instruction fetching resumes at offset 0x00C00 from the physical base address indicated by MSR[IP].

### 4.5.11 Trace Exception (0x00D00)

The trace exception is taken if MSR[SE] = 1 or if MSR[BE] = 1 and the currently completing instruction is a branch. Each instruction considered during trace mode completes before a trace exception is taken.

**Implementation Note:** The 750GX processor diverges from the PowerPC Architecture in that it does not take trace exceptions on the **isync** instruction.

When a trace exception is taken, instruction fetching resumes at offset 0x00D00 from the base address indicated by MSR[IP].

### 4.5.12 Floating-Point Assist Exception (0x00E00)

The optional floating-point assist exception defined by the PowerPC Architecture is not implemented in the 750GX.

### 4.5.13 Performance-Monitor Interrupt (0x00F00)

The 750GX microprocessor provides a performance-monitor facility to monitor and count predefined events such as processor clocks, misses in either the instruction cache or the data cache, instructions dispatched to a particular execution unit, mispredicted branches, and other occurrences. The count of such events can be used to trigger the performance-monitor exception. The performance-monitor facility is not defined by the PowerPC Architecture.

The performance monitor can be used for the following situations:

- To increase system performance with efficient software, especially in a multiprocessing system. Memory hierarchy behavior must be monitored and studied to develop algorithms that schedule tasks (and perhaps partition them) and that structure and distribute data optimally.
- To help system developers bring up and debug their systems.

The performance monitor uses the following SPRs.

- The Performance-Monitor Counter Registers (PMC1–PMC4) are used to record the number of times a certain event has occurred. UPMC1–UPMC4 provide user-level read access to these registers.
- The Monitor Mode Control Registers (MMCR0–MMCR1) are used to enable various performance-monitor interrupt functions. UMMCR0–UMMCR1 provide user-level read access to these registers.
- The Sampled Instruction Address Register (SIA) contains the effective address of an instruction executing at or around the time that the processor signals the performance-monitor interrupt condition. The USIA register provides user-level read access to the SIA.

*Table 4-10* lists register settings when a performance-monitor interrupt exception is taken.

*Table 4-10. Performance-Monitor Interrupt Exception—Register Settings*

| Register | Setting Description |
|---|---|
| SRR0 | Set to the effective address of the instruction that the processor would have attempted to execute next if no exception conditions were present. |
| SRR1 | 0 Loaded with equivalent MSR bits.<br>1:4 Cleared.<br>5:9 Loaded with equivalent MSR bits.<br>10:15 Cleared.<br>16:31 Loaded with equivalent MSR bits. |
| MSR | POW 0 FP 0 BE 0 DR 0<br>ILE — ME — FE1 0 PM 0<br>EE 0 FE0 0 IP — RI 0<br>PR 0 SE 0 IR 0 LE Set to value of ILE |

As with other PowerPC exceptions, the performance-monitor interrupt follows the normal PowerPC exception model with a defined exception vector offset (0x00F00). The priority of the performance-monitor interrupt lies between the external interrupt and the decrementer interrupt (see *Table 4-3* on page 155). The contents of the SIA are described in *Sampled Instruction Address Register (SIA)* on page 75. The performance monitor is described in *Chapter 11, Performance Monitor and System Related Features,* on page 349.

### 4.5.14 Instruction Address Breakpoint Exception (0x01300)

An instruction address breakpoint interrupt occurs when the following conditions are met:

- The instruction breakpoint address IABR[0:29] matches EA[0:29] of the next instruction to complete in program order. The instruction that triggers the instruction address breakpoint exception is not executed before the exception handler is invoked.
- The translation enable bit (IABR[TE]) matches MSR[IR].
- The breakpoint enable bit (IABR[BE]) is set. The address match is also reported to the JTAG/common on-chip processor (COP) block, which can subsequently generate a soft or hard reset. The instruction tagged with the match does not complete before the breakpoint exception is taken.

See *Section 2.1.2.1, Instruction Address Breakpoint Register (IABR),* on page 64 for the format of the IABR.

*Table 4-11* lists register settings when an instruction address breakpoint exception is taken.

*Table 4-11. Instruction Address Breakpoint Exception—Register Settings*

| Register | Setting Description |
|---|---|
| SRR0 | Set to the effective address of the instruction that the processor would have attempted to execute next if no exception conditions were present. |
| SRR1 | 0 Loaded with equivalent MSR bits.<br>1:4 Cleared.<br>5:9 Loaded with equivalent MSR bits.<br>10:15 Cleared.<br>16:31 Loaded with equivalent MSR bits. |
| MSR | POW 0 FP 0 BE 0 DR 0<br>ILE — ME — FE1 0 PM 0<br>EE 0 FE0 0 IP — RI 0<br>PR 0 SE 0 IR 0 LE Set to value of ILE |

The 750GX requires that an **mtspr** to the IABR be followed by a context-synchronizing instruction. The 750GX cannot generate a breakpoint response for that context-synchronizing instruction if the breakpoint is enabled by the **mtspr** instruction to the **IABR** immediately preceding it. The 750GX also cannot block a breakpoint response on the context-synchronizing instruction if the breakpoint was disabled by the **mtspr** instruction to the **IABR** immediately preceding it.

When an instruction address breakpoint exception is taken, instruction fetching resumes at offset 0x01300 from the base address indicated by MSR[IP].

### 4.5.15 System Management Interrupt (0x01400)

The 750GX implements a system management interrupt exception, which is not defined by the PowerPC Architecture. The system management exception is very similar to the external interrupt exception and is particularly useful in implementing the nap mode. It has priority over an external interrupt (see *Table 4-3* on page 155), and it uses a different vector in the exception table (offset 0x01400).

*Table 4-12* lists register settings when a system management interrupt exception is taken.

*Table 4-12. System Management Interrupt Exception—Register Settings*

| Register | Setting Description |
|---|---|
| SRR0 | Set to the effective address of the instruction that the processor would have attempted to execute next if no exception conditions were present. |
| SRR1 | 0 Loaded with equivalent MSR bits.<br>1:4 Cleared.<br>5:9 Loaded with equivalent MSR bits.<br>10:15 Cleared.<br>16:31 Loaded with equivalent MSR bits. |
| MSR | POW 0<br>ILE —<br>EE 0<br>PR 0<br>FP 0<br>ME —<br>FE0 0<br>SE 0<br>BE 0<br>FE1 0<br>IP —<br>IR 0<br>DR 0<br>PM 0<br>RI 0<br>LE Set to value of ILE |

Like the external interrupt, a system management interrupt is signaled to the 750GX by the assertion of an input signal. The system management interrupt signal (SMI) is expected to remain asserted until the interrupt is taken. If SMI is negated early, recognition of the interrupt request is not guaranteed. After the 750GX begins execution of the system management interrupt handler, the system can safely negate SMI. After the assertion of SMI is detected, the 750GX stops dispatching instructions and waits for all pending instructions to complete. This allows any instructions in progress that need to take an exception to do so before the system management interrupt is taken.

When a system management interrupt exception is taken, instruction fetching resumes at offset 0x01400 from the base address indicated by MSR[IP].

### 4.5.16 Thermal-Management Interrupt Exception (0x01700)

A thermal-management interrupt is generated when the junction temperature crosses a threshold programmed in either THRM1 or THRM2. The exception is enabled by the thermal-management interrupt enable (TIE) bit of either THRM1 or THRM2, and can be masked by setting MSR[EE].

*Table 4-13* lists register settings when a thermal-management interrupt exception is taken.

*Table 4-13. Thermal-Management Interrupt Exception—Register Settings*

| Register | Setting Description |
|---|---|
| SRR0 | Set to the effective address of the instruction that the processor would have attempted to execute next if no exception conditions were present. |
| SRR1 | 0 Loaded with equivalent MSR bits<br>1:4 Cleared<br>5:9 Loaded with equivalent MSR bits<br>10:15 Cleared<br>16:31 Loaded with equivalent MSR bits |
| MSR | POW 0<br>ILE —<br>EE 0<br>PR 0<br>FP 0<br>ME —<br>FE0 0<br>SE 0<br>BE 0<br>FE1 0<br>IP —<br>IR 0<br>DR 0<br>PM 0<br>RI 0<br>LE Set to value of ILE |

The thermal-management interrupt is similar to the system management and external interrupt. The 750GX requires the next instruction in program order to complete or take an exception, blocks completion of any following instructions, and allows the completed store queue to drain. Any exceptions encountered in this process are taken first, and the thermal-management interrupt exception is delayed until a recoverable halt is achieved, at which point the 750GX saves the machine state, as shown in *Table 4-13*. When a thermal-management interrupt exception is taken, instruction fetching resumes at offset 0x01700 from the base address indicated by MSR[IP].

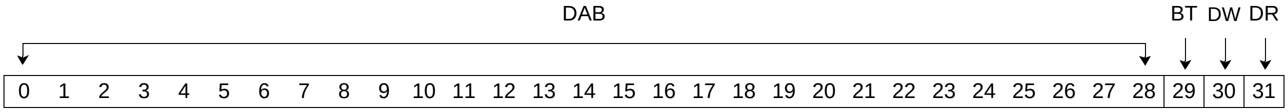
*Chapter 10, Power and Thermal Management,* on page 335 gives the details about thermal management.

### 4.5.17 Data Address Breakpoint Exception

The Data Address Breakpoint Register (DABR) is a Special Purpose Register that can cause a data-storage exception (DSI). When enabled, data addresses are compared with an effective address that is stored in the DABR (bits 0:28). The granularity of these compares is a double-word. Bit 29 is the translation enable bit and is compared with the MSR[DR] bit. Bit 30 is a store enable. Bit 31 is a load enable. The DABR is enabled by setting either the data store enable (DW) or data read enabled (DR) bit. The format of the DABR register is shown in *Section 4.5.17.1* on page 175.

#### *4.5.17.1 Data Address Breakpoint Register (DABR)*

For a full description of this register, see the *PowerPC Microprocessor Family: The Programming Environments Manual*.



| Bits | Field Name | Description |
|---|---|---|
| 0:28 | DAB | Double-word address to be compared. |
| 29 | BT | Translation enabled. |
| 30 | DW | Data store enabled. |
| 31 | DR | Data read enabled. |

### 4.5.18 Soft Stops

Both trace and breakpoint exception conditions will generate a soft stop instead of an exception if soft stop has been enabled by the JTAG/COP logic. If trace and instruction breakpoint conditions occur simultaneously, instruction breakpoint takes priority over trace in both the exception and soft stop enabled cases.

A soft stop can also be generated with a request from the COP. This request is treated like an external exception, except that it is nonmaskable and generates a soft stop instead of an exception.

If soft stop is enabled, only one soft stop will be generated before completion of an instruction with an IABR match. This holds true if a soft stop is generated before that instruction for any other reason, such as trace mode on for the preceding instruction or a COP soft stop request.

### 4.5.19 Exception Latencies

Latencies for taking various exceptions are variable based on the state of the machine when conditions to produce an exception occur. The shortest latency possible is one cycle. In this case, an exception is signaled in the cycle following the appearance of the conditions that generated that exception. In most cases, a hard reset or machine check has a single-cycle latency to exception. The only situation that can prevent this is when a speculative instruction is the next to complete. This case, which produces an extra 2-cycle minimum, 3-cycle maximum delay, only occurs if the branch guess that forced this instruction to be speculative was resolved to be incorrect.

Another latency variable is introduced for a soft reset exception–recoverability. The time to reach a recoverable state can depend on the time needed to complete or to cause an exception to an instruction at the point of completion, the time needed to drain the completed store queue, or the time waiting for a correct empty state so that a valid exception prefix (IP) can be saved. For other externally-generated exceptions, a further delay might be incurred waiting for another exception, generated while reaching a recoverable state, to be serviced.

Further delays are possible for other types of exceptions depending on the number and type of instructions that must be completed before that exception can be serviced. See *Section 4.5.20, Summary of Front-End Exception Handling,* to determine possible maximum latencies for different exceptions.

### 4.5.20 Summary of Front-End Exception Handling

*Table 4-14* describes how the 750GX handles exceptions up to the point of signaling the appropriate exception to occur. Note that a recoverable state is reached in the 750GX if the completed store queue is empty (drained, not canceled), and the instruction that is next in the program order has been signaled to complete and has completed. If MSR[RI] = 0, the 750GX is in a nonrecoverable state by default. Also, completion of an instruction is defined as performing all architectural register writes associated with that instruction, and then removing that instruction from the completion buffer queue.

*Table 4-14. Front-End Exception Handling Summary* (Page 1 of 2)

| Exception Type | Specific Exception | Description |
|---|---|---|
| Asynchronous Nonmaskable Nonrecoverable | System Reset for $\overline{\text{HRESET}}$ | Has highest priority and is taken immediately regardless of other pending exceptions or recoverability. A nonspeculative address is guaranteed. |
| Asynchronous Maskable Nonrecoverable | Machine Check | Takes priority over any other pending exception except system reset for HRESET or POR. Taken immediately, regardless of recoverability. A nonspeculative address is guaranteed. |
| Asynchronous Nonmaskable Recoverable | System Reset for $\overline{\text{SRESET}}$ | Takes priority over any other pending exception except system reset for HRESET or POR or machine check. Taken immediately when a recoverable state is reached. |
| Asynchronous Maskable Recoverable | SMI, EI, DEC | Before handling this type of exception, the next instruction in program order must complete or cause an exception. If this action causes another type of exception, that exception is taken and the asynchronous maskable recoverable (AMR) exception remains pending. Once an instruction is able to complete without causing an exception, while the AMR exception is enabled, further instruction completion is halted. The AMR exception is then taken once a recoverable state is reached. |

*Table 4-14. Front-End Exception Handling Summary* (Page 2 of 2)

| Exception Type | Specific Exception | Description |
|---|---|---|
| Instruction Fetch | ISI | Once this type of exception is detected, dispatch is halted and the current instruction stream is allowed to drain out of the machine. If completing any of the instructions in this stream causes an exception, that exception is taken and the instruction fetch exception is forgotten. Otherwise, once the machine is empty and a recoverable state is reached, the instruction fetch exception is taken. |
| Instruction Dispatch/Execution | Program, DSI, Alignment, FPA, SC, IABR, DABR | This type of exception is determined at dispatch or execution of an instruction. The exception remains pending until all instructions in program order before the exception-causing instruction are completed. The exception is then taken without completing the exception-causing instruction. If any other exception condition is created in completing these previous instructions in the machine, that exception takes priority over the pending Instruction Dispatch/Execution exception, which is then forgotten. |
| Post Instruction Execution | Trace | This type of exception is generated following execution and completion of an instruction while a trace mode is enabled. If executing the instruction produces conditions for another type of exception, that exception is taken and the Post Instruction Execution exception is forgotten for that instruction. |

### 4.5.21 Timer Facilities

At power-on reset (POR), the 750GX initializes the Time Base and Decrementer Registers to the following values:

- Time Base Upper Register (TBU) = 0x00000000
- Time Base Lower Register (TBL) = 0x00000000
- Decrementer Register (DEC) = 0xFFFFFFFF

### 4.5.22 External Access Instructions

The 750GX implements the **eciwx** and **ecowx** instructions. Executing these instructions while MSR[DR] = 0 is considered a programming error, and the physical address on the bus is undefined. Executing these instructions to a direct-store (T = 1) segment causes a data-storage exception (DSI).

The 750GX implements the External Access Register (EAR) to support the external access instructions. Bit 0 implements the Enable bit. Bits 1 to 25 are reserved. Bits 26 and 27 are not implemented and are reserved. Bits 28 to 31 are the implemented bits of the Resource ID (RID).

# 5. Memory Management

This chapter describes the 750GX microprocessor’s implementation of the memory management unit (MMU) specifications provided by the operating environment architecture (OEA) for PowerPC processors. The primary function of the MMU in a PowerPC processor is the translation of logical (effective) addresses to physical addresses (referred to as real addresses in the architecture specification) for memory accesses and I/O accesses (I/O accesses are assumed to be memory-mapped). In addition, the MMU provides access protection on a segment, block, or page basis. This chapter describes the specific hardware used to implement the MMU model of the OEA in the 750GX. See Chapter 7, “Memory Management,” in the *PowerPC Microprocessor Family: The Programming Environments Manual* for a complete description of the conceptual model. Note that the 750GX does not implement the optional direct-store facility, and it is not likely to be supported in future devices.

Two general types of memory accesses generated by PowerPC processors require address translation—instruction accesses and data accesses generated by load-and-store instructions. Generally, the address-translation mechanism is defined in terms of the segment descriptors and page tables that PowerPC processors use to locate the effective-to-physical address mapping for memory accesses. The segment information translates the effective address to an interim virtual address, and the page table information translates the interim virtual address to a physical address.

The segment descriptors, used to generate the interim virtual addresses, are stored as on-chip segment registers on 32-bit implementations (such as the 750GX). In addition, two translation lookaside buffers (TLBs) are implemented on the 750GX to keep recently-used page-address translations on-chip. Although the PowerPC OEA describes one MMU (conceptually), the 750GX hardware maintains separate TLBs and table-search resources for instruction and data accesses that can be performed independently (and simultaneously). Therefore, the 750GX is described as having two MMUs, one for instruction accesses (IMMU) and one for data accesses (DMMU).

The block-address translation (BAT) mechanism is a software-controlled array that stores 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). There are separate instruction and data BAT mechanisms,. In the 750GX, they reside in the instruction and data MMUs, respectively.

The MMUs, together with the exception processing mechanism, provide the necessary support for the operating system to implement a virtual memory environment and for enforcing protection of designated memory areas.

Exception processing is described in *Chapter 4, Exceptions,* on page 151. Specifically, *Section 4.3, Exception Processing,* on page 156 describes the Machine State Register (MSR), which controls some of the critical functionality of the MMUs.

## 5.1 MMU Overview

The 750GX implements the memory-management specification of the PowerPC OEA for 32-bit implementations. Thus, it provides four gigabytes of effective address space accessible to supervisor and user programs, with a 4-KB page size and 256-MB segment size. In addition, the MMUs of 32-bit PowerPC processors use an interim virtual address (52 bits) and hashed page tables in the generation of 32-bit physical addresses. PowerPC processors also have a BAT mechanism for mapping large blocks of memory. Block sizes range from 128 KB to 256 MB and are software-programmable.

Basic features of the 750GX MMU implementation defined by the OEA are as follows:

- Support for real-addressing mode—Effective-to-physical address translation can be disabled separately for data and instruction accesses.
- Block-address translation—Each of the BAT array entries (eight IBAT entries and eight DBAT entries) provides a mechanism for translating blocks as large as 256 MB 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.
- Segmented address translation—The 32-bit effective address is extended to a 52-bit virtual address by substituting 24 bits of upper address bits from the segment register, for the 4 upper bits of the effective address (EA), which are used as an index into the segment register file. This 52-bit virtual address space is divided into 4-KB pages, each of which can be mapped to a physical page.

The 750GX also provides the following features that are not required by the PowerPC Architecture:

- Separate translation lookaside buffers (TLBs)—The 128-entry, 2-way set-associative instruction TLBs (ITLBs) and data TLBs (DTLBs) keep recently-used page-address translations on-chip.
- Table-search operations performed in hardware—The 52-bit virtual address is formed and the MMU attempts to fetch the page table entry (PTE), which contains the physical address, from the appropriate TLB on-chip. If the translation is not found in a TLB (that is, a TLB miss occurs), the hardware performs a table-search operation (using a hashing function) to search for the PTE.
- TLB invalidation— The 750GX implements the optional TLB Invalidate Entry (**tlbie**) and TLB Synchronize (**tlbsync**) instructions, which can be used to invalidate TLB entries. For more information on the **tlbie** and **tlbsync** instructions, see *Section 5.4.3.2, TLB Invalidation,* on page 201.

*Figure 5-1* summarizes the 750GX MMU features, including those defined by the PowerPC Architecture (OEA) for 32-bit processors and those specific to the 750GX.

*Table 5-1. MMU Feature Summary* (Page 1 of 2)

| Feature Category | Architecturally Defined/ 750GX-Specific | Feature |
|---|---|---|
| Address ranges | Architecturally defined | $2^{32}$ bytes of effective address |
| | | $2^{52}$ bytes of virtual address |
| | | $2^{32}$ bytes of physical address |
| Page size | Architecturally defined | 4 KB |
| Segment size | Architecturally defined | 256 MB |
| Block-address translation | Architecturally defined | Range of 128 KB–256 MB sizes |
| | | Implemented with IBAT and DBAT registers in BAT array |
| Memory protection | Architecturally defined | Segments selectable as no-execute |
| | | Pages selectable as user/supervisor and read-only or guarded |
| | | Blocks selectable as user/supervisor and read-only or guarded |
| Page history | Architecturally defined | Referenced and changed bits defined and maintained |
| Page-address translation | Architecturally defined | Translations stored as PTEs in hashed page tables in memory |
| | | Page table size determined by mask in SDR1 register |

*Table 5-1. MMU Feature Summary* (Page 2 of 2)

| Feature Category | Architecturally Defined/ 750GX-Specific | Feature |
|---|---|---|
| TLBs | Architecturally defined | Instructions for maintaining TLBs (**tlbie** and **tlbsync** instructions in the 750GX) |
| | 750GX-specific | 128-entry, 2-way set-associative ITLB<br>128-entry, 2-way set-associative DTLB<br>Least recently used (LRU) replacement algorithm |
| Segment descriptors | Architecturally defined | Stored as segment registers on-chip (two identical copies maintained) |
| Page table-search support | 750GX-specific | The 750GX performs the table-search operation in hardware. |

### 5.1.1 Memory Addressing

A program references memory using the effective (logical) address computed by the processor when it executes a load, store, branch, or cache instruction, and when it fetches the next instruction. The effective address is translated to a physical address according to the procedures described in Chapter 7, “Memory Management” in the *PowerPC Microprocessor Family: The Programming Environments Manual*, augmented with information in this section. The memory subsystem uses the physical address for the access.

For a discussion of effective address calculation, see *Section 2.3.2.3* on page 90.

### 5.1.2 MMU Organization

*Figure 5-1, MMU Conceptual Block Diagram,* on page 183 shows the conceptual organization of a PowerPC MMU in a 32-bit implementation. However, it does not describe the specific hardware used to implement the memory-management function for a particular processor. Processors might optionally implement on-chip TLBs, hardware support for the automatic search of the page tables for PTEs, and other hardware features (invisible to the system software) that are not shown.

The 750GX maintains two on-chip TLBs with the following characteristics:

- 128 entries, 2-way set associative (64 × 2), LRU replacement
- Data TLB supports the DMMU; instruction TLB supports the IMMU
- Hardware TLB update
- Hardware update of referenced (R) and changed (C) bits in the translation table

In the event of a TLB miss, the hardware attempts to load the TLB based on the results of a translation table-search operation.

*Figure 5-2, PowerPC 750GX Microprocessor IMMU Block Diagram,* on page 184 and *Figure 5-3, 750GX Microprocessor DMMU Block Diagram,* on page 185 show the conceptual organization of the 750GX’s instruction and data MMUs, respectively. The instruction addresses shown in *Figure 5-2* are generated by the processor for sequential instruction fetches and addresses that correspond to a change of program flow. The data addresses shown in *Figure 5-3* are generated by load, store, and cache instructions.

As shown in the figures, after an address is generated, the high-order bits of the effective address, EA[0–19] (or a smaller set of address bits, EA[0–*n*], in the cases of blocks), are translated into physical address bits PA[0–19]. The low-order address bits, A[20–31], are untranslated and are therefore identical for both effective and physical addresses. After translating the address, the MMUs pass the resulting 32-bit physical address to the memory subsystem. The MMUs record whether the translation is for an instruction or data access, whether the processor is in user or supervisor mode, and for data accesses, whether the access is a load or a store operation.

The MMUs use this information to appropriately direct the address translation and to enforce the protection hierarchy programmed by the operating system. (*Section 4.3, Exception Processing,* on page 156 describes the MSR, which controls some of the critical functionality of the MMUs.)

The figures show how address bits A[20–26] index into the on-chip instruction and data caches to select a cache set. The remaining physical address bits are then compared with the tag fields (comprised of bits PA[0–19]) of the eight selected cache blocks to determine if a cache hit has occurred. In the case of a cache miss on the 750GX, the instruction or data access is then forwarded to the L2 tags to check for an L2 cache hit. In case of a miss, the access is forwarded to the bus interface unit, which initiates an external memory access.

*Figure 5-1. MMU Conceptual Block Diagram*

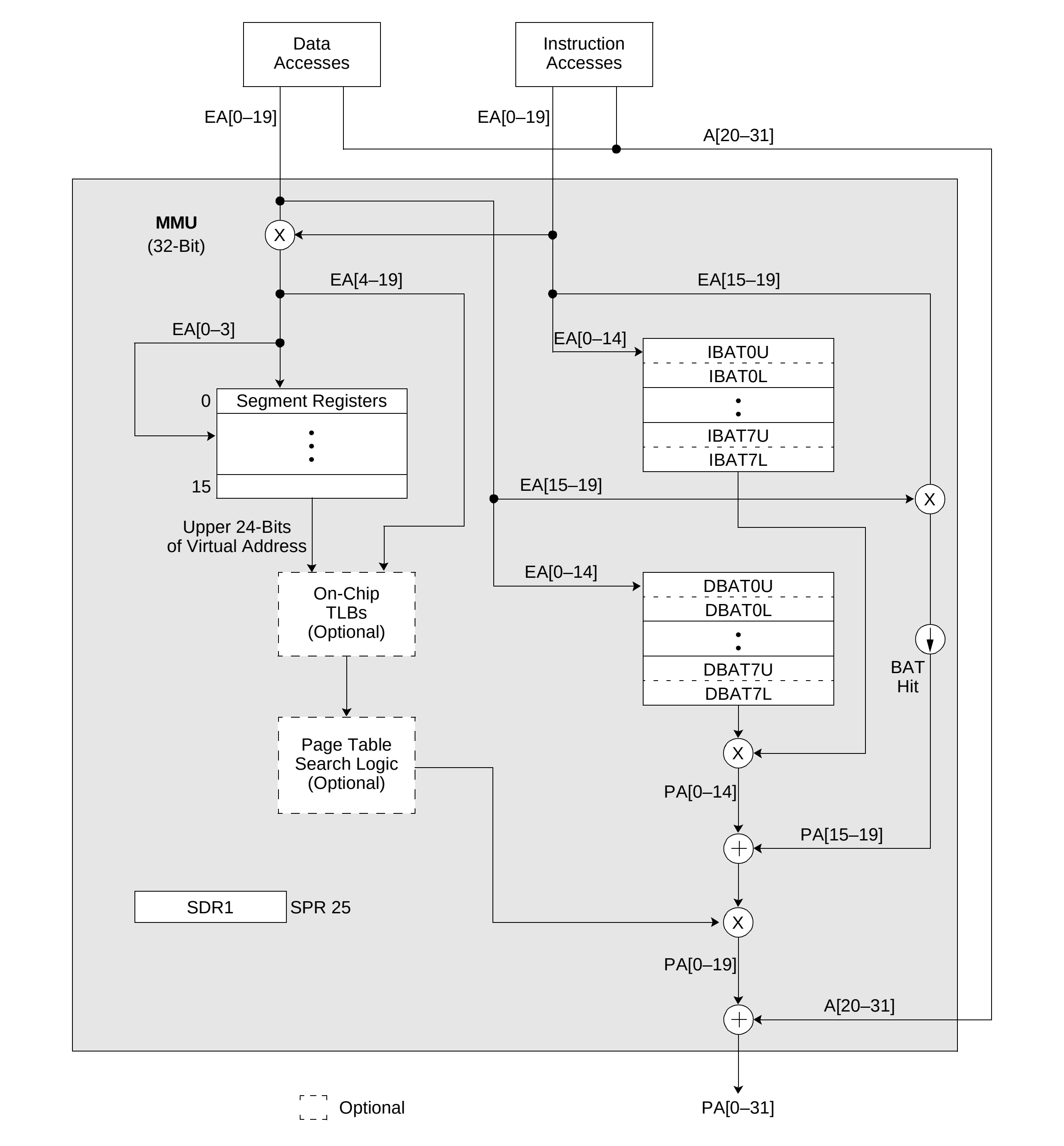


*Figure 5-2. PowerPC 750GX Microprocessor IMMU Block Diagram*

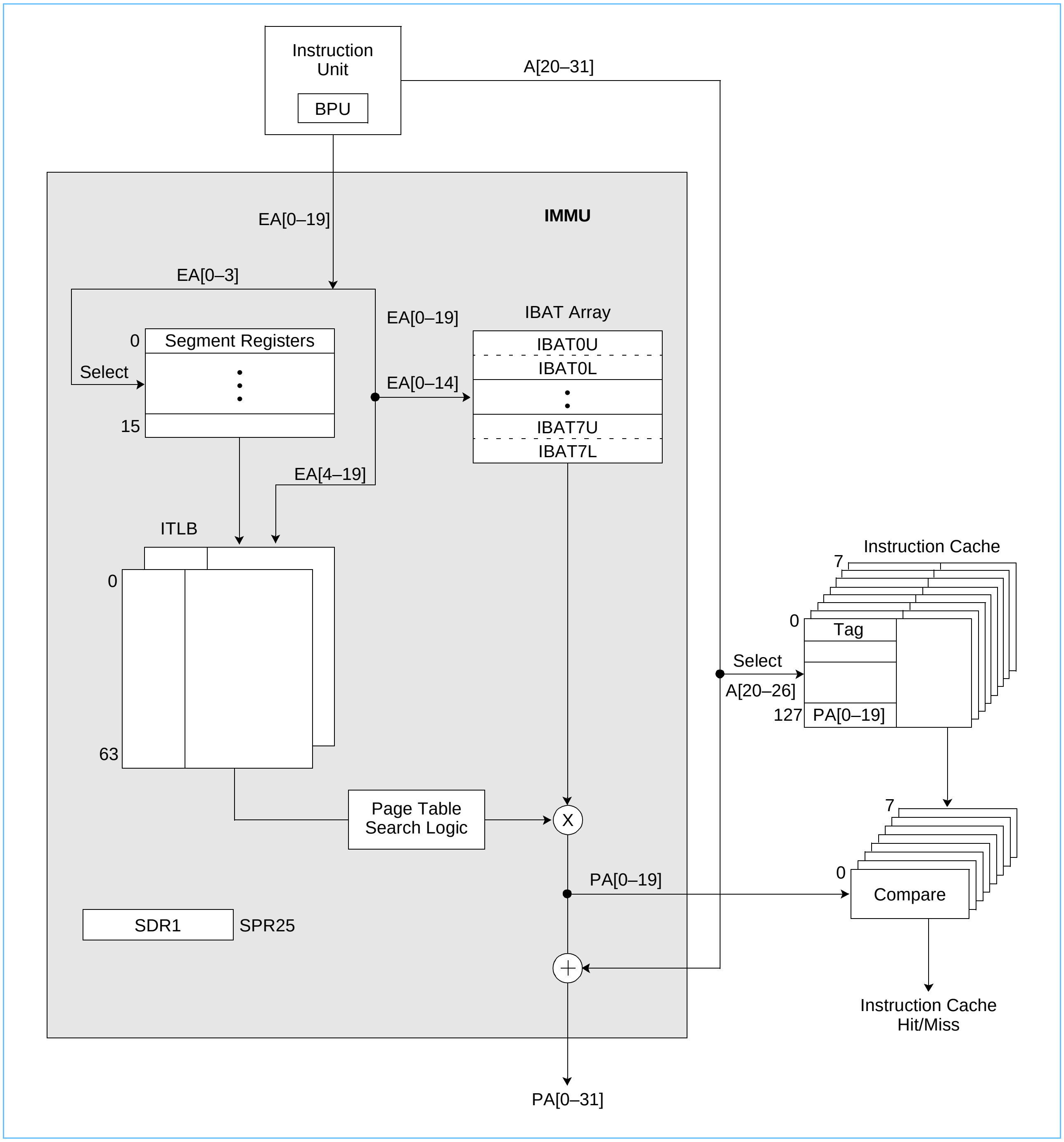
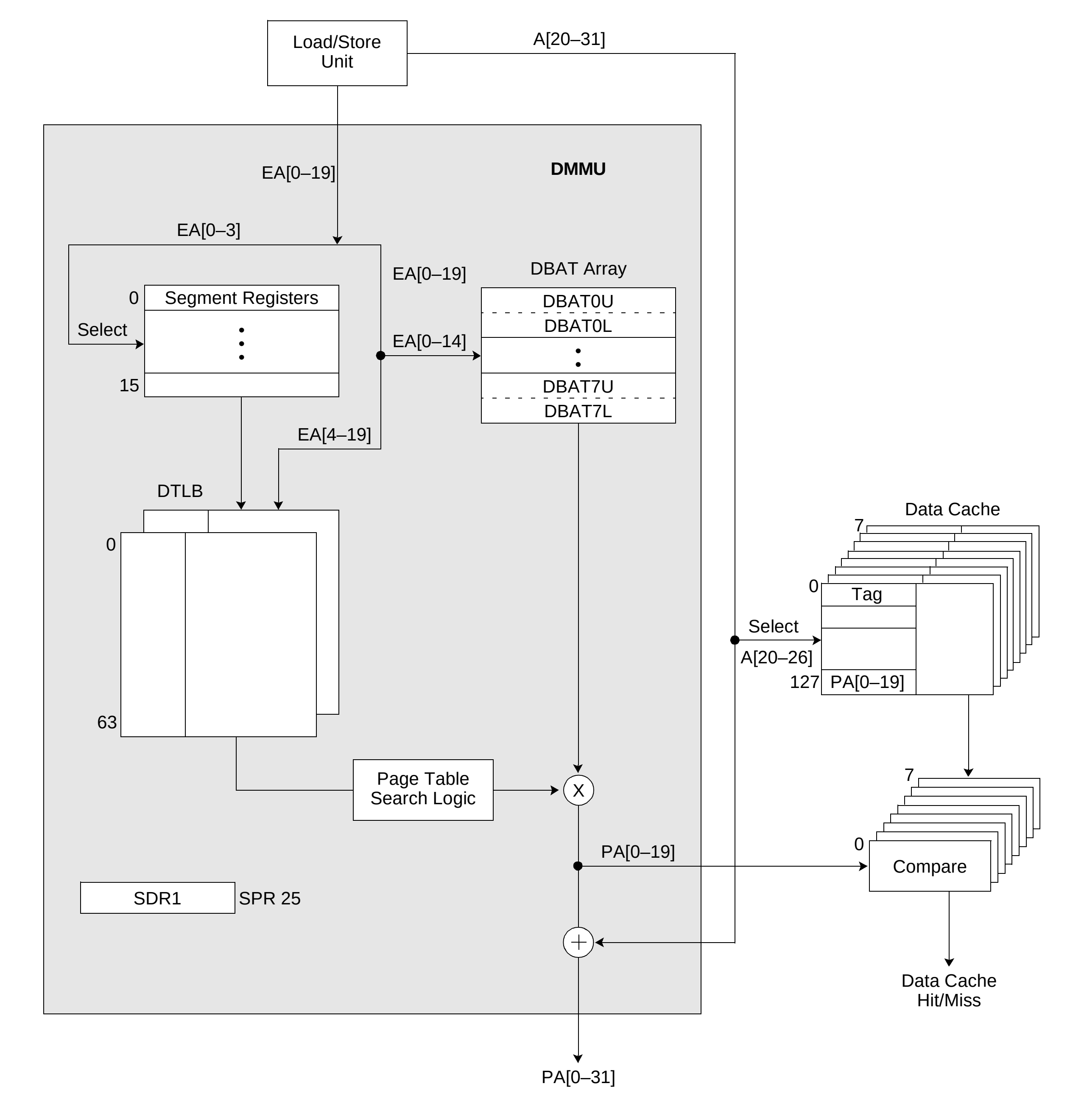


*Figure 5-3. 750GX Microprocessor DMMU Block Diagram*



### 5.1.3 Address-Translation Mechanisms

PowerPC processors support the following three types of address translation:

Page address — Translates the page frame address for a 4-KB page size.

Block address — Translates the block number for blocks that range in size from 128 KB to 256 MB.

Real-addressing mode address — When address translation is disabled, the physical address is identical to the effective address.

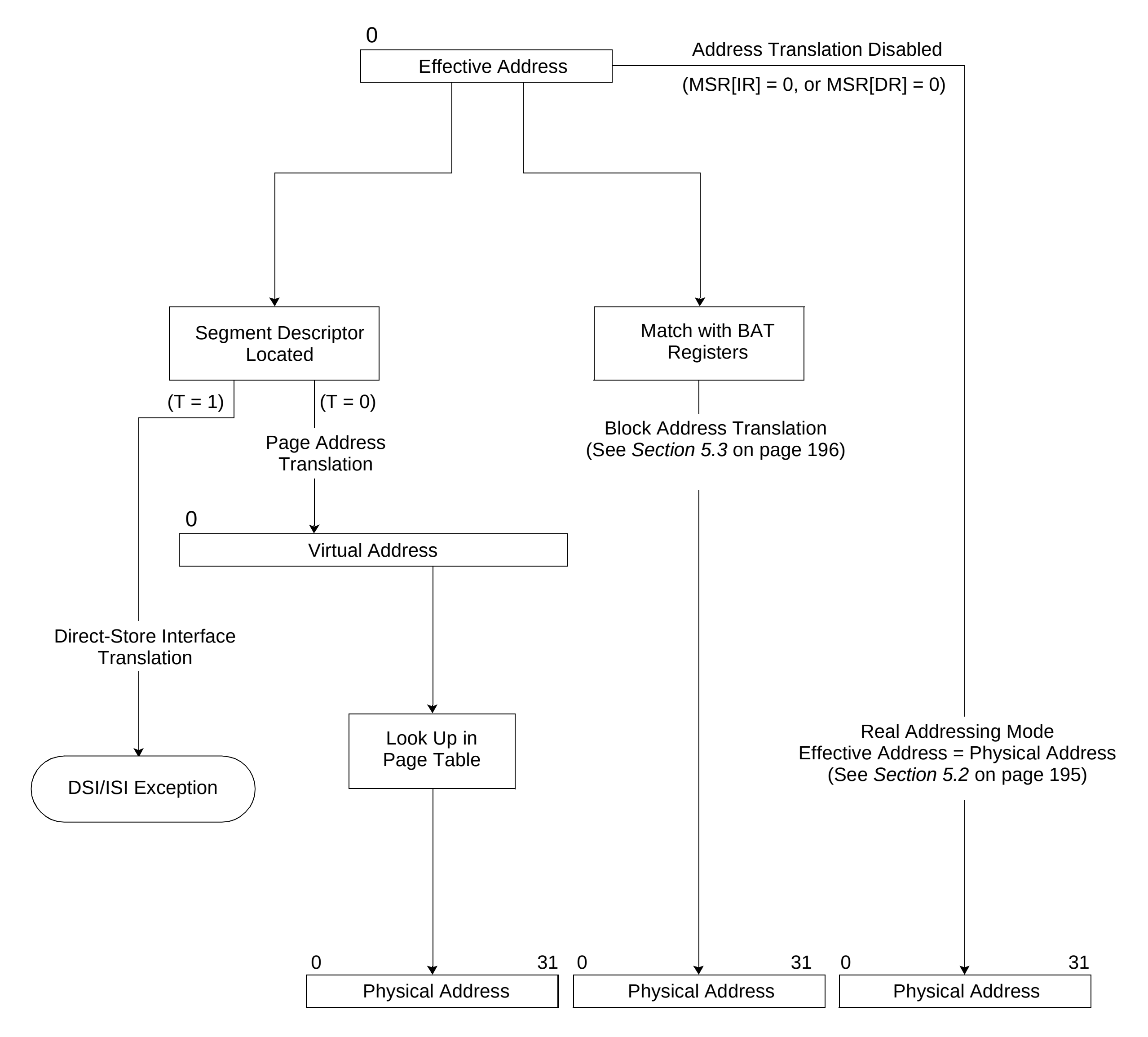
*Figure 5-4, Address-Translation Types* shows the three address-translation mechanisms provided by the MMUs. The segment descriptors shown in the figure control the page-address-translation mechanism. When an access uses page-address translation, the appropriate segment descriptor is required. In 32-bit implementations, the appropriate segment descriptor is selected from the 16 on-chip segment registers by the 4 highest-order effective address bits.

A control bit in the corresponding segment descriptor then determines if the access is to memory (memory-mapped) or to the direct-store interface space. Note that the direct-store interface was present in the architecture only for compatibility with existing I/O devices that use this interface. However, it is being removed from the architecture, and the 750GX does not support it. When an access is determined to be to the direct-store interface space, the 750GX takes a data-storage interrupt (DSI) exception if it is a data access (see *Section 4.5.3, DSI Exception (0x00300),* on page 169), and takes an instruction storage interrupt (ISI) exception if it is an instruction access (see *Section 4.5.4, ISI Exception (0x00400),* on page 169).

For memory accesses translated by a segment descriptor, the interim virtual address is generated using the information in the segment descriptor. Page-address translation corresponds to the conversion of this virtual address into the 32-bit physical address used by the memory subsystem. In most cases, the physical address for the page resides in an on-chip TLB and is available for quick access. However, if the page-address translation misses in the on-chip TLB, the MMU causes a search of the page tables in memory (using the virtual address information and a hashing function) to locate the required physical address.

Because blocks are larger than pages, there are fewer upper-order effective address bits to be translated into physical address bits (more low-order address bits (at least 17) are untranslated to form the offset into a block) for block-address translation. Also, instead of segment descriptors and a TLB, block-address translations use the on-chip BAT registers as a BAT array. If an effective address matches the corresponding field of a BAT register, the information in the BAT register is used to generate the physical address. In this case, the results of the page translation (occurring in parallel) are ignored.

*Figure 5-4. Address-Translation Types*


0
Effective Address
Address Translation Disabled
(MSR[IR] = 0, or MSR[DR] = 0)
Segment Descriptor Located
Match with BAT Registers
(T = 1)
(T = 0)
Page Address Translation
Block Address Translation
(See *Section 5.3* on page 196)
0
Virtual Address
Direct-Store Interface Translation
DSI/ISI Exception
Look Up in Page Table
Real Addressing Mode
Effective Address = Physical Address
(See *Section 5.2* on page 195)
0
31
Physical Address
0
31
Physical Address
0
31
Physical Address

When the processor generates an access, and the corresponding address-translation-enable bit in the MSR is cleared, the resulting physical address is identical to the effective address, and all other translation mechanisms are ignored. Instruction address translation and data address translation are enabled by setting MSR[IR] and MSR[DR], respectively.

### 5.1.4 Memory-Protection Facilities

In addition to the translation of effective addresses to physical addresses, the MMUs provide access protection of supervisor areas from user access and can designate areas of memory as read-only, as well as no-execute or guarded. *Table 5-2* on page 188 shows the protection options supported by the MMUs for pages.

*Table 5-2. Access Protection Options for Pages*

| Option | User Read | | User Write | Supervisor Read | | Supervisor Write |
|---|---|---|---|---|---|---|
| | I-Fetch | Data | | I-Fetch | Data | |
| Supervisor-only | V | V | V | A | A | A |
| Supervisor-only-no-execute | V | V | V | V | A | A |
| Supervisor-write-only | A | A | V | A | A | A |
| Supervisor-write-only-no-execute | V | A | V | V | A | A |
| Both (user/supervisor) | A | A | A | A | A | A |
| Both (user-/supervisor) no-execute | V | A | A | V | A | A |
| Both (user-/supervisor) read-only | A | A | V | A | A | V |
| Both (user/supervisor) read-only-no-execute | V | A | V | V | A | V |

A Access permitted
V Protection violation

The no-execute option provided in the segment register lets the operating system program determine whether instructions can be fetched from an area of memory. The remaining options are enforced based on a combination of information in the segment descriptor and the page table entry. Thus, the supervisor-only option allows only read and write operations generated while the processor is operating in supervisor mode (MSR[PR] = 0) to access the page. User accesses that map into a supervisor-only page cause an exception.

Finally, a facility in the virtual environment architecture (VEA) and the operating environment architecture (OEA) allows pages or blocks to be designated as guarded, preventing out-of-order accesses that might cause undesired side effects. For example, areas of the memory map used to control I/O devices can be marked as guarded so accesses do not occur unless they are explicitly required by the program.

For more information on memory protection, see “Memory Protection Facilities,” in Chapter 7, “Memory Management,” in the *PowerPC Microprocessor Family: The Programming Environments Manual*.

### 5.1.5 Page History Information

The MMUs of PowerPC processors also define referenced (R) and changed (C) bits in the page-address-translation mechanism that can be used as history information relevant to the page. The operating system can use these bits to determine which areas of memory to write back to disk when new pages must be allocated in main memory. While these bits are initially programmed by the operating system into the page table, the architecture specifies that they can be maintained either by the processor hardware (automatically) or by some software-assist mechanism.

**Implementation Note:** When loading the TLB, the 750GX checks the state of the changed and referenced bits for the matched PTE. If the referenced bit is not set and the table-search operation is initially caused by a load operation or by an instruction fetch, then the 750GX automatically sets the referenced bit in the translation table. Similarly, if the table-search operation is caused by a store operation and either the referenced bit or the changed bit is not set, then the hardware automatically sets both bits in the translation table. In addition, when the address translation of a store operation hits in the DTLB, the 750GX checks the state of the changed bit. If the bit is not already set, the hardware automatically updates the DTLB and the translation table in memory to set the changed bit. For more information, see *Section 5.4.1, Page History Recording,* on page 196.

### 5.1.6 General Flow of MMU Address Translation

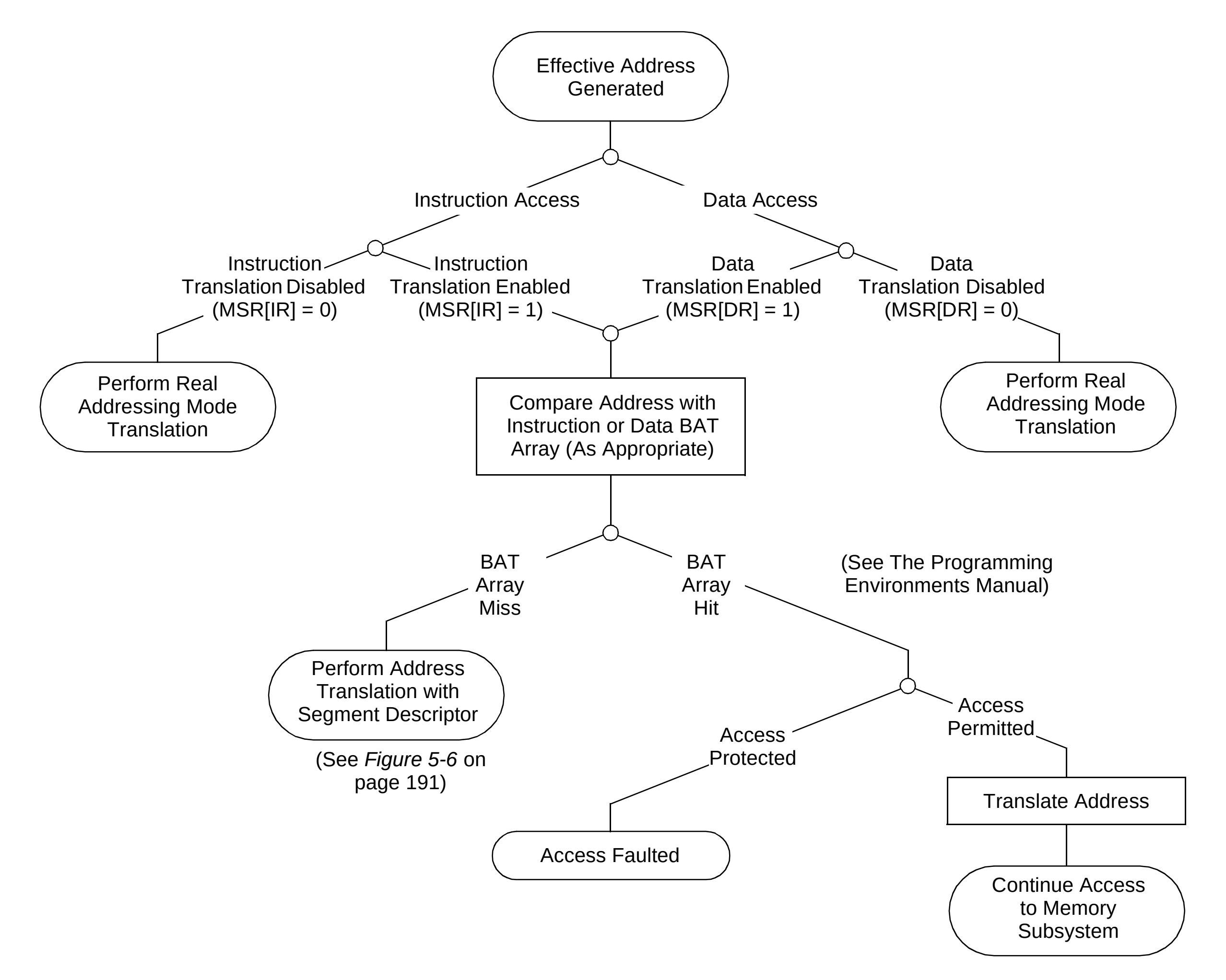
The following sections describe the general flow used by PowerPC processors to translate effective addresses to virtual and then physical addresses.

#### *5.1.6.1 Real-Addressing Mode and Block-Address-Translation Selection*

When an instruction or data access is generated and the corresponding instruction or data translation is disabled (MSR[IR] = 0 or MSR[DR] = 0), then the real-addressing mode is used (physical address equals effective address), and the access continues to the memory subsystem as described in *Section 5.2, Real-Addressing Mode,* on page 195.

*Figure 5-5* shows the flow the MMUs use in determining whether to select real-addressing mode, block-address translation, or the segment descriptor to select page-address translation.

*Figure 5-5. General Flow of Address Translation (Real-Addressing Mode and Block)*


Effective Address Generated
Instruction Access
Data Access
Instruction Translation Disabled (MSR[IR] = 0)
Instruction Translation Enabled (MSR[IR] = 1)
Data Translation Enabled (MSR[DR] = 1)
Data Translation Disabled (MSR[DR] = 0)
Perform Real Addressing Mode Translation
Compare Address with Instruction or Data BAT Array (As Appropriate)
Perform Real Addressing Mode Translation
BAT Array Miss
BAT Array Hit
(See The Programming Environments Manual)
Perform Address Translation with Segment Descriptor
(See Figure 5-6 on page 191)
Access Protected
Access Permitted
Translate Address
Access Faulted
Continue Access to Memory Subsystem

**Note:** If the BAT array search results in a hit, then the access is qualified with the appropriate protection bits. If the access violates the protection mechanism, then an exception (either ISI or DSI) is generated.

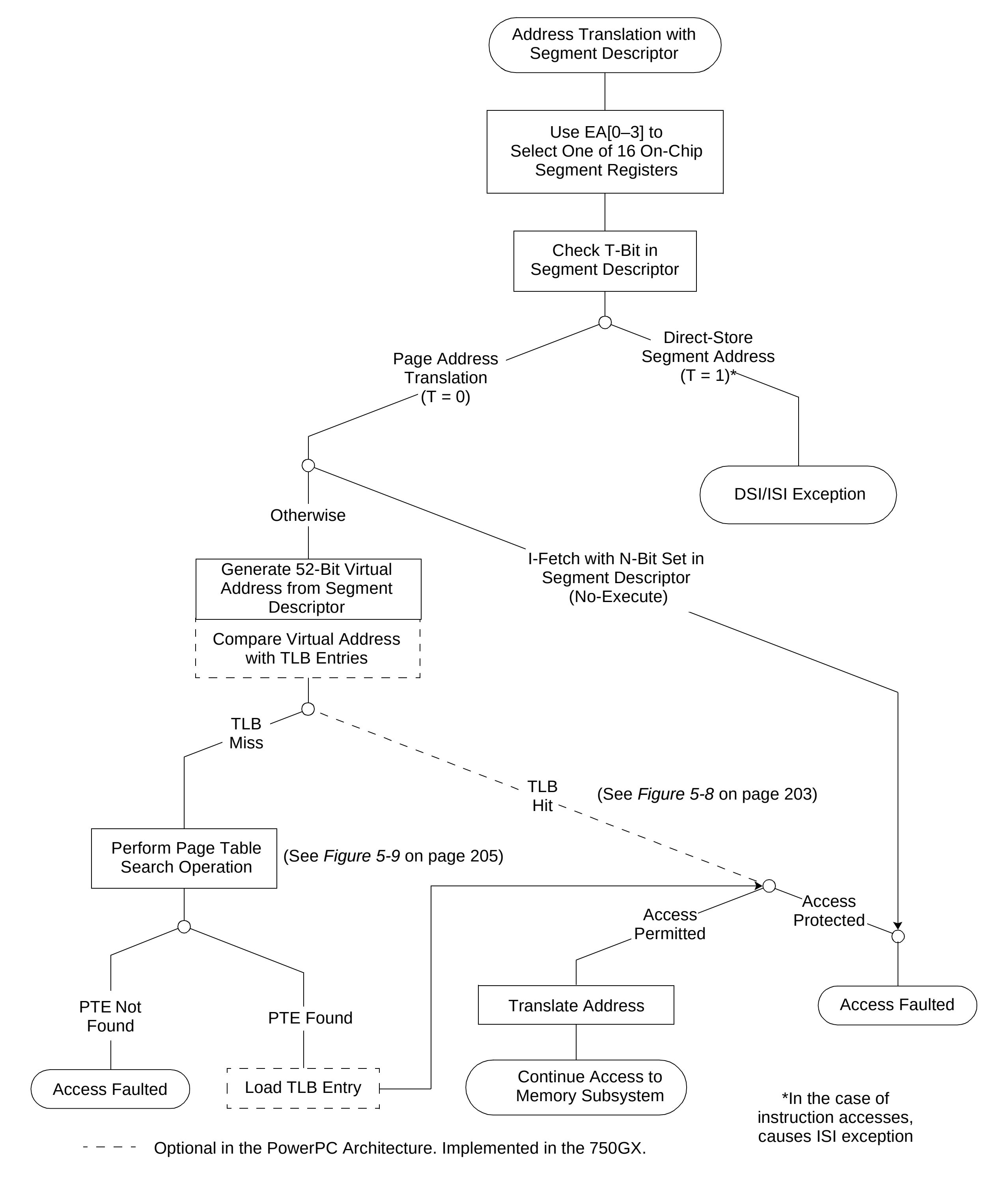
### *5.1.6.2 Page-Address-Translation Selection*

If address translation is enabled and the effective address information does not match a BAT array entry, then the segment descriptor must be located. When the segment descriptor is located, the T bit in the segment descriptor selects whether the translation is to a page or to a direct-store segment as shown in *Figure 5-6, General Flow of Page and Direct-Store Interface Address Translation,* on page 191.

For 32-bit implementations, the segment descriptor for an access is contained in one of the 16 on-chip Segment Registers. Effective address bits EA[0–3] select one of the 16 Segment Registers.

**Note:** The 750GX does not implement the direct-store interface, and accesses to these segments cause a DSI or ISI exception. In addition, *Figure 5-6* shows how the no-execute protection is enforced. If the no-execute (N) bit in the segment descriptor is set and the access is an instruction fetch, the access is faulted as described in Chapter 7, “Memory Management,” in the *PowerPC Microprocessor Family: The Programming Environments Manual*. *Figure 5-6* shows the flow for these cases as described by the PowerPC OEA, and so the TLB references are shown as optional. Because the 750GX implements TLBs, these branches are valid and are described in more detail throughout this section.

*Figure 5-6. General Flow of Page and Direct-Store Interface Address Translation*


Address Translation with Segment Descriptor
Use EA[0–3] to Select One of 16 On-Chip Segment Registers
Check T-Bit in Segment Descriptor
Page Address Translation (T = 0)
Direct-Store Segment Address (T = 1)*
DSI/ISI Exception
Otherwise
I-Fetch with N-Bit Set in Segment Descriptor (No-Execute)
Generate 52-Bit Virtual Address from Segment Descriptor
Compare Virtual Address with TLB Entries
TLB Miss
TLB Hit
(See *Figure 5-8* on page 203)
Perform Page Table Search Operation
(See *Figure 5-9* on page 205)
Access Permitted
Access Protected
PTE Not Found
PTE Found
Translate Address
Access Faulted
Access Faulted
Load TLB Entry
Continue Access to Memory Subsystem
*In the case of instruction accesses, causes ISI exception
Optional in the PowerPC Architecture. Implemented in the 750GX.

If the T bit in the Segment Register is cleared (SR[T] = 0), then page-address translation is selected. The information in the segment descriptor is then used to generate the 52-bit virtual address. The virtual address is used to identify the page-address-translation information (stored as page table entries [PTEs] in a page table in memory). For increased performance, the 750GX has two on-chip TLBs to cache recently-used translations on-chip.

If an access hits in the appropriate TLB, page translation succeeds and the physical address bits are forwarded to the memory subsystem. If the required translation is not resident, the MMU performs a search of the page table. If the required PTE is found, a TLB entry is allocated and the page translation is attempted again. This time, the TLB is guaranteed to hit. When the translation is located, the access is qualified with the appropriate protection bits. If the access causes a protection violation, either an ISI or DSI exception is generated.

If the PTE is not found by the table-search operation, a page-fault condition exists, and an ISI or DSI exception occurs so software can handle the page fault.

### 5.1.7 MMU Exceptions Summary

To complete any memory access, the effective address must be translated to a physical address. As specified by the architecture, an MMU exception condition occurs if this translation fails for one of the following reasons:

- Page fault. There is no valid entry in the page table for the page specified by the effective address (and segment descriptor), and there is no valid BAT translation.
- An address translation is found, but the access is not allowed by the memory-protection mechanism.

The translation exception conditions defined by the OEA for 32-bit implementations cause either the ISI or the DSI exception to be taken as shown in *Table 5-3*.

*Table 5-3. Translation Exception Conditions* (Page 1 of 2)

| Condition | Description | Exception |
|---|---|---|
| Page fault (no PTE found) | No matching PTE found in page tables (and no matching BAT array entry) | I access: ISI exception<br>SRR1[1] = 1 |
| | | D access: DSI exception<br>DSISR[1] =1 |
| Block protection violation | Conditions described for blocks in "Block Memory Protection" in Chapter 7, "Memory Management," in the *PowerPC Microprocessor Family: The Programming Environments Manual.*" | I access: ISI exception<br>SRR1[4] = 1 |
| | | D access: DSI exception<br>DSISR[4] =1 |
| Page-protection violation | Conditions described for pages in "Page Memory Protection" in Chapter 7, "Memory Management," in the *PowerPC Microprocessor Family: The Programming Environments Manual.* | I access: ISI exception<br>SRR1[4] = 1 |
| | | D access: DSI exception<br>DSISR[4] = 1 |
| No-execute protection violation | Attempt to fetch instruction when SR[N] = 1 | ISI exception<br>SRR1[3] = 1 |

*Table 5-3. Translation Exception Conditions* (Page 2 of 2)

| Condition | Description | Exception |
|---|---|---|
| Instruction fetch from direct-store segment | Attempt to fetch instruction when SR[T] = 1 | ISI exception<br>SRR1[3] =1 |
| Data access to direct-store segment (including floating-point accesses) | Attempt to perform load or store (including floating-point (FP) load or store) when SR[T] = 1 | DSI exception<br>DSISR[5] =1 |
| Instruction fetch from guarded memory | Attempt to fetch instruction when MSR[IR] = 1 and either matching xBAT[G] = 1, or no matching BAT entry and PTE[G] = 1 | ISI exception<br>SRR1[3] =1 |

The state saved by the processor for each of these exceptions contains information that identifies the address of the failing instruction. See *Chapter 4, Exceptions,* on page 151 for a more detailed description of exception processing.

In addition to the translation exceptions, there are other MMU-related conditions (some of them defined as implementation-specific, and therefore not required by the architecture) that can cause an exception to occur.

These exception conditions map to processor exceptions as shown in *Table 5-4*. The only MMU exception conditions that occur when MSR[DR] = 0 are those that cause an alignment exception for data accesses. For more detailed information about the conditions that cause an alignment exception (in particular for string/multiple instructions), see *Section 4.5.6, Alignment Exception (0x00600),* on page 170.

**Notes:**

- Some exception conditions depend upon whether the memory area is set up as write-though (W = 1) or caching-inhibited (I = 1).
- These bits are described fully in “Memory/Cache Access Attributes,” in Chapter 5, “Cache Model and Memory Coherency,” of the *PowerPC Microprocessor Family: The Programming Environments Manual.*
- Also see *Chapter 4, Exceptions,* on page 151 and Chapter 6, “Exceptions,” in the *PowerPC Microprocessor Family: The Programming Environments Manual* for a complete description of the SRR1 and DSISR bit settings for these exceptions.

*Table 5-4. Other MMU Exception Conditions for the 750GX Processor* (Page 1 of 2)

| Condition | Description | Exception |
|---|---|---|
| Data Cache Block Set to Zero (**dcbz**) with W = 1 or I = 1 | **dcbz** instruction to write-through or cache-inhibited segment or block | Alignment exception (not required by architecture for this condition) |
| Load Word and Reserve Indexed (**lwarx)**, Store Word Conditional Indexed (**stwcx).**, External Control In Word Indexed (**eciwx**), or External Control Out Word Indexed (**ecowx**) instruction to direct-store segment | Reservation instruction or external control instruction when SR[T] = 1 | DSI exception<br>DSISR[5] = 1 |
| Floating-point load or store to direct-store segment | Floating-point memory access when SR[T] =1 | See data access to direct-store segment in *Table 5-3* on page 192. |
| Load or store that results in a direct-store error | A DSI exception is taken when a load or store is attempted to a direct-store segment (SR[T] = 1) | DSI exception<br>For additional information, see *Section 4.5.3, DSI Exception (0x00300),* on page 169. |

*Table 5-4. Other MMU Exception Conditions for the 750GX Processor* (Page 2 of 2)

| Condition | Description | Exception |
|---|---|---|
| **eciwx** or **ecowx** attempted when external control facility disabled | **eciwx** or **ecowx** attempted with EAR[E] = 0 | DSI exception<br>DSISR[11] = 1 |
| Load Multiple Word (**lmw**), Store Multiple Word (**stmw**), **lswi**, Load String Word Immediate (**lswx**), Store String Word Immediate (**stswi**), or Store String Word Indexed x-form (**stswx**) instruction attempted in little-endian mode | **lmw**, **stmw**, **lswi**, **lswx**, **stswi**, or **stswx** instruction attempted while MSR[LE] = 1 | Alignment exception |
| Operand misalignment | Translation enabled and a floating-point load/store, **stmw**, **stwcx.**, **lmw**, **lwarx**, **eciwx**, or **ecowx** instruction operand is not word-aligned | Alignment exception (some of these cases are implementation-specific) |

### 5.1.8 MMU Instructions and Register Summary

The MMU instructions and registers allow the operating system to set up the block-address-translation areas and the page tables in memory.

**Notes:**

- Because the implementation of TLBs is optional, the instructions that refer to these structures are also optional. However, as these structures serve as caches of the page table, the architecture specifies a software protocol for maintaining coherency between these caches and the tables in memory whenever the tables in memory are modified. When the tables in memory are changed, the operating system purges these caches of the corresponding entries, allowing the translation caching mechanism to refetch from the tables when the corresponding entries are required.
- Also note that the 750GX implements all TLB-related instructions except TLB Invalidate All (**tlbia**), which is treated as an illegal instruction.

Because the MMU specification for PowerPC processors is so flexible, it is recommended that the software that uses these instructions and registers be encapsulated into subroutines to minimize the impact of migrating across the family of implementations.

*Table 5-5* summarizes the 750GX’s instructions that specifically control the MMU. For more detailed information about the instructions, see *Chapter 2, Programming Model,* on page 57 and Chapter 8, “Instruction Set,” in the *PowerPC Microprocessor Family: The Programming Environments Manual*.

*Table 5-5. 750GX Microprocessor Instruction Summary—Control MMUs* (Page 1 of 2)

| Instruction | Description |
|---|---|
| **mtsr** SR,**r**S | Move-to Segment Register<br>SR[SR#]⟵ **r**S |
| **mtsrin** **r**S,**r**B | Move-to Segment Register Indirect<br>SR[**r**B[0–3]]⟵**r**S |
| **mfsr** **r**D,SR | Move-from Segment Register<br>**r**D⟵SR[SR#] |
| **mfsrin** **r**D,**r**B | Move-from Segment Register Indirect<br>**r**D⟵SR[**r**B[0–3]] |

*Table 5-5. 750GX Microprocessor Instruction Summary—Control MMUs* (Page 2 of 2)

| Instruction | Description |
|---|---|
| **tlbie** rB[1] | TLB Invalidate Entry<br>For effective address specified by **r**B, TLB[V]⟵0<br>The **tlbie** instruction invalidates all TLB entries indexed by the EA, and operates on both the instruction and data TLBs simultaneously invalidating four TLB entries. The index corresponds to bits 14–19 of the EA.<br>Software must ensure that instruction fetches or memory references to the virtual pages specified by the **tlbie** instruction have been completed prior to executing the **tlbie** instruction. |
| **tlbsync**[1] | TLB Synchronize<br>Synchronizes the execution of all other **tlbie** instructions in the system. In the 750GX, when the TLB Invalidate Synchronize ($\overline{\text{TLBISYNC}}$) signal is negated, instruction execution can continue or resume after the completion of a **tlbsync** instruction. When the $\overline{\text{TLBISYNC}}$ signal is asserted, instruction execution stops after the completion of a **tlbsync** instruction. |

[1] These instructions are defined by the PowerPC Architecture, but are optional.

*Figure 5-6* summarizes the registers that the operating system uses to program the 750GX’s MMUs. These registers are accessible to supervisor-level software only.

These registers are described in *Chapter 2, Programming Model,* on page 57.

*Table 5-6. 750GX Microprocessor MMU Registers*

| Register | Description |
|---|---|
| Segment registers<br>(SR0–SR15) | The sixteen 32-bit Segment Registers are present only in 32-bit implementations of the PowerPC Architecture. The fields in the Segment Register are interpreted differently depending on the value of bit 0. The Segment Registers are accessed by the **mtsr**, **mtsrin**, **mfsr**, and **mfsrin** instructions. |
| BAT registers<br>(IBAT0U–IBAT7U, IBAT0L–IBAT7L, DBAT0U–DBAT7U, and DBAT0L–DBAT7L) | There are 32 BAT registers, organized as eight pairs of instruction BAT registers (IBAT0U–IBAT7U paired with IBAT0L–IBAT7L) and eight pairs of data BAT registers (DBAT0U–DBAT7U paired with DBAT0L–DBAT7L). The BAT registers are defined as 32-bit registers in 32-bit implementations. These are Special-Purpose Registers that are accessed by the Move-to Special Purpose Register (**mtspr**) and Move-from Special Purpose Register (**mfspr**) instructions. |
| SDR1 | The SDR1 register specifies the variables used in accessing the page tables in memory. SDR1 is defined as a 32-bit register for 32-bit implementations. This Special-Purpose Register is accessed by the **mtspr** and **mfspr** instructions. |

## 5.2 Real-Addressing Mode

If address translation is disabled (MSR[IR] = 0 or MSR[DR] = 0) for a particular access, the effective address is treated as the physical address and is passed directly to the memory subsystem as described in Chapter 7, “Memory Management,” in the *PowerPC Microprocessor Family: The Programming Environments Manual*.

Note that the default WIMG bits (0b0011) cause data accesses to be considered cacheable (I = 0), and thus load-and-store accesses are weakly ordered. This is the case even if the data cache is disabled in the HID0 register (as it is after a hard reset). If I/O devices require load-and-store accesses to occur in strict program order (strongly ordered), translation must be enabled so that the corresponding I bit can be set. Note also, that the G bit must be set to ensure that the accesses are strongly ordered. For instruction accesses, the default memory-access mode bits (WIMG) are also 0b0001. That is, instruction accesses are considered cacheable (I = 0), and the memory is guarded. Again, instruction accesses are considered cacheable even if the instruction cache is disabled in the HID0 register (after a hard reset). The W and M bits have no effect on the instruction cache.

For information on the synchronization requirements for changes to MSR[IR] and MSR[DR], see *Section 2.3.2.4, Synchronization,* on page 90 in this manual and "Synchronization Requirements for Special Registers and for Lookaside Buffers" in Chapter 2 of the *PowerPC Microprocessor Family: The Programming Environments Manual.*

## 5.3 Block-Address Translation

The block-address-translation (BAT) mechanism in the OEA provides a way to map ranges of effective addresses larger than a single page into contiguous areas of physical memory. Such areas can be used for data that is not subject to normal virtual memory handling (paging), such as a memory-mapped display buffer or an extremely large array of numerical data.

Block-address translation in the 750GX is described in Chapter 7, "Memory Management," in the *PowerPC Microprocessor Family: The Programming Environments Manual* for 32-bit implementations.

**Implementation Note:** The 750GX's BAT registers are not initialized by the hardware after the power-up or reset sequence. Consequently, all valid bits in both instruction and data BATs must be cleared before setting any BAT for the first time. This is true regardless of whether address translation is enabled. Also, software must avoid overlapping blocks while updating a BAT or areas. *Even if translation is disabled, multiple BAT hits are treated as programming errors and can corrupt the BAT registers and produce unpredictable results. Always reset to zero during the reset Interrupt Service Routine (ISR). After zeroing all BATs, set them (in order) to the desired values. A hard reset ($\overline{\text{HRESET}}$) disorders the BATs. A soft reset ($\overline{\text{SRESET}}$) does not.*

## 5.4 Memory Segment Model

The 750GX adheres to the memory segment model as defined in Chapter 7, "Memory Management," in the *PowerPC Microprocessor Family: The Programming Environments Manual* for 32-bit implementations. Memory in the PowerPC OEA is divided into 256-MB segments. This segmented memory model provides a way to map 4-KB pages of effective addresses to 4-KB pages in physical memory (page-address translation), while providing the programming flexibility afforded by a large virtual address space (52 bits).

The segment/page-address-translation mechanism might be superseded by the block-address-translation (BAT) mechanism described in *Section 5.3, Block-Address Translation,* on page 196. If not, the translation proceeds in the following two steps:

1. From effective address to the virtual address (which never exists as a specific entity but can be considered to be the concatenation of the virtual page number and the byte offset within a page), and
2. From virtual address to physical address.

This section highlights those areas of the memory segment model defined by the OEA that are specific to the 750GX.

### 5.4.1 Page History Recording

Referenced (R) and changed (C) bits in each PTE keep history information about the page. They are maintained by a combination of the 750GX's table-search hardware and the system software. The operating system uses this information to determine which areas of memory to write back to disk when new pages must be allocated in main memory. Referenced and changed recording is performed only for accesses made with page-address translation and not for translations made with the BAT mechanism or for accesses that correspond to direct-store (T = 1) segments. Furthermore, R and C bits are maintained only for accesses made while address translation is enabled (MSR[IR] = 1 or MSR[DR] = 1).

In the 750GX, the referenced and changed bits are updated as follows.

- For TLB hits, the C bit is updated according to *Table 5-7*.
- For TLB misses, when a table-search operation is in progress to locate a PTE, the R and C bits are updated (set, if required) to reflect the status of the page based on this access.

*Table 5-7. Table-Search Operations to Update History Bits—TLB Hit Case*

| R and C bits in TLB Entry | Processor Action |
|---|---|
| 00 | Combination does not occur |
| 01 | Combination does not occur |
| 10 | Read: No special action<br>Write: The 750GX initiates a table-search operation to update the C bit. |
| 11 | No special action for read or write |

*Table 5-7* shows that the status of the C bit in the TLB entry (in the case of a TLB hit) is what causes the processor to update the C bit in the PTE (the R bit is assumed to be set in the page tables if there is a TLB hit). Therefore, when software clears the R and C bits in the page tables in memory, it must invalidate the TLB entries associated with the pages whose referenced and changed bits were cleared.

The Data Cache Block Touch (**dcbt**) and Data Cache Block Touch for Store (**dcbtst**) instructions can execute if there is a TLB or BAT hit or if the processor is in real-addressing mode. In the case of a TLB or BAT miss, these instructions are treated as no-ops. They do not initiate a table-search operation, and they do not set either the R or C bits.

As defined by the PowerPC Architecture, the referenced and changed bits are updated as if address translation were disabled (real-addressing mode). If these update accesses hit in the data cache, they are not seen on the external bus. If they miss in the data cache, they are performed as typical cache-line-fill accesses on the bus (assuming the data cache is enabled).

#### *5.4.1.1 Referenced Bit*

The referenced (R) bit of a page is located in the PTE in the page table. Every time a page is referenced (with a read or write access) and the R bit is zero, the 750GX sets the R bit in the page table. The OEA specifies that the referenced bit can be set immediately, or the setting can be delayed until the memory access is determined to be successful. Because the reference to a page is what causes a PTE to be loaded into the TLB, the referenced bit in all TLB entries is effectively always set. The processor never automatically clears the referenced bit.

The referenced bit is only a hint to the operating system about the activity of a page. At times, the referenced bit might be set although the access was not logically required by the program or even if the access was prevented by memory protection. Examples of this in PowerPC systems include the following:

- Fetching of instructions not subsequently executed.
- A memory reference caused by a speculatively executed instruction that is mispredicted.
- Accesses generated by an **lswx** or **stswx** instruction with a zero length.
- Accesses generated by an **stwcx.** instruction when no store is performed because a reservation does not exist.

- Accesses that cause exceptions and are not completed.

#### 5.4.1.2 Changed Bit

The changed bit of a page is located both in the PTE in the page table and in the copy of the PTE loaded into the TLB (if a TLB is implemented, as in the 750GX). Whenever a data store instruction is executed successfully, if the TLB search (for page-address translation) results in a hit, then the changed bit in the matching TLB entry is checked. If it is already set, it is not updated. If the TLB changed bit is 0, the 750GX initiates the table-search operation to set the C bit in the corresponding PTE in the page table. The 750GX then reloads the TLB (with the C bit set).

The changed bit (in both the TLB and the PTE in the page tables) is set only when a store operation is allowed by the page memory-protection mechanism and the store is guaranteed to be in the execution path (unless an exception, other than those caused by the **sc**, **rfi**, or trap instructions, occurs). Furthermore, the following conditions can cause the C bit to be set:

- The execution of an **stwcx.** instruction is allowed by the memory-protection mechanism but a store operation is not performed.
- The execution of an **stswx** instruction is allowed by the memory-protection mechanism but a store operation is not performed because the specified length is zero.
- The store operation is not performed because an exception occurs before the store is performed.

Again, note that although the execution of the **dcbt** and **dcbtst** instructions might cause the R bit to be set, they never cause the C bit to be set.

#### 5.4.1.3 Scenarios for Referenced and Changed Bit Recording

This section provides a summary of the model (defined by the OEA) that is used by PowerPC processors for maintaining the referenced and changed bits. In some scenarios, the bits are guaranteed to be set by the processor; in some scenarios, the architecture allows the bits to be set (not absolutely required); and in some scenarios, the bits are guaranteed to not be set. Note that when the 750GX updates the R and C bits in memory, the accesses are performed as if MSR[DR] = 0 and G = 0 (that is, as nonguarded cacheable operations in which coherency is required).

*Table 5-8* on page 198 defines a prioritized list of the R and C bit settings for all scenarios. The entries in the table are prioritized from top to bottom, so that a scenario near the top of the table takes precedence over a scenario near the bottom of the table. For example, if an **stwcx.** instruction causes a page-protection violation and there is no reservation, the C bit is not altered. Note that in the table, load operations include those generated by load instructions, by the **eciwx** instruction, and by the cache-management instructions that are treated as a load with respect to address translation. Similarly, store operations include those operations generated by store instructions, by the **ecowx** instruction, and by the cache-management instructions that are treated as a store with respect to address translation.

*Table 5-8. Model for Guaranteed R and C Bit Settings* (Page 1 of 2)

| Priority | Scenario | Causes Setting of R Bit | | Causes Setting of C Bit | |
|---|---|---|---|---|---|
| | | OEA | 750GX | OEA | 750GX |
| 1 | No-execute protection violation | No | No | No | No |
| 2 | Page-protection violation | Maybe | Yes | No | No |
| 3 | Out-of-order instruction fetch or load operation | Maybe | No | No | No |

*Table 5-8. Model for Guaranteed R and C Bit Settings* (Page 2 of 2)

| Priority | Scenario | Causes Setting of R Bit | | Causes Setting of C Bit | |
|---|---|---|---|---|---|
| | | OEA | 750GX | OEA | 750GX |
| 4 | Out-of-order store operation. Required by the sequential execution model in the absence of system-caused or imprecise exceptions, or of floating-point assist exception for instructions that would cause no other kind of precise exception. | Maybe[1] | No | No | No |
| 5 | All other out-of-order store operations | Maybe[1] | No | Maybe[1] | No |
| 6 | Zero-length load (**lswx**) | Maybe | No | No | No |
| 7 | Zero-length store (**stswx**) | Maybe[1] | No | Maybe[1] | No |
| 8 | Store conditional (**stwcx.**) that does not store | Maybe[1] | Yes | Maybe[1] | Yes |
| 9 | In-order instruction fetch | Yes | Yes | No | No |
| 10 | Load instruction or **eciwx** | Yes | Yes | No | No |
| 11 | Store instruction, **ecowx,** or **dcbz** instruction | Yes | Yes | Yes | Yes |
| 12 | Instruction Cache Block Invalidate (**icbi**), **dcbt**, or **dcbtst** instruction | Maybe | No | No | No |
| 13 | Data Cache Block Store (**dcbst**) or Data Cache Block Flush (**dcbf**) instruction | Maybe | Yes | No | No |
| 14 | Data Cache Block Invalidate (**dcbi**) instruction | Maybe[1] | Yes | Maybe[1] | Yes |

**Note:**
[1] If C is set, R is guaranteed to be set also.

For more information, see “Page History Recording” in Chapter 7, “Memory Management,” of the *PowerPC Microprocessor Family: The Programming Environments Manual*.

### 5.4.2 Page Memory Protection

The 750GX implements page memory protection as it is defined in Chapter 7, “Memory Management,” in the *PowerPC Microprocessor Family: The Programming Environments Manual*.

### 5.4.3 TLB Description

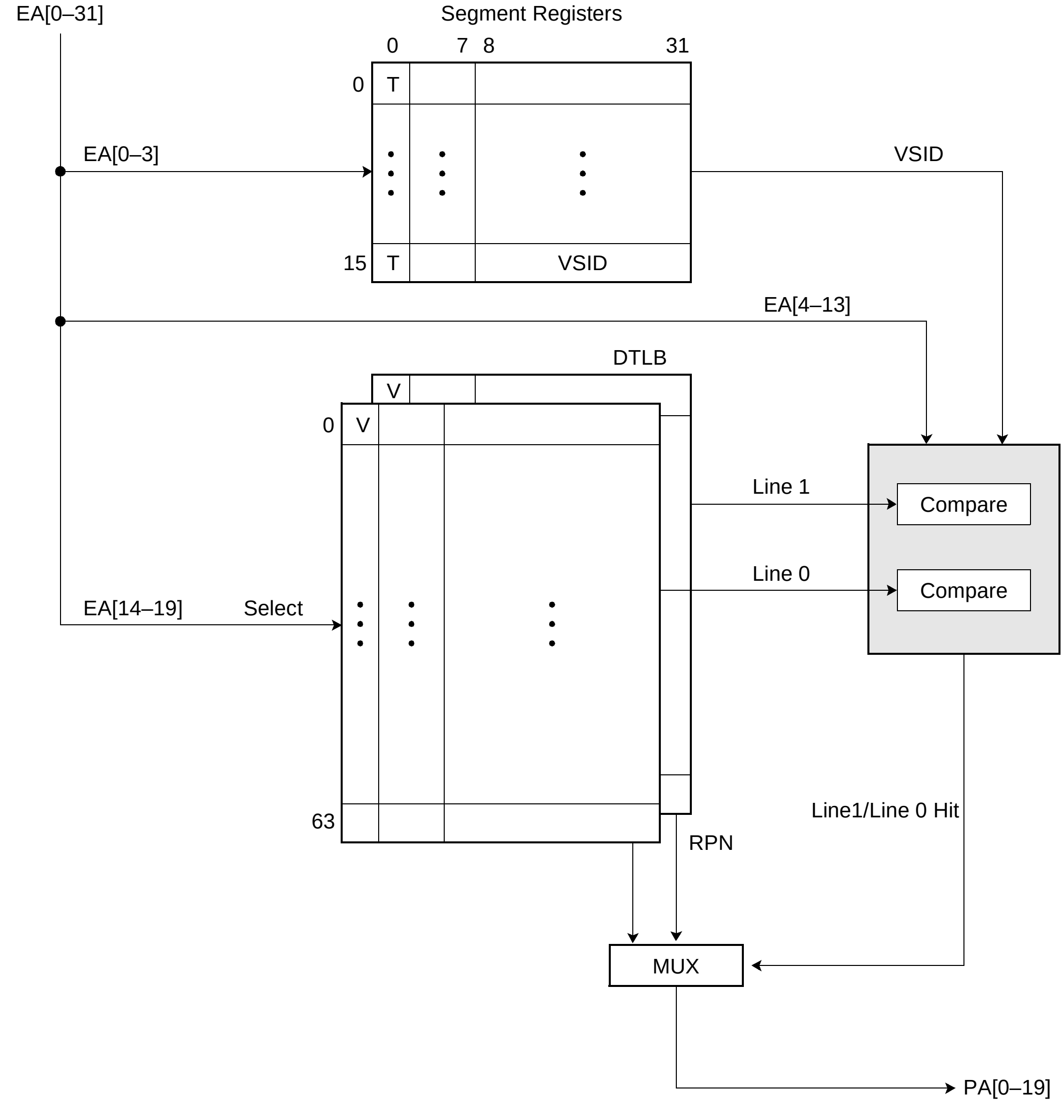
The 750GX implements separate 128-entry data and instruction TLBs to maximize performance. This section describes the hardware resources provided in the 750GX to facilitate page-address translation. Note that the architecture does not specify the hardware implementation of the MMU, and while this description applies to the 750GX, it does not necessarily apply to other PowerPC processors.

#### *5.4.3.1 TLB Organization*

Because the 750GX has two MMUs (IMMU and DMMU) that operate in parallel, some of the MMU resources are shared, and some are actually duplicated (shadowed) in each MMU to maximize performance. For example, although the architecture defines a single set of Segment Registers for the MMU, the 750GX maintains two identical sets of Segment Registers, one for the IMMU and one for the DMMU. When an instruction that updates the Segment Register executes, the 750GX automatically updates both sets.

Each TLB contains 128 entries organized as a 2-way set-associative array with 64 sets as shown in *Figure 5-7* for the DTLB (the ITLB organization is the same). When an address is being translated, a set of two TLB entries is indexed in parallel with the access to a Segment Register. If the address in one of the two TLB entries is valid and matches the 40-bit virtual page number, that TLB entry contains the translation. If no match is found, a TLB miss occurs.

*Figure 5-7. Segment Register and DTLB Organization*



Unless the access is the result of an out-of-order access, a hardware table-search operation begins if there is a TLB miss. If the access is out of order, the table-search operation is postponed until the access is required, at which point the access is no longer out of order. When the matching PTE is found in memory, it is loaded into the TLB entry selected by the LRU replacement algorithm, and the translation process begins again, this time with a TLB hit.

To uniquely identify a TLB entry as the required PTE, each TLB entry contains, in addition to the PTE, an additional 4-bit field called the Extended Page Index (EPI). The EPI contains bits 10–13 of the EA. Software cannot access the TLB arrays directly, except to invalidate an entry with the **tlbie** instruction.

Each set of TLB entries has one associated LRU bit. The LRU bit for a set is updated any time either entry is used, even if the access is speculative. Invalid entries are always the first to be replaced.

Although both MMUs can be accessed simultaneously (both sets of Segment Registers and TLBs can be accessed in the same clock), only one exception condition can be reported at a time. ITLB miss exception conditions are reported when there are no more instructions to be dispatched or retired (the pipeline is empty), and DTLB miss exception conditions are reported when the load or store instruction is ready to be retired. See *Chapter 6, Instruction Timing,* on page 209 for more detailed information about the internal pipelines and the reporting of exceptions.

When an instruction or data access occurs, the effective address is routed to the appropriate MMU. EA0–EA3 select one of the 16 Segment Registers and the remaining effective address bits, and the virtual segment ID (VSID) field from the Segment Register is passed to the TLB. EA[14–19] then select two entries in the TLB. The valid bits are checked and the 40-bit virtual page number (24-bit VSID and EA[4–19]) must match the VSID, EPI, and API fields of the TLB entries. If one of the entries hits, the page-protection (PP) bits are checked for a protection violation. If these bits do not cause an exception, the C bit is checked and a table-search operation is initiated if C must be updated. If C does not require updating, the real page number (RPN) value is passed to the memory subsystem and the WIMG bits are then used as attributes for the access.

Although address translation is disabled on a reset condition, the valid bits of TLB entries are not automatically cleared. Thus, TLB entries must be explicitly cleared by the system software (with the **tlbie** instruction) before the valid entries are loaded and address translation is enabled. Also, note that the Segment Registers do not have a valid bit, and so they should also be initialized before translation is enabled.

### *5.4.3.2 TLB Invalidation*

The 750GX implements the optional **tlbie** and **tlbsync** instructions, which are used to invalidate TLB entries. The execution of the **tlbie** instruction always invalidates four entries—both the ITLB and DTLB entries indexed by EA[14–19].

The architecture allows **tlbie** to optionally enable a TLB invalidate signaling mechanism in hardware so that other processors also invalidate their resident copies of the matching PTE. The 750GX does not signal the TLB invalidation to other processors, nor does it perform any action when a TLB invalidation is performed by another processor.

The **tlbsync** instruction causes instruction execution to stop if the $\overline{\text{TLBISYNC}}$ signal is asserted. If $\overline{\text{TLBISYNC}}$ is negated, instruction execution might continue or resume after the completion of a **tlbsync** instruction. *Section 8.7.2, TLBISYNC Input,* on page 319 describes the TLB synchronization mechanism in further detail.

The **tlbia** instruction is not implemented on the 750GX**,** and when its opcode is encountered, an illegal instruction program exception is generated. To invalidate all entries of both TLBs, 64 **tlbie** instructions must be executed, incrementing the value in EA14–EA19 by one each time. (See Chapter 8, “Instruction Set” in the the *PowerPC Microprocessor Family: The Programming Environments Manual* for detailed information about this instruction.) Software must ensure that instruction fetches or memory references to the virtual pages specified by the **tlbie** have been completed prior to executing the **tlbie** instruction.

Other than the possible TLB miss on the next instruction prefetch, the **tlbie** instruction does not affect the instruction fetch operation—that is, the prefetch buffer is not purged and does not cause these instructions to be refetched.

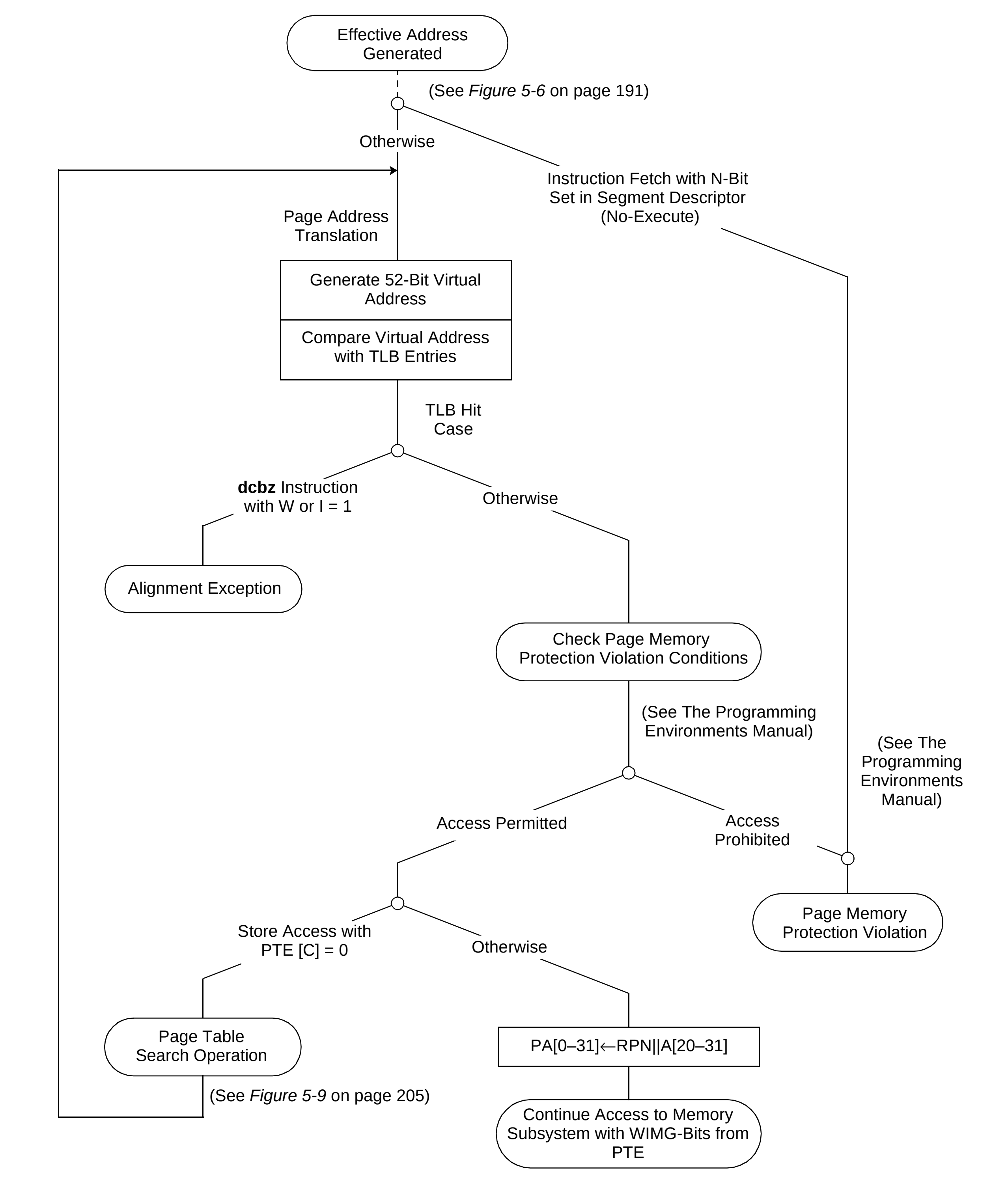
### 5.4.4 Page-Address-Translation Summary

*Figure 5-8* on page 203 provides the detailed flow for the page-address-translation mechanism. The figure includes the checking of the N bit in the segment descriptor and then expands on the 'TLB Hit' branch of *Figure 5-6* on page 191.

The detailed flow for the 'TLB Miss' branch of *Figure 5-6* on page 191 is described in *Section 5.4.5, Page Table-Search Operation,* on page 204.

**Note:** As in the case of block-address translation, if an attempt is made to execute a **dcbz** instruction to a page marked either write-through or caching-inhibited (W = 1 or I = 1), an alignment exception is generated. The checking of memory-protection violation conditions is described in Chapter 7, "Memory Management" in the *PowerPC Microprocessor Family: The Programming Environments Manual*.

*Figure 5-8. Page-Address-Translation Flow—TLB Hit*


Effective Address Generated
(See *Figure 5-6* on page 191)
Otherwise
Instruction Fetch with N-Bit Set in Segment Descriptor (No-Execute)
Page Address Translation
Generate 52-Bit Virtual Address
Compare Virtual Address with TLB Entries
TLB Hit Case
**dcbz** Instruction with W or I = 1
Otherwise
Alignment Exception
Check Page Memory Protection Violation Conditions
(See The Programming Environments Manual)
(See The Programming Environments Manual)
Access Permitted
Access Prohibited
Page Memory Protection Violation
Store Access with PTE [C] = 0
Otherwise
Page Table Search Operation
PA[0–31]←RPN||A[20–31]
(See *Figure 5-9* on page 205)
Continue Access to Memory Subsystem with WIMG-Bits from PTE

### 5.4.5 Page Table-Search Operation

If the translation is not found in the TLBs (a TLB miss), the 750GX initiates a table-search operation, which is described in this section. Formats for the PTE are given in “PTE Format for 32-Bit Implementations,” in Chapter 7, “Memory Management” of the *PowerPC Microprocessor Family: The Programming Environments Manual.*

The following is a summary of the page-table-search process performed by the 750GX.

1. The 32-bit physical address of the primary page-table-entry group (PTEG) is generated as described in “Page Table Addresses” in Chapter 7, “Memory Management” of the *PowerPC Microprocessor Family: The Programming Environments Manual*.
2. The first PTE (PTE0) in the primary PTEG is read from memory if cache is enabled. PTE reads occur with an implied WIM memory/cache mode control bit setting of 0b001. Therefore, they are considered cacheable, read (burst) from memory, and placed in the cache.
3. The PTE in the selected PTEG is tested for a match with the virtual page number (VPN) of the access. The VPN is the VSID concatenated with the page index field of the virtual address. For a match to occur, the following must be true:
   - PTE[H] = 0
   - PTE[V] = 1
   - PTE[VSID] = VA[0–23]
   - PTE[API] = VA[24–29]
4. If a match is not found, step 3 is repeated for each of the other seven PTEs in the primary PTEG. If a match is found, the table-search process continues as described in step 8. If a match is not found within the eight PTEs of the primary PTEG, the address of the secondary PTEG is generated.
5. The first PTE (PTE0) in the secondary PTEG is read from memory if cache is enabled. Again, because PTE reads have a WIM bit combination of 0b001, an entire cache line is read into the on-chip cache.
6. The PTE in the selected secondary PTEG is tested for a match with the virtual page number (VPN) of the access. For a match to occur, the following must be true:
   - PTE[H] = 1
   - PTE[V] = 1
   - PTE[VSID] = VA[0–23]
   - PTE[API] = VA[24–29]
7. If a match is not found, step 6 is repeated for each of the other seven PTEs in the secondary PTEG. If it is never found, an exception is taken (step 9).
8. If a match is found, the PTE is written into the on-chip TLB and the R bit is updated in the PTE in memory (if necessary). If there is no memory-protection violation, the C bit is also updated in memory (if the access is a write operation), and the table search is complete.
9. If a match is not found within the eight PTEs of the secondary PTEG, the search fails, and a page-fault exception condition occurs (either an ISI exception or a DSI exception).

*Figure 5-9* on page 205 and *Figure 5-10* on page 206 show how the conceptual model for the primary and secondary page table-search operations, described in the *PowerPC Microprocessor Family: The Programming Environments Manual*, are realized in the 750GX.

*Figure 5-9* shows the case of a **dcbz** instruction that is executed with W = 1 or I = 1, and that the R bit can be updated in memory (if required) before the operation is performed or the alignment exception occurs. The R bit can also be updated if memory protection is violated.

*Figure 5-9. Primary Page Table Search*

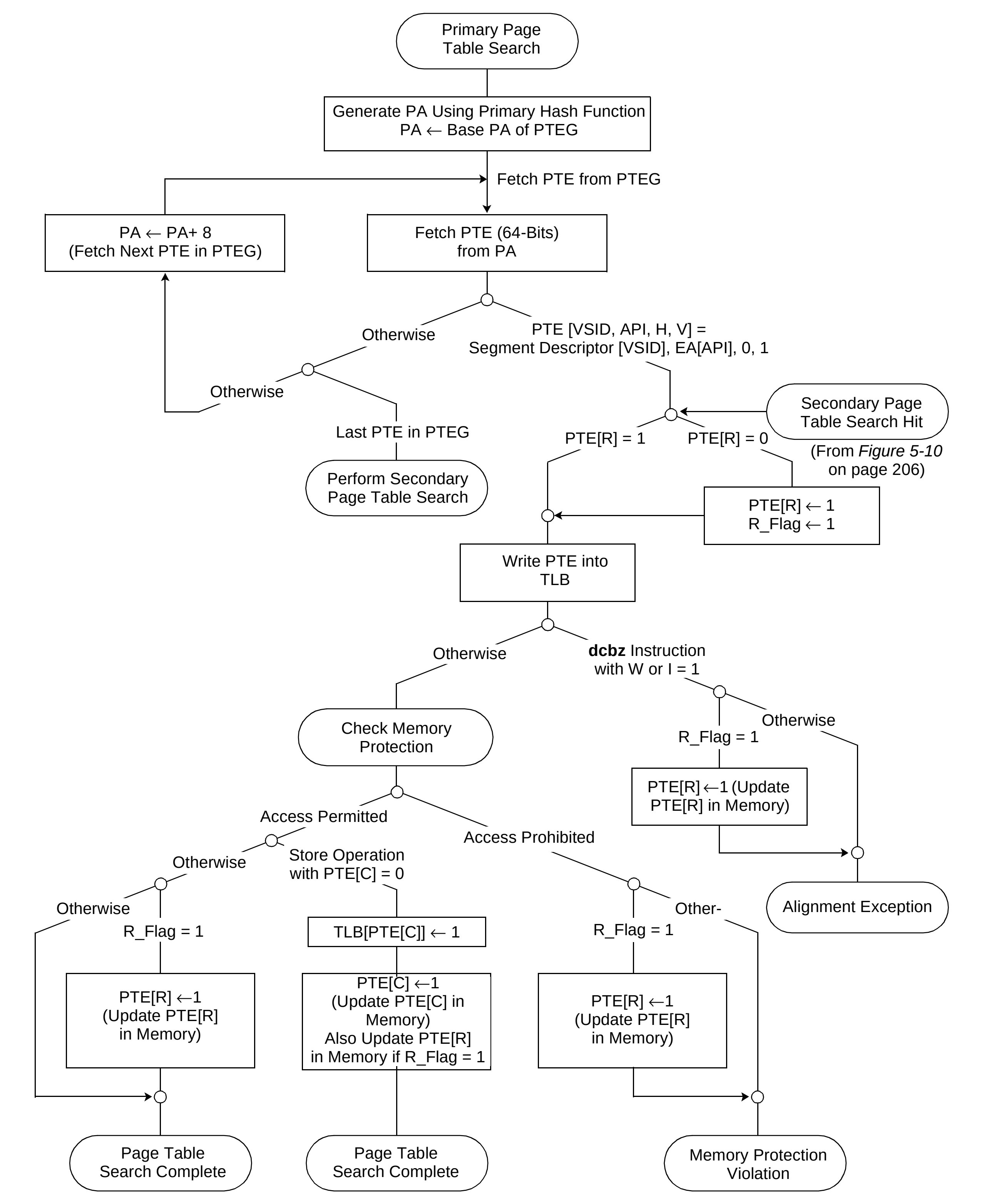
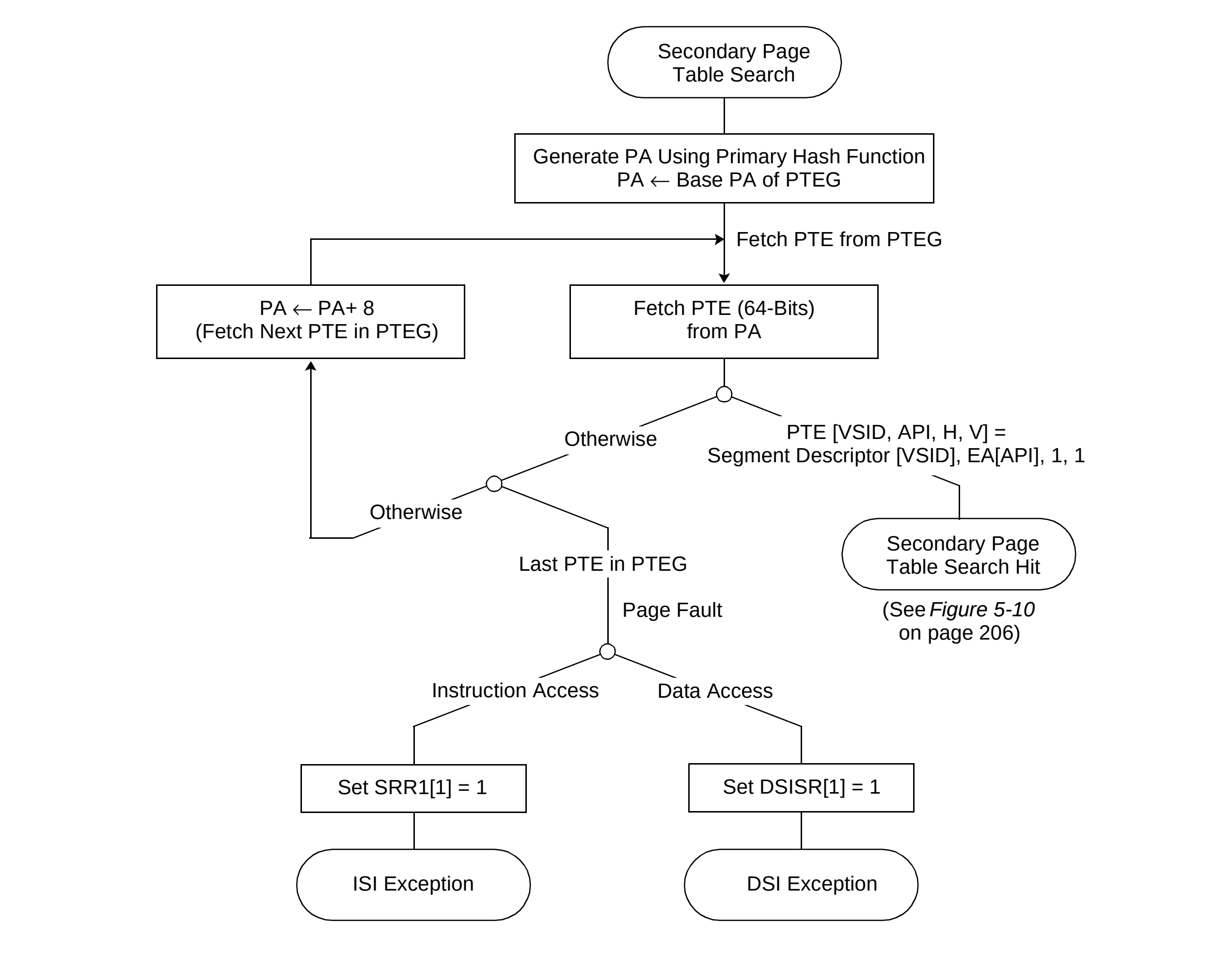

Primary Page Table Search
Generate PA Using Primary Hash Function
PA ← Base PA of PTEG
Fetch PTE from PTEG
PA ← PA+ 8
(Fetch Next PTE in PTEG)
Fetch PTE (64-Bits) from PA
Otherwise
PTE [VSID, API, H, V] = Segment Descriptor [VSID], EA[API], 0, 1
Otherwise
Last PTE in PTEG
Perform Secondary Page Table Search
Secondary Page Table Search Hit
(From Figure 5-10 on page 206)
PTE[R] = 1
PTE[R] = 0
PTE[R] ← 1
R_Flag ← 1
Write PTE into TLB
Otherwise
dcbz Instruction with W or I = 1
R_Flag = 1
Otherwise
PTE[R] ←1 (Update PTE[R] in Memory)
Check Memory Protection
Access Permitted
Access Prohibited
Alignment Exception
Otherwise
Store Operation with PTE[C] = 0
Otherwise
R_Flag = 1
TLB[PTE[C]] ← 1
Other-
R_Flag = 1
PTE[R] ←1
(Update PTE[R] in Memory)
PTE[C] ←1
(Update PTE[C] in Memory)
Also Update PTE[R] in Memory if R_Flag = 1
PTE[R] ←1
(Update PTE[R] in Memory)
Page Table Search Complete
Page Table Search Complete
Memory Protection Violation

*Figure 5-10. Secondary Page-Table-Search Flow*


Secondary Page Table Search
Generate PA Using Primary Hash Function
PA ← Base PA of PTEG
Fetch PTE from PTEG
PA ← PA+ 8
(Fetch Next PTE in PTEG)
Fetch PTE (64-Bits) from PA
Otherwise
PTE [VSID, API, H, V] = Segment Descriptor [VSID], EA[API], 1, 1
Otherwise
Secondary Page Table Search Hit
Last PTE in PTEG
Page Fault
(See Figure 5-10 on page 206)
Instruction Access
Data Access
Set SRR1[1] = 1
Set DSISR[1] = 1
ISI Exception
DSI Exception

The load store unit (LSU) initiates out-of-order accesses without knowing whether it is legal to do so. Therefore, the MMU does not perform a hardware table search due to TLB misses until the request is required by the program flow. In these out-of-order cases, the MMU does detect protection violations and whether a **dcbz** instruction specifies a page marked as write-through or cache-inhibited. The MMU also detects alignment exceptions caused by the **dcbz** instruction and prevents the changed bit in the PTE from being updated erroneously in these cases.

If an MMU register is being accessed by an instruction in the instruction stream, the IMMU stalls for one translation cycle to perform that operation. The sequencer serializes instructions to ensure the data correctness. To update the IBATs and SRs, the sequencer classifies those operations as fetch serializing. After such an instruction is dispatched, the instruction buffer is flushed, and the fetch stalls until the instruction completes. However, to read from the IBATs, the operation is classified as execution serializing. As long as the LSU ensures that all previous instructions can be executed, subsequent instructions can be fetched and dispatched.

### 5.4.6 Page Table Updates

When TLBs are implemented (as in the 750GX), they are defined as noncoherent caches of the page tables. TLB entries must be flushed explicitly with the TLB invalidate entry instruction (**tlbie**) whenever the corresponding PTE is modified. As the 750GX is intended primarily for uniprocessor environments, it does not provide coherency of TLBs between multiple processors. If the 750GX is used in a multiprocessor environment where TLB coherency is required, all synchronization must be implemented in software.

Processors can write referenced and changed bits with unsynchronized, atomic byte store operations. Note that the valid (V), R, and C bits each reside in a distinct byte of a PTE. Therefore, extreme care must be taken to use byte writes when updating only one of these bits.

Explicitly altering certain MSR bits (using the **mtmsr** instruction), or explicitly altering PTEs, or certain system registers, can have the side effect of changing the effective or physical addresses from which the current instruction stream is being fetched. This kind of side effect is defined as an implicit branch. Implicit branches are not supported, and an attempt to perform one causes boundedly-undefined results. Therefore, PTEs must not be changed in a manner that causes an implicit branch.

Chapter 2, “PowerPC Register Set” in the *PowerPC Microprocessor Family: The Programming Environments Manual* lists the possible implicit branch conditions that can occur when system registers and MSR bits are changed.

### 5.4.7 Segment Register Updates

Synchronization requirements for using the Move-to Segment Register instructions are described in “Synchronization Requirements for Special Registers and for Lookaside Buffers” in Chapter 2, “PowerPC Register Set” in the *PowerPC Microprocessor Family: The Programming Environments Manual*.

# 6. Instruction Timing

This chapter describes how the PowerPC 750GX microprocessor fetches, dispatches, and executes instructions and how it reports the results of instruction execution. It gives detailed descriptions of how the 750GX’s execution units work, and how those units interact with other parts of the processor, such as the instruction-fetching mechanism, register files, and caches. It gives examples of instruction sequences, showing potential bottlenecks and how to minimize their effects. Finally, it includes tables that identify the unit that executes each instruction implemented on the 750GX, the latency for each instruction, and other information that is useful for the assembly language programmer.

## 6.1 Terminology and Conventions

This section provides an alphabetical glossary of terms used in this chapter. These definitions are provided as a review of commonly used terms and to point out specific ways these terms are used in this chapter.

Branch prediction

The process of guessing whether a branch will or will not be taken. Such predictions can be correct or incorrect. The term ‘predicted’ as it is used here does not imply that the prediction is correct (successful). Instructions along the predicted path are fetched and dispatched to their respective execution units conditionally and can reach the completion unit. However, these instructions must first be validated by the branch-resolution process before they can be retired.

The PowerPC Architecture defines a means for static branch prediction as part of the instruction encoding. The 750GX processor implements two types of dynamic branch prediction. See *Section 6.4.1.2, Branch Instructions and Completion,* on page 227.

Branch resolution

The determination of the path that a branch instruction must take. If a branch prediction and branch resolution occur on the same cycle, the processor simply fetches instructions on the correct path as determined by the branch instruction. For predicted branches, branch resolution must determine if the prediction was correct. If the prediction was correct, all speculatively fetched instructions that have been passed to their execution units are validated. If the prediction was wrong, the speculatively fetched instructions must be invalidated (flushed), and instruction fetching must resume along the other path for the branch instruction.

Completion

Completion occurs when an instruction has finished executing, and its results are stored in a Rename Register allocated to it by the dispatch unit. These results are available to subsequent instructions or to previously predicted branches.

Dispatch

The process of moving an instruction from the instruction queue to an execution unit. In the 750GX processor, the dispatch unit can process up to three instruction in a single cycle if one of the three is a branch. For the non-branch-type instructions, the dispatch must do a partial decode to determine the type of instruction in order to pass it to the respective execution unit. Also, a Rename Register and a place in the completion queue must be reserved; otherwise, a stall occurs. If a branch updates either the Link Register (LR) or Count Register (CTR), it must also be allocated to a completion queue entry.

Fall-through

A not-taken branch.

Fetch

The process of bringing instructions from the system memory (such as a cache or the main memory) into the instruction queue.

Folding (branch folding)

On the 750GX, a branch is expunged from (folded out of) the instruction queue via the dispatch mechanism, without being either passed to an execution unit or given a position in the completion queue. Subsequent instructions are fetched from sequential addresses for branches-not-taken and from target addresses for branches-taken.

Finish

Finishing occurs in the last cycle of execution. (This could also be the first cycle of execution for instructions that only require one cycle for execution.) In this cycle, the output Rename Register and the completion queue entry are updated to indicate that the instruction has finished executing.

Latency

The number of clock cycles necessary to execute an instruction and make ready the results of that execution for a subsequent instruction.

Pipeline

In the context of instruction timing, the term ‘pipeline’ refers to the interconnection of the stages. The events necessary to process an instruction are broken into several cycle-length tasks to allow work to be performed on several instructions simultaneously—analogous to an assembly line. As an instruction is processed, it passes from one stage to the next. When it completes one stage, that stage becomes available for the next instruction.

Although an individual instruction can take many cycles to complete (the number of cycles is called instruction latency), pipelining makes it possible to overlap the processing so that the throughput (number of instructions completed per cycle) is greater than if pipelining were not implemented.

Program order

The order of instructions in an executing program. More specifically, this term is used to refer to the original order in which program instructions are fetched into the instruction queue from the system memory.

Rename register

Temporary buffers used to hold either source or destination values for instructions that are in a stage of execution. This simplifies the passing of data outside of the General Purpose Register (GPR) file between instructions during execution.

Reservation station

A buffer between the dispatch and execution units where instructions await execution.

Retirement

Removal of a completed instruction from the completion queue. At this time, any output from the completed instruction is written to the appropriate architected destination register. This might be a GPR, a Floating Point Register (FPR), or a Condition Register (CR) field.

Stage — The processing of instructions in the 750GX is done in stages. They are: fetch, decode/dispatch, execute, complete, and retirement. The fetch unit brings instructions from the memory system into the instruction queue. Once in the instruction queue, the dispatch unit must do a partial decode on the instruction to determine its type. If the instruction is an integer, it is passed to the integer execution unit. If it is a floating-point type, it is passed to the floating-point execution unit. If it is a branch, it is processed immediately by branch folding and branch prediction functions. Instructions spend one or more cycles in each stage as they are being processed by the 750GX processor.

Stall — An occurrence when an instruction cannot proceed to the next stage. An instruction can spend multiple cycles in one stage. An integer multiply, for example, takes multiple cycles in the execute stage. When this occurs, subsequent instructions might stall.

Superscalar — A superscalar processor is one that has multiple execution units. The 750GX processor has one floating-point unit, two integer units, one load/store unit, and a system unit for miscellaneous instructions. PowerPC instructions are processed in parallel by these execution units.

Throughput — A measure of the total number of instructions that are processed by all execution units per unit of time.

Write-back — Write-back, in the context of instruction handling, occurs when a result is written into the architectural registers (typically the GPRs and FPRs). Results are written back at retirement time from the Rename Registers for most instructions. The instruction is also removed from the completion queue at this time.

## 6.2 Instruction Timing Overview

The 750GX design minimizes average instruction execution latency, the number of clock cycles it takes to fetch, decode, dispatch, and execute instructions and make the results available for a subsequent instruction. Some instructions, such as loads and stores, access memory and require additional clock cycles between the execute phase and the write-back phase. These latencies vary depending on whether the access is to cacheable or noncacheable memory, whether it hits in the L1 or L2 cache, whether the cache access generates a write-back to memory, whether the access causes a snoop hit from another device that generates additional activity, and other conditions that affect memory accesses.

The 750GX implements many features to improve throughput, such as pipelining, issuing superscalar instructions, branch folding, 2-level speculative branch handling, two types of branch prediction, and multiple execution units that operate independently and in parallel.

As an instruction passes from stage to stage in a pipelined system, multiple instruction are in various stages of execution at any given time. Also, with multiple execution units operating in parallel, more then one instruction can be completed in a single cycle.

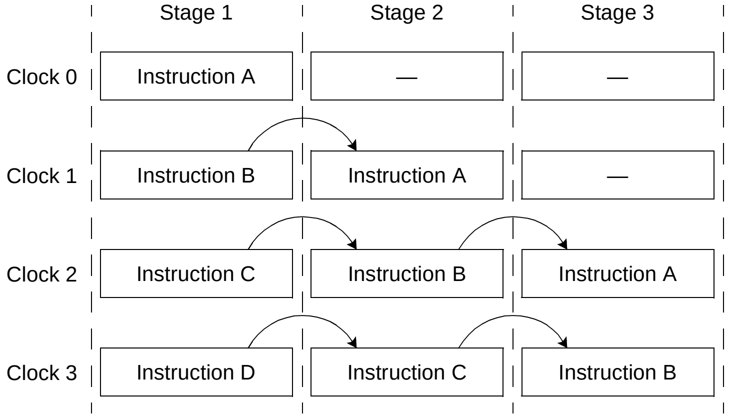
The 750GX contains the following execution units that operate independently and in parallel:

- Branch processing unit (BPU)
- Integer unit 1 (IU1)—executes all integer instructions
- Integer unit 2 (IU2)—executes all integer instructions except multiplies and divides

- 64-bit floating-point unit (FPU)
- Load/store unit (LSU)
- System register unit (SRU)

*Figure 6-1* represents a generic pipelined execution unit.

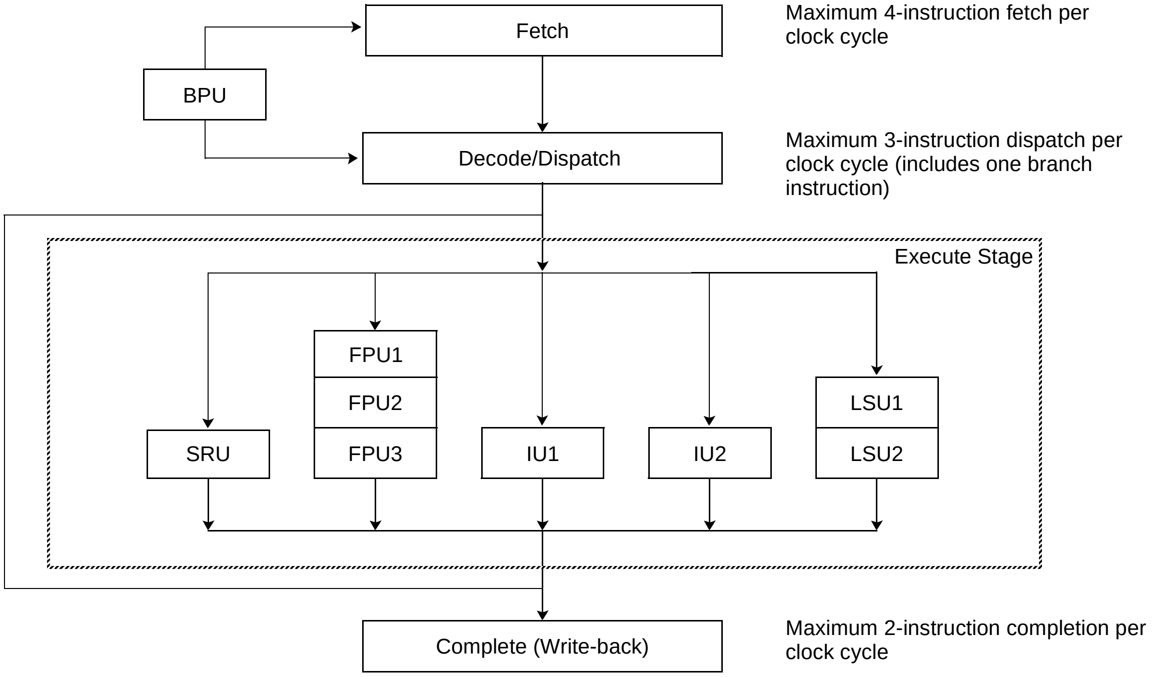
*Figure 6-1. Pipelined Execution Unit*



|  | Stage 1 | Stage 2 | Stage 3 |
|---|---|---|---|
| Clock 0 | Instruction A | — | — |
| Clock 1 | Instruction B | Instruction A | — |
| Clock 2 | Instruction C | Instruction B | Instruction A |
| Clock 3 | Instruction D | Instruction C | Instruction B |

The 750GX can retire two instructions in every clock cycle. In general, the 750GX processes instructions in four stages—fetch, decode/dispatch, execute, and complete as shown in *Figure 6-2*. Note that the example of a pipelined execution unit in *Figure 6-1* is similar to the 3-stage FPU pipeline in *Figure 6-2*.

*Figure 6-2. Superscalar/Pipeline Diagram*



Maximum 4-instruction fetch per clock cycle

Maximum 3-instruction dispatch per clock cycle (includes one branch instruction)

Maximum 2-instruction completion per clock cycle

The instruction pipeline stages are described as follows:

- The instruction fetch stage includes the clock cycles necessary to request instructions from the memory system and the time the memory system takes to respond to the request. Instruction fetch timing depends on many variables, such as whether the instruction is in the branch target instruction cache, the L1 instruction cache, or the L2 cache. If instructions must be fetched from system memory, other factors affect instruction fetch timing including the processor-to-bus clock ratio, the amount of bus traffic, and whether any cache-coherency operations are required.

  Because there are so many variables, unless otherwise specified, the instruction timing examples below assume optimal performance and assume instructions are available in the instruction queue in the same clock cycle that they are requested. The fetch stage ends when instructions are loaded into the instruction queue.
- The decode/dispatch stage consists of the time it takes to decode the instruction and dispatch it from the instruction queue to the appropriate execution unit. Instruction dispatch requires the following:
  - Instructions can be dispatched only from the two lowest instruction queue entries, IQ0 and IQ1.
  - A maximum of two instructions can be dispatched per clock cycle, and one additional branch instruction can be handled by the BPU.
  - Only one instruction can be dispatched to each execution unit per clock cycle.
  - There must be a vacancy in the specified execution-unit reservation station.
  - A Rename Register must be available for each destination operand specified by the instruction.
  - For an instruction to dispatch, the appropriate execution-unit reservation station must be available, and there must be an open position in the completion queue. If no entry is available, the instruction remains in the instruction queue (IQ).
- The execute stage consists of the time between dispatch to the execution unit (or reservation station) and the point at which the instruction vacates the execution unit.

  Most integer instructions have a 1-cycle latency; results of these instructions can be used in the clock cycle after an instruction enters the execution unit. However, integer multiply and divide instructions take multiple clock cycles to complete. IU1 can process all integer instructions; IU2 can process all integer instructions except multiply and divide instructions.

  The LSU and FPU are pipelined (as shown in *Figure 6-2* on page 212).
- The complete (complete/write-back) pipeline stage maintains the correct architectural machine state and commits the rename register values to the architectural registers at the proper time. If the completion logic detects an instruction containing an exception status, all subsequent instructions are cancelled; their execution results in the Rename Registers are discarded; and the correct instruction stream is fetched.

  The complete stage ends when the instruction is retired. Two instructions can be retired per cycle. Instructions are retired only from the two lowest completion queue entries, CQ0 and CQ1.

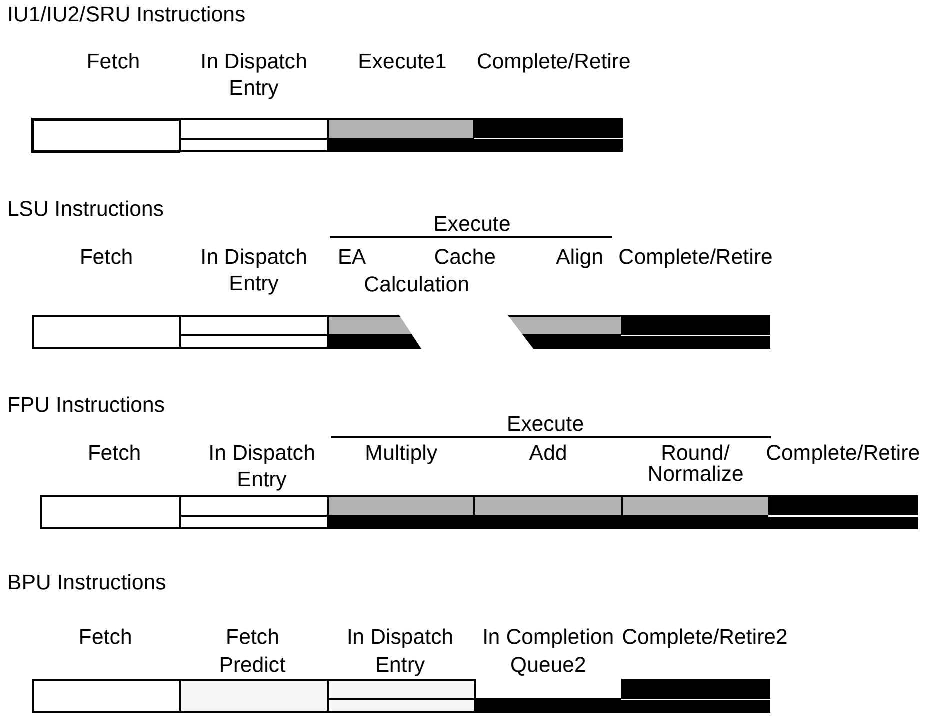
The notation conventions used in the instruction timing examples are as follows:

*Table 6-1. Notation Conventions for Instruction Timing*

| Symbol | Description |
|---|---|
| □ | Fetch. The fetch stage includes the time between when an instruction is requested and when it is brought into the instruction queue. This latency can vary, depending upon whether the instruction is in the branch target instruction cache (BTIC), the L1 instruction cache, the L2 cache, or system memory (in which case latency can be affected by bus speed and traffic on the system bus, and address-translation issues). Therefore, in the examples in this chapter, the fetch stage is usually idealized. That is, an instruction is usually shown to be in the fetch stage when it is a valid instruction in the instruction queue. The instruction queue has six entries, IQ0–IQ5. |
| ⊟ | In dispatch entry (IQ0/IQ1). Instructions can be dispatched from IQ0 and IQ1. Because dispatch is instantaneous, it is perhaps more useful to describe it as an event that marks the point in time between the last cycle in the fetch stage and the first cycle in the execute stage. |
| ▤ | Execute. The operations specified by an instruction are being performed by the appropriate execution unit. The black stripe is a reminder that the instruction occupies an entry in the completion queue, described in *Figure 6-3*. |
| ■ | Complete. The instruction is in the completion queue. In the final stage, the results of the executed instruction are written back, and the instruction is retired. The completion queue has six entries, CQ0–CQ5. |
| ▬ | In retirement entry. Completed instructions can be retired from CQ0 and CQ1. Like dispatch, retirement is an event that, in this case, occurs at the end of the final cycle of the complete stage. |

*Figure 6-3* shows the stages of the 750GX’s execution units.

*Figure 6-3. PowerPC 750GX Microprocessor Pipeline Stages*



1. Several integer instructions, such as multiply and divide instructions, require multiple cycles in the execute stage.
2. Only those branch instructions that update the LR or CTR take an entry in the completion queue.

## 6.3 Timing Considerations

The 750GX is a superscalar processor; as many as three instructions can be issued to the execution units (one branch instruction to the branch processing unit, and two instructions issued from the dispatch queue to the other execution units) during each clock cycle. Only one instruction can be dispatched to each execution unit.

Although instructions appear to the programmer to execute in program order, the 750GX improves performance by executing multiple instructions at a time, using hardware to manage dependencies. When an instruction is dispatched, the register file or a Rename Register from a previous instruction provides the source data to the execution unit. The register files and Rename Register have sufficient bandwidth to allow dispatch of two instructions per clock under most conditions.

The 750GX's BPU decodes and executes branches immediately after they are fetched. When a conditional branch cannot be resolved due to a CR data (or any) dependency, the branch direction is predicted and execution continues on the predicted path. If the prediction is incorrect, the following steps are taken:

1. The instruction queue is purged and fetching continues from the correct path.
2. Any instructions behind (in program order) the predicted branch in the completion queue are allowed to complete.
3. Instructions fetched on the mispredicted path of the branch are purged.
4. Fetching resumes along the correct (other) path.

After an execution unit finishes executing an instruction, it places resulting data into the appropriate GPR or FPR Rename Register. The results are then stored into the correct GPR or FPR during the write-back stage (retirement). If a subsequent instruction needs the result as a source operand, it is made available simultaneously to the appropriate execution unit, which allows a data-dependent instruction to be decoded and dispatched without waiting to read the data from the register file. Branch instructions that update either the LR or CTR write back their results in a similar fashion.

*Section 6.3.1* describes this process in greater detail.

### 6.3.1 General Instruction Flow

As many as four instructions can be fetched into the instruction queue (IQ) in a single clock cycle. Instructions enter the IQ and are issued to the various execution units from the dispatch queue. The 750GX tries to keep the IQ full at all times, unless instruction-cache throttling is operating.

The number of instructions requested in a clock cycle is determined by the number of vacant spaces in the IQ during the previous clock cycle. This is shown in the examples in this section. Although the instruction queue can accept as many as four new instructions in a single clock cycle, if only one IQ entry is vacant, only one instruction is fetched. Typically, instructions are fetched from the L1 instruction cache, but they might also be fetched from the branch target instruction cache (BTIC) if a branch is taken. If the branch taken instruction request hits in the BTIC, it can usually present the first two instructions of the new instruction stream in the next clock cycle, giving enough time for the next pair of instructions to be fetched from the instruction L1 cache. This results in no idle cycles in the instruction stream (also known as a zero-cycle branch). If instructions are not in the BTIC or the L1 instruction cache, they are fetched from the L2 cache or from system memory.

The 750GX’s instruction-cache throttling feature, managed through the Instruction Cache Throttling Control (ICTC) register, can lower the processor’s overall junction temperature by slowing the instruction fetch rate. See *Chapter 10, Power and Thermal Management,* on page 335 for more information.

Branch instructions are identified by the fetcher, and forwarded to the BPU directly, bypassing the dispatch queue. If the branch is unconditional or if the specified conditions are already known, the branch can be resolved immediately. That is, the branch direction is known and instruction fetching can continue along the correct path. Otherwise, the branch direction must be predicted. The 750GX offers several resources to aid in the quick resolution of branch instructions and to improve the accuracy of branch predictions. These include:

Branch target instruction cache
The 64-entry (4-way-associative) branch target instruction cache (BTIC) holds branch target instructions so when a branch is encountered in a repeated loop, usually the first two instructions in the target stream can be fetched into the instruction queue on the next clock cycle. The BTIC can be disabled and invalidated through bits in Hardware-Implementation-Dependent Register 0 (HID0). Coherency of the BTIC table is maintained by table reset on an instruction-cache flash invalidate, Instruction Cache Block Invalidate (**icbi**) or Return from Interrupt (**rfi**) instruction execution, or when an exception is taken.

Dynamic branch prediction
The 512-entry branch history table (BHT) is implemented with two bits per entry for four degrees of prediction—not-taken, strongly not-taken, taken, strongly taken. Whether a branch instruction is taken or not-taken can change the strength of the next prediction. This dynamic branch prediction is not defined by the PowerPC Architecture.

To reduce aliasing, only predicted branches update the BHT entries. Dynamic branch prediction is enabled by setting HID0[BHT]; otherwise, static branch prediction is used.

Static branch prediction
Static branch prediction is defined by the PowerPC Architecture and is encoded in the branch instructions. See *Static Branch Prediction on page 229*.

Branch instructions that do not update the LR or CTR are removed from the instruction stream by branch folding, as described in *Section 6.4.1.1, Branch Folding,* on page 226. Branch instructions that update the LR or CTR are treated as if they require dispatch (even through they are not issued to an execution unit in the process). They are assigned a position in the completion queue to ensure that the CTR and LR are updated in the correct program order.

All other instructions are issued from the IQ0 and IQ1. The dispatch rate depends upon the availability of resources such as the execution units, Rename Registers, and completion queue entries, and upon the serializing behavior of some instructions. Instructions are dispatched in program order; an instruction in IQ1 cannot be dispatched ahead of one in IQ0.

### 6.3.2 Instruction Fetch Timing

Instruction fetch latency depends on whether the fetch hits the BTIC, the L1 instruction cache, or the L2 cache. If no cache hit occurs, a memory transaction is required in which case fetch latency is affected by bus traffic, bus clock speed, and memory translation. These issues are discussed further in the following sections.

### *6.3.2.1 Cache Arbitration*

When the instruction fetcher requests instructions from the instruction cache, two things might happen. If the instruction cache is idle and the requested instructions are present, they are provided on the next clock cycle. However, if the instruction cache is busy due to a cache-line-reload operation, instructions cannot be fetched until that operation completes.

### *6.3.2.2 Cache Hit*

If the instruction fetch hits the instruction cache, it takes only one clock cycle after the request for as many as four instructions to enter the instruction queue. Note that the cache is not blocked to internal accesses while a cache reload completes (hits under misses). The critical double word is written simultaneously to the cache and forwarded to the requesting unit, minimizing stalls due to load delays.

*Figure 6-4* on page 218 shows the paths taken by instructions.

*Figure 6-4. Instruction Flow Diagram*

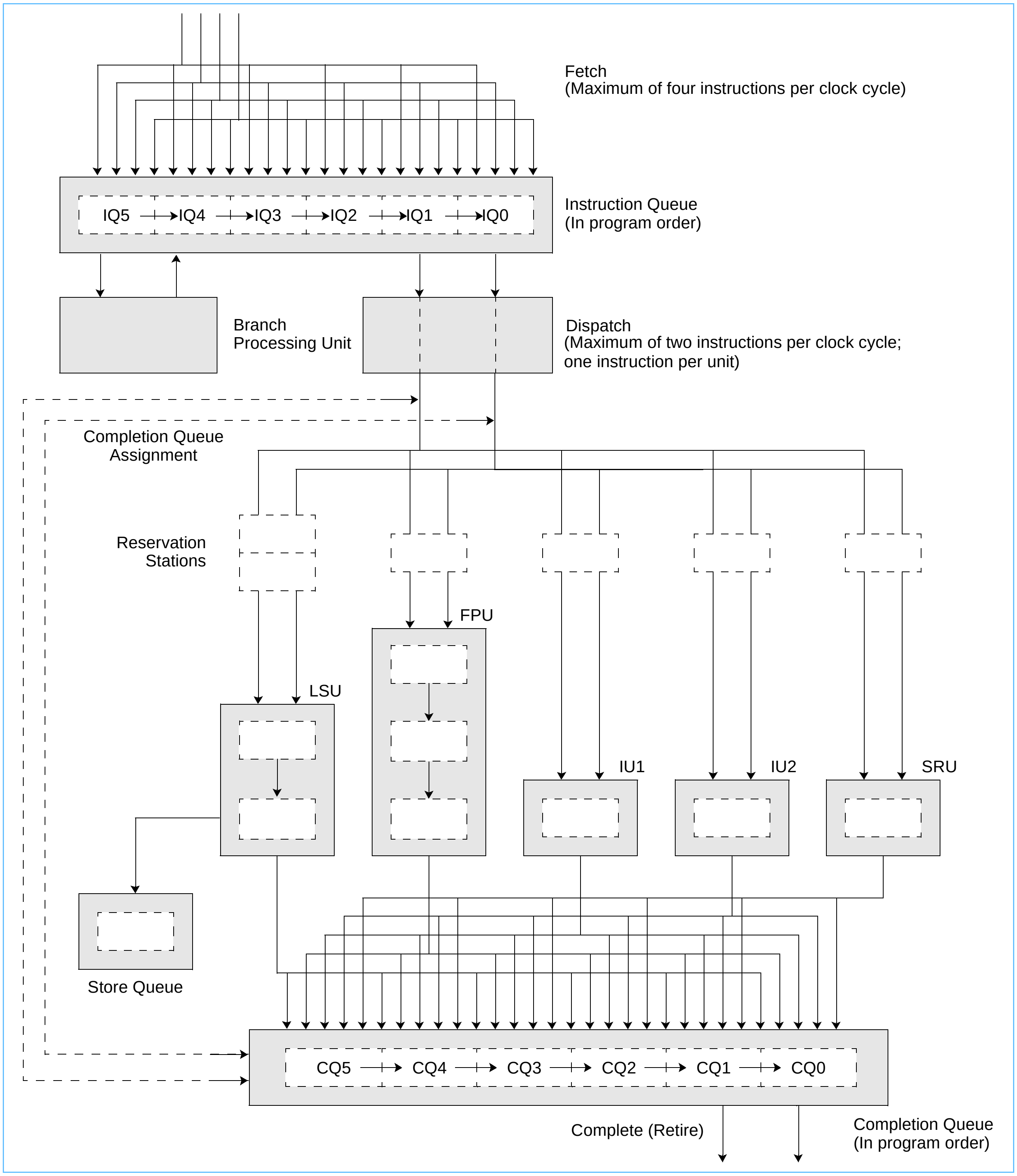
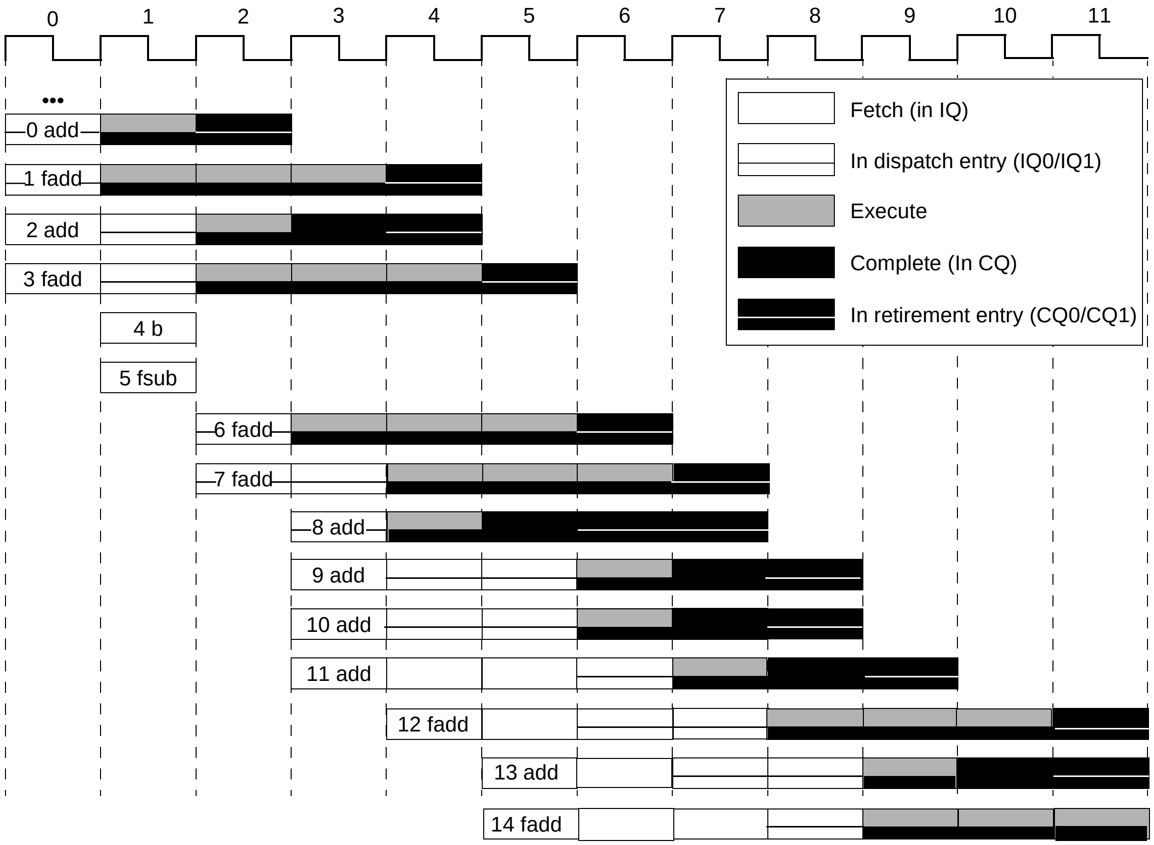


*Figure 6-5* on page 220 shows a simple example of instruction fetching that hits in the L1 cache. This example uses a series of integer add and double-precision floating-point add instructions to show how the number of instructions to be fetched is determined, how program order is maintained by the instruction and completion queues, how instructions are dispatched and retired in pairs (maximum), and how the FPU, IU1, and IU2 pipelines function. The following instruction sequence is examined.

```
 0 add
 1 fadd
 2 add
 3 fadd
 4 b 6
 5 fsub
 6 fadd
 7 fadd
 8 add
 9 add
10 add
11 add
12 fadd
13 add
14 fadd
15 •
16 •
17 •
```

*Figure 6-5. Instruction Timing—Cache Hit*



| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Instruction Queue | | | | | | 14 | | | | | | |
| | | | | 11 | | 13 | | (16) | (17) | | | |
| | 3 | 5 | | 10 | 12 | 12 | 14 | (15) | (16) | (18) | | |
| | 2 | 4 | | 9 | 11 | 11 | 13 | 14 | (15) | (17) | | |
| | 1 | 3 | 7 | 8 | 10 | 10 | 12 | 13 | 14 | (16) | | |
| | 0 | 2 | 6 | 7 | 9 | 9 | 11 | 12 | 13 | (15) | | |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Completion Queue | | | | | 8 | | | | | | | |
| | | | | | 7 | | 10 | 11 | | | | |
| | | | 3 | 6 | 6 | 8 | 9 | 10 | 12 | 14 | | |
| | | | 2 | 3 | 3 | 7 | 8 | 9 | 11 | 13 | 14 | 14 |
| | | 1 | 1 | 2 | 2 | 6 | 7 | 8 | 10 | 12 | 13 | 13 |
| | | 0 | 0 | 1 | 1 | 3 | 6 | 7 | 9 | 11 | 12 | 12 |

The instruction timing for this example is described cycle-by-cycle as follows:

1. In cycle 0, instructions 0–3 are fetched from the instruction cache. Instructions 0 and 1 are placed in the two entries in the instruction queue from which they can be dispatched on the next clock cycle.
2. In cycle 1, instructions 0 and 1 are dispatched to the IU2 and FPU, respectively. Notice that, for instructions to be dispatched, they must be assigned positions in the completion queue. In this case, since the completion queue was empty, instructions 0 and 1 take the two lowest entries in the completion queue. Instructions 2 and 3 drop into the two dispatch positions in the instruction queue. Because there were two positions available in the instruction queue in clock cycle 0, two instructions (4 and 5) are fetched into the instruction queue. Instruction 4 is a branch unconditional instruction, which resolves immediately as taken. Because the branch is taken, it can therefore be folded from the instruction queue.
3. In cycle 2, assume a BTIC hit occurs and target instructions 6 and 7 are fetched into the instruction queue, replacing the folded **b** instruction (4) and instruction 5. Instruction 0 completes, writes back its results, and vacates the completion queue by the end of the clock cycle. Instruction 1 enters the second FPU execute stage; instruction 2 is dispatched to the IU2; and instruction 3 is dispatched into the first FPU execute stage. Because the taken branch instruction (4) does not update either CTR or LR, it does not require a position in the completion queue and can be folded.
4. In cycle 3, target instructions (6 and 7) are fetched, replacing instructions 4 and 5 in IQ0 and IQ1. This replacement on taken branches is called branch folding. Instruction 1 proceeds through the last of the three FPU execute stages. Instruction 2 has executed, but must remain in the completion queue until instruction 1 completes. Instruction 3 replaces instruction 1 in the second stage of the FPU, and instruction 6 replaces instruction 3 in the first stage.

   Because there were four vacancies in the instruction queue in the previous clock cycle, instructions 8–11 are fetched in this clock cycle.
5. Instruction 1 completes in cycle 4, allowing instruction 2 to complete. Instructions 3 and 6 continue through the FPU pipeline. Because there were two openings in the completion queue in the previous cycle, instructions 7 and 8 are dispatched to the FPU and IU2, respectively, filling the completion queue. Similarly, because there was one opening in the instruction queue in clock cycle 3, one instruction is fetched.
6. In cycle 5, instruction 3 completes, and instructions 13 and 14 are fetched. Instructions 6 and 7 continue through the FPU pipeline. No instructions are dispatched in this clock cycle because there were no vacant CQ entries in cycle 4.
7. In cycle 6, instruction 6 completes, instruction 7 is in stage 3 of the FPU execute stage, and although instruction 8 has executed, it must wait for instruction 7 to complete. The two integer instructions, 9 and 10, are dispatched to the IU2 and IU1, respectively. No instructions are fetched because the instruction queue was full on the previous cycle.
8. In cycle 7, instruction 7 completes, allowing instruction 8 to complete as well. Instructions 9 and 10 remain in the completion stage, since at most two instructions can complete in a cycle. Because there was one opening in the completion queue in cycle 6, instruction 11 is dispatched to the IU2. Two more instructions, 15 and 16 (which are shown only in the instruction queue), are fetched.
9. In cycle 8, instructions 9–11 are through executing. Instructions 9 and 10 complete, write back, and vacate the completion queue. Instruction 11 must wait to complete in the following cycle. Because the completion queue had one opening in the previous cycle, instruction 12 can be dispatched to the FPU. Similarly, the instruction queue had one opening in the previous cycle, so one additional instruction, 17, can be fetched.

10. In cycle 9, instruction 11 completes, instruction 12 continues through the FPU pipeline, and instructions 13 and 14 are dispatched. One new instruction, 18, can be fetched on this cycle because the instruction queue had one opening on the previous clock cycle.

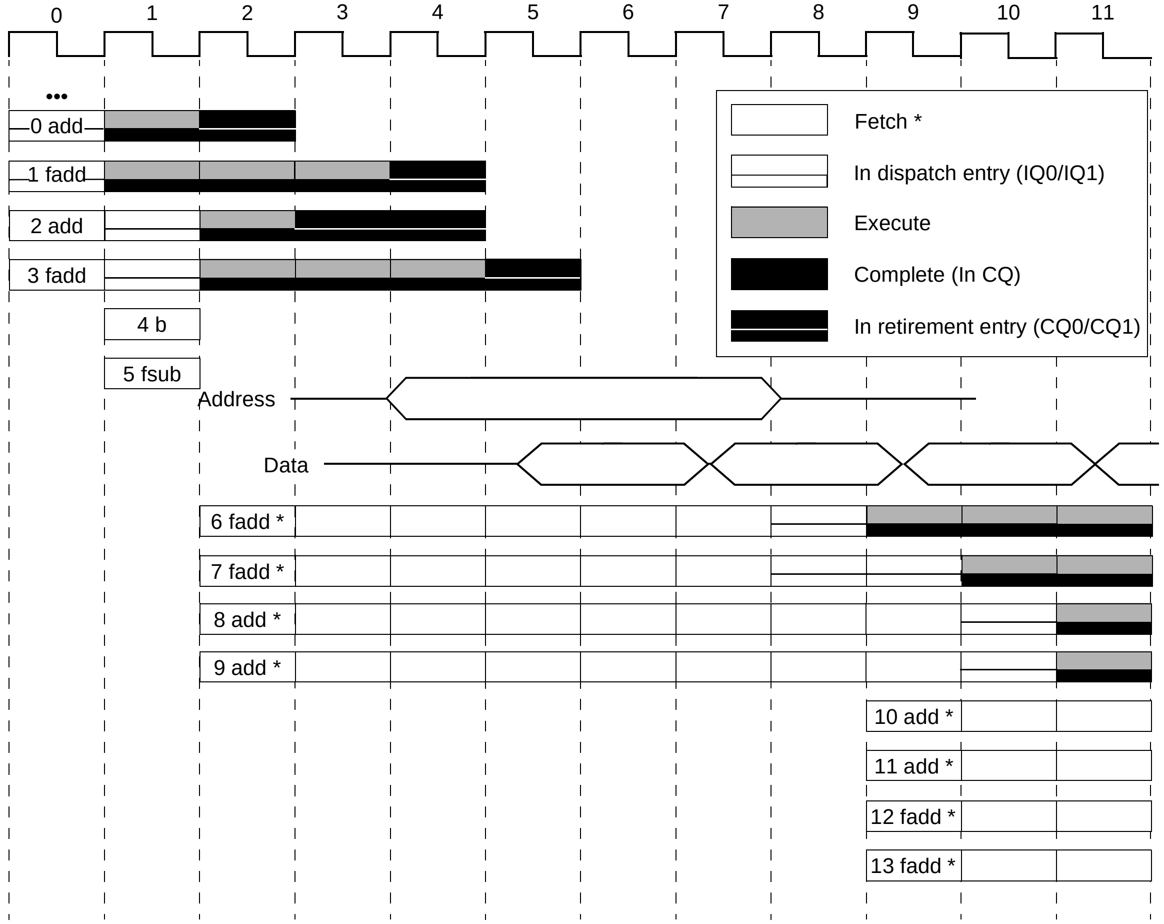
#### *6.3.2.3 Cache Miss*

*Figure 6-6* on page 223 shows an instruction fetch that misses both the L1 cache and L2 cache. A processor/bus clock ratio of 1:2 is used. The same instruction sequence is used as in *Section 6.3.2.2, Cache Hit*. However, in this example, the branch target instruction is not in either the L1 or L2 cache.

A cache miss extends the latency of the fetch stage, so, in this example, the fetch stage shown represents not only the time the instruction spends in the IQ, but the time required for the instruction to be loaded from system memory, beginning in clock cycle 2.

During clock cycle 3, the target instruction for the **b** instruction is not in the BTIC, the instruction cache, or the L2 cache; therefore, a memory access must occur. During clock cycle 5, the address of the block of instructions is sent to the system bus. During clock cycle 7, two instructions (64 bits) are returned from memory on the first beat and are forwarded both to the cache and the instruction fetcher.

*Figure 6-6. Instruction Timing—Cache Miss*



Instruction Queue

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| 3 | 5 | | | | | | | | | | |
| 2 | 4 | | | | | | | | | | |
| 1 | 3 | | | | | | | 7 | | 9 | |
| 0 | 2 | | | | | | | 6 | 7 | 8 | |

Completion Queue

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | 3 | | | | | | | | | 9 |
| | | 2 | 3 | 3 | | | | | | | 8 |
| | 1 | 1 | 2 | 2 | | | | | | 7 | 7 |
| | 0 | 0 | 1 | 1 | 3 | | | | 6 | 6 | 6 |

* Instructions 5 and 6 are not in the IQ in clock cycle 5. Here, the fetch stage shows cache latency.

### *6.3.2.4 L2 Cache Access Timing Considerations*

If an instruction fetch misses both the BTIC and the L1 instruction cache, the 750GX next looks in the L2 cache. If the requested instructions are there, they are burst into the 750GX in much the same way as shown in *Figure 6-6* on page 223.

An instruction fetch from the L2 cache has a latency of five cycles.

### *6.3.2.5 Instruction Dispatch and Completion Considerations*

Several factors affect the 750GX’s ability to dispatch instructions at a peak rate of two per cycle—the availability of the execution unit, destination Rename Registers, and completion queue, as well as the handling of completion-serialized instructions. Several of these limiting factors are illustrated in the previous instruction timing examples.

To reduce dispatch-unit stalls due to instruction data dependencies, the 750GX provides a single-entry reservation station for the FPU, SRU, and each IU, and a 2-entry reservation station for the LSU. If a data dependency keeps an instruction from starting execution, that instruction is dispatched to the reservation station associated with its execution unit (and the Rename Registers are assigned), thereby freeing the positions in the instruction queue so instructions can be dispatched to other execution units. Execution begins during the same clock cycle that the rename buffer is updated with the data the instruction is dependent on.

If both instructions in IQ0 and IQ1 require the same execution unit, they must be executed sequentially where IQ1 follows IQ0 through the execution unit. If these instructions require different execution units, they can be dispatched on the same cycle, execute in parallel on separate execution units, and could complete together and be retired together on the same cycle.

The completion unit maintains program order after instructions are dispatched from the instruction queue, guaranteeing in-order completion and a precise-exception model. Completing an instruction implies committing execution results to the architected destination registers. In-order completion ensures the correct architectural state when the 750GX must recover from a mispredicted branch or an exception.

Instruction state and all information required for completion is kept in the 6-entry, first-in/first-out completion queue. A completion queue entry is allocated for each instruction when it is dispatched to an execution unit. If no entry is available, the dispatch-unit stalls. A maximum of two instructions per cycle can be completed and retired from the completion queue, and the flow of instructions can stall when a longer-latency instruction reaches the last position in the completion queue. Subsequent instructions cannot be completed and retired until that longer-latency instruction completes and retires. Examples of this are shown in *Section 6.3.2.2, Cache Hit* and *Section 6.3.2.3, Cache Miss*.

The 750GX can execute instructions out-of-order, but in-order completion by the completion unit ensures a precise-exception mechanism. Program-related exceptions are signaled when the instruction causing the exception reaches the last position in the completion queue. By this time, previous instructions are retired.

### *6.3.2.6 Rename Register Operation*

To avoid contention for a given register file location in the course of out-of-order execution, the 750GX provides Rename Registers for holding instruction results before the completion commits them to the architected register. There are six GPR Rename Registers, six FPR Rename Registers, and one each for the CR, LR, and CTR.

When the dispatch unit dispatches an instruction to its execution unit, it allocates a Rename Register (or registers) for the results of that instruction. If an instruction is dispatched to a reservation station associated with an execution unit due to a data dependency, the dispatcher also provides a tag to the execution unit identifying the Rename Register that forwards the required data at completion. When the source data reaches the rename register, execution can begin.

Instruction results are transferred from the Rename Registers to the architected registers by the completion unit when an instruction is retired from the completion queue, provided no exceptions precede it and any predicted branch conditions have been resolved correctly. If a branch prediction was incorrect, the instructions fetched along the predicted path are flushed from the completion queue, and any results of those instructions are flushed from the Rename Registers.

### *6.3.2.7 Instruction Serialization*

Although the 750GX can dispatch and complete two instructions per cycle, so-called serializing instructions limit dispatch and completion to one instruction per cycle. There are three types of instruction serialization:

Execution — Execution-serialized instructions are dispatched, held in the functional unit, and do not execute until all prior instructions have completed. A functional unit holding an execution-serialized instruction will not accept further instructions from the dispatcher. For example, execution serialization is used for instructions that modify nonrenamed resources. Results from these instructions are generally not available or forwarded to subsequent instructions until the instruction completes (using a Move-to Special Purpose Register [**mtspr**] instruction to write to an LR or CTR does provide forwarding to branch instructions).

Completion (also referred to as post-dispatch or tail serialization) — Completion-serialized instructions inhibit dispatching of subsequent instructions until the serialized instruction completes. Completion serialization is used for instructions that bypass the normal rename mechanism.

Refetch (flush) — Refetch-serialized instructions inhibit dispatch of subsequent instructions and force refetching of subsequent instructions after completion.

## 6.4 Execution-Unit Timings

The following sections describe instruction timing considerations within each of the respective execution units in the 750GX.

### 6.4.1 Branch Processing Unit Execution Timing

Flow-control operations (conditional branches, unconditional branches, and traps) are typically expensive to execute in most machines because they disrupt normal flow in the instruction stream. When a change in program flow occurs, the IQ must be reloaded with the target instruction stream. Previously issued instructions will continue to execute while the new instruction stream makes its way into the IQ. However, depending on whether the target instruction is in the BTIC, instruction L1 cache, L2 cache, or in system memory, some opportunities might be missed to execute instructions. The example in *Section 6.3.2.3, Cache Miss,* on page 222 illustrates this situation.

Performance features such as branch folding, BTIC, dynamic branch prediction (implemented in the BHT), 2-level branch prediction, and the implementation of nonblocking caches minimize the penalties associated with flow-control operations on the 750GX. The timing for branch instruction execution is determined by many factors including:

- Whether the branch is taken
- Whether instructions in the target stream, typically the first two instructions in the target stream, are in the branch target instruction cache (BTIC)
- Whether the target instruction stream is in the L1 cache
- Whether the branch is predicted
- Whether the prediction is correct

#### *6.4.1.1 Branch Folding*

When a branch instruction is encountered by the fetcher, the BPU immediately begins to decode it and tries to resolve it. Branch folding is the removal of branches from the instruction stream. This is independent of whether the branch is taken or not taken. However, if the branch instruction updates either the LR or CTR it cannot be removed and must be allocated a position in the completion queue. If a branch cannot be resolved immediately, it is predicted and instruction fetching resumes along the predicted path. Those instructions are conditionally fed into the instruction queue. Later, if the prediction is finally correctly resolved, the fetched instructions are validated and allowed to complete and be retired. If the prediction is resolved incorrectly, then the instructions fetched are invalidated, and instruction fetching resumes along the other path of the branch.

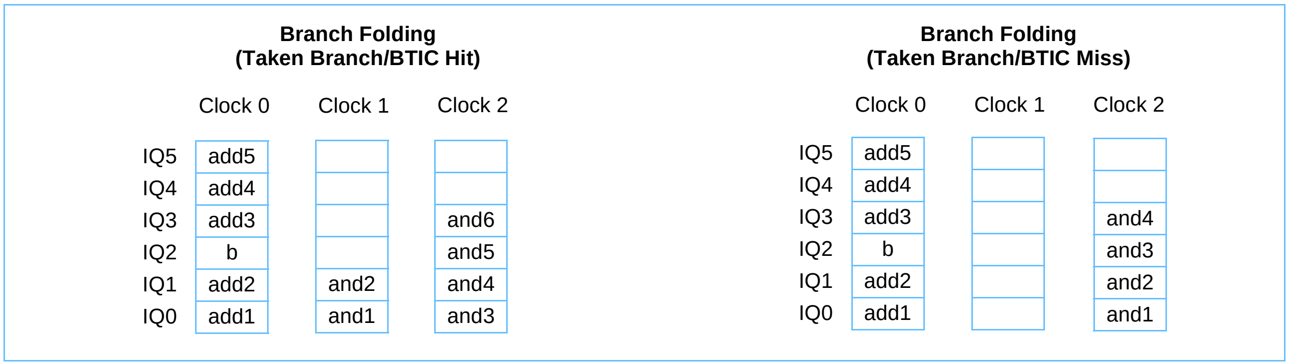
*Figure 6-7* on page 227 shows branch folding. Here a **b** instruction is encountered in a series of **add** instructions. The branch is resolved as taken. What happens on the next clock cycle depends on whether the target instruction stream is in the BTIC, the instruction L1 cache, or if it must be fetched from the L2 cache or from system memory.

*Figure 6-7* shows cases where there is a BTIC hit, and where there is a BTIC miss (and instruction-cache hit). If there is a BTIC hit on the next clock cycle, the **b**x instruction is replaced by the target instruction, **and**1, which was found in the BTIC. The second **and** instruction is also fetched from the BTIC. On the next clock cycle, the next four **and** instructions from the target stream are fetched from the instruction cache.

If the target instruction is not in the BTIC, there is an idle cycle while the fetcher attempts to fetch the first four instructions from the instruction cache (on the next clock cycle). In the example in *Figure 6-7*, the first four target instruction are fetched on the next clock.

If the target instruction misses in the BTIC or L1 caches, an L2 cache or memory access is required. The latency of this access is dependent on several factors, such as processor/bus clock ratios. In most cases, new instructions arrive in the IQ before the execution units become idle.

*Figure 6-7. Branch Taken*



**Branch Folding**
**(Taken Branch/BTIC Hit)**

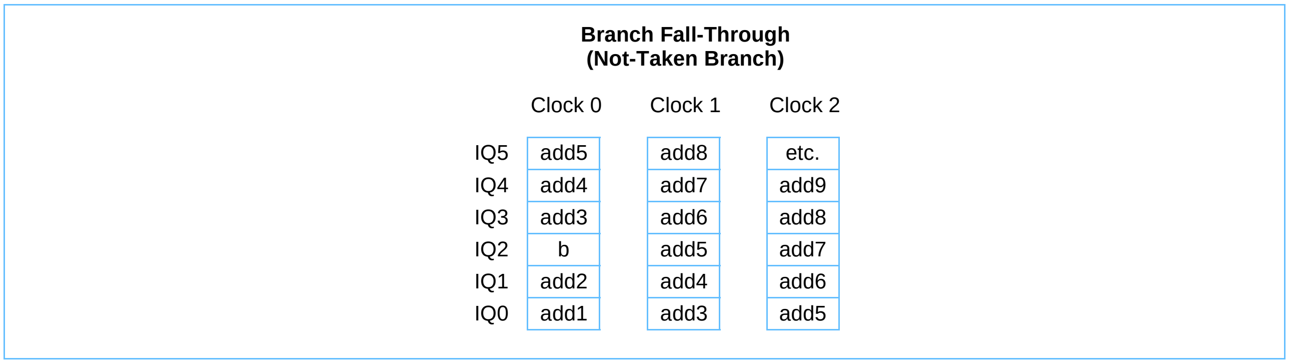
|  | Clock 0 | Clock 1 | Clock 2 |
|---|---|---|---|
| IQ5 | add5 |  |  |
| IQ4 | add4 |  |  |
| IQ3 | add3 |  | and6 |
| IQ2 | b |  | and5 |
| IQ1 | add2 | and2 | and4 |
| IQ0 | add1 | and1 | and3 |

**Branch Folding**
**(Taken Branch/BTIC Miss)**

|  | Clock 0 | Clock 1 | Clock 2 |
|---|---|---|---|
| IQ5 | add5 |  |  |
| IQ4 | add4 |  |  |
| IQ3 | add3 |  | and4 |
| IQ2 | b |  | and3 |
| IQ1 | add2 |  | and2 |
| IQ0 | add1 |  | and1 |

*Figure 6-8* shows the removal of fall-through branch instructions, which occurs when a branch is not taken or is predicted as not taken.

*Figure 6-8. Removal of Fall-Through Branch Instruction*



**Branch Fall-Through**
**(Not-Taken Branch)**

|  | Clock 0 | Clock 1 | Clock 2 |
|---|---|---|---|
| IQ5 | add5 | add8 | etc. |
| IQ4 | add4 | add7 | add9 |
| IQ3 | add3 | add6 | add8 |
| IQ2 | b | add5 | add7 |
| IQ1 | add2 | add4 | add6 |
| IQ0 | add1 | add3 | add5 |

When a branch instruction is detected before it reaches a dispatch position, and if the branch is correctly predicted as taken, folding the branch instruction (and any instructions from the incorrect path) reduces the latency required for flow control to zero. Instruction execution proceeds as though the branch was never there.

The advantage of removing the fall-through branch instructions at dispatch is only marginally less than that of branch folding. Because the branch is not taken, only the branch instruction needs to be discarded. The only cost of expelling the branch instruction from one of the dispatch entries rather than folding it is missing a chance to dispatch an executable instruction from that position.

### *6.4.1.2 Branch Instructions and Completion*

As described in the previous section, instructions that do not update either the LR or CTR are removed from the instruction stream before they reach the completion queue, either for branch taken or by removing fall-through branch instructions at dispatch. However, branch instructions that update the architected LR and CTR must do so in program order. Therefore, they must perform write-back in the completion stage, like the instructions that update the FPRs and GPRs.

Branch instructions that update the CTR or LR pass through the instruction queue like nonbranch instructions. At the point of dispatch, however, they are not sent to an execution unit, but rather are assigned a slot in the completion queue, as shown in *Figure 6-9* on page 228.

*Figure 6-9. Branch Completion*

**Branch Completion**
**(LR/CTR Write-Back)**

| | Clock 0 | Clock 1 | Clock 2 | Clock 3 |
|---|---|---|---|---|
| IQ5 | add5 | | | |
| IQ4 | add4 | | | |
| IQ3 | add3 | add5 | add7 | add9 |
| IQ2 | bc | add4 | add6 | add8 |
| IQ1 | add2 | add3 | add5 | add7 |
| IQ0 | add1 | bc | add4 | add6 |
| CQ5 | | | | |
| CQ4 | | | | |
| CQ3 | | | | |
| CQ2 | | | | |
| CQ1 | | add2 | add3 | add5 |
| CQ0 | | add1 | bc | add4 |

In this example, the Branch Conditional (**bc**) instruction is encoded to decrement the CTR. It is predicted as not-taken in clock cycle 0. In clock cycle 2, **bc** and **add**3 are both dispatched. In clock cycle 3, the architected CTR is updated, and the **bc** instruction is retired from the completion queue.

### *6.4.1.3 Branch Prediction and Resolution*

The 750GX supports the following two types of branch prediction:

- Static branch prediction—This is defined by the PowerPC Architecture as part of the encoding of branch instructions.
- Dynamic branch prediction—This is a processor-specific mechanism implemented in hardware (in particular the branch history table, or BHT) that monitors branch instruction behavior and maintains a record from which the next occurrence of the branch instruction is predicted.

When a conditional branch cannot be resolved due to a CR data dependency, the BPU predicts whether it will be taken, and instruction fetching proceeds down the predicted path. If the branch prediction resolves as incorrect, the instruction queue and all subsequently executed instructions are purged, instructions executed prior to the predicted branch are allowed to complete, and instruction fetching resumes down the correct path.

The 750GX executes through two levels of prediction. Instructions from the first unresolved branch can execute, but they cannot be retired until the branch is resolved. If a second branch instruction is encountered in the predicted instruction stream, it can be predicted and instructions can be fetched, but not executed, from the second branch. No action can be taken for a third branch instruction until at least one of the two previous branch instructions is resolved.

The number of instructions that can be executed after the issue of a predicted branch instruction is limited by the fact that no instruction executed after a predicted branch can actually update (be retired) the register files or memory until the branch is resolved. That is, instructions can be issued and executed, but cannot be retired from the completion unit. When an instruction following a predicted branch completes execution, it does not write back its results to the architected registers. Instead, it stalls in the completion queue. Of course, when the completion queue is full, no additional instructions can be dispatched, even if an execution unit is idle.

In the case of a misprediction, the 750GX can easily redirect the instruction stream because the programming model has not been updated. When a branch is mispredicted, all instructions that were dispatched after the predicted branch instruction are flushed from the completion queue, and any results are flushed from the Rename Registers.

The BTIC is a cache of two recently used instructions at the target (branch-to address) of branch instructions. If a taken-branch hits in the BTIC, two instructions are fed into the instruction queue on the next cycle. If a taken-branch misses in the BTIC, instruction fetching is done from the L1 instruction cache. Coherency of the BTIC table is maintained by table reset on an instruction-cache flash invalidate, **icbi** or **rfi** instruction execution, or when an exception is taken.

In some situations, an instruction sequence creates dependencies that keep a branch instruction from being predicted because the address for the target of the branch is not available. This delays execution of the subsequent instruction stream. The instruction sequences and the resulting action of the branch instruction are as follows:

- An **mtspr**(LR) followed by a Branch Conditional to Link Register (**bclr**)—Fetching stops, and the branch waits for the **mtspr** to execute.
- An **mtspr**(CTR) followed by a Branch Conditional to Count Register (**bcctr**)—Fetching stops, and the branch waits for the **mtspr** to execute.
- An **mtspr**(CTR) followed by a **bc** (CTR decrement)—Fetching stops, and the branch waits for the **mtspr** to execute.
- A third **bc** (based on CR) is encountered while there are two unresolved **bc** (based on CR) instructions. The third **bc** (based on CR) is not executed, and fetching stops until one of the previous **bc** (based on CR) instructions is resolved. (Note that branch conditions can be a function of the CTR and the CR; if the CTR condition is sufficient to resolve the branch, then a CR-dependency is ignored.)

#### *Static Branch Prediction*

The PowerPC Architecture provides a field in branch instructions (the BO field) to allow software to speculate (hint) whether a branch is likely to be taken. Rather than delaying instruction processing until the condition is known, the 750GX uses the instruction encoding to predict whether the branch is likely to be taken and begins fetching and executing along that path. When the branch condition is known, the prediction is evaluated. If the prediction was correct, program flow continues along that path. Otherwise, the processor flushes any instructions and their results from the mispredicted path, and program flow resumes along the correct path.

Static branch prediction is used when HID0[BHT] is cleared. That is, the branch history table, which is used for dynamic branch prediction, is disabled.

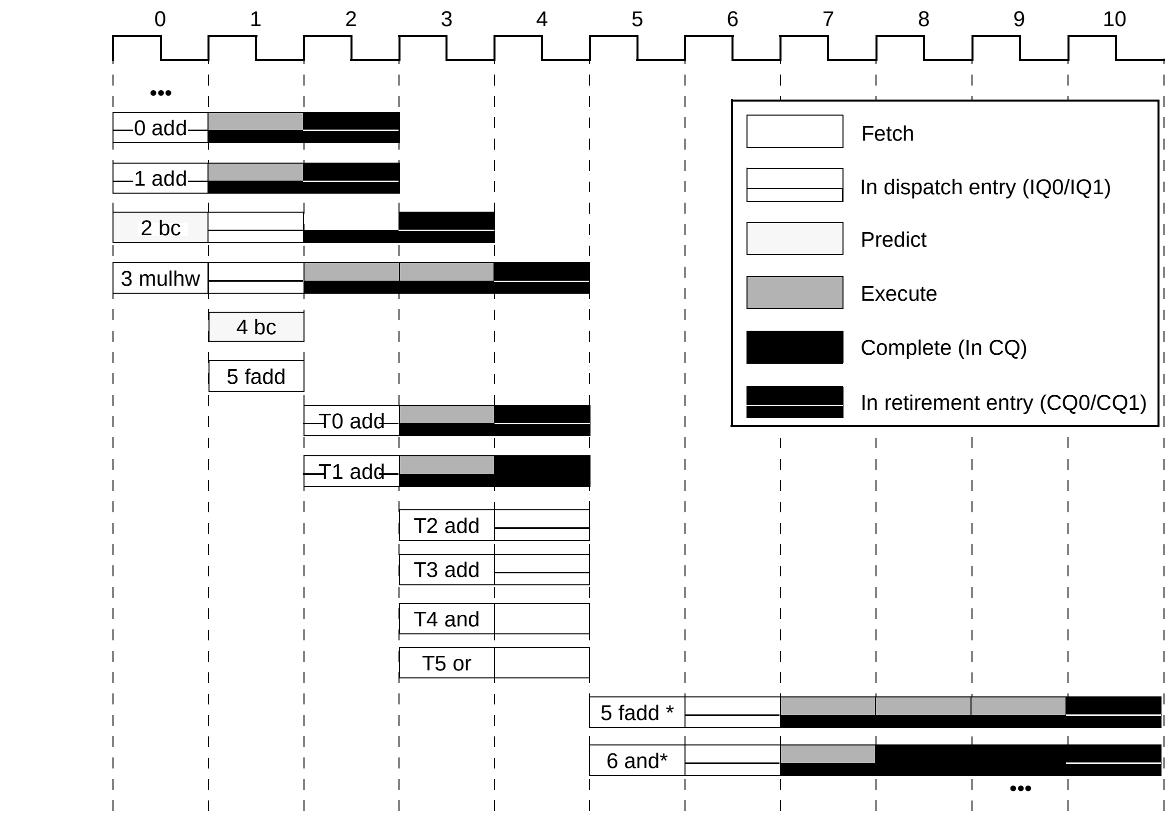
For information about static branch prediction, see “Conditional Branch Control,” in Chapter 4, “Addressing Modes and Instruction Set Summary” in the *PowerPC Microprocessor Family: The Programming Environments Manual*.

*Predicted Branch Timing Examples*

*Figure 6-10* on page 231 shows cases where branch instructions are predicted. It shows how both taken and not-taken branches are handled, and how the 750GX handles both correct and incorrect predictions. The example shows the timing for the following instruction sequence:

```
0  add
1  add
2  bc
3  mulhw
4  bc T0
5  fadd
6  and
7  or
8  sub
•
•
•
 T0 add
 T1 add
 T2 add
 T3 add
 T4 and
 T5 or
```

*Figure 6-10. Branch Instruction Timing*



Instruction Queue

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | |
| | | | | | | | | | | |
| 3 | 5 | | T5 | T5 | | (8) | | | | |
| 2 (bc) | 4 | | T4 | T4 | | (7) | | | | |
| 1 | 3 | T1 | T3 | T3 | | 6 | | | | |
| 0 | 2 | T0 | T2 | T2 | | 5 | | | | |

Completion Queue

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | 3 | T1 | | | | | (8) | (8) | (8) |
| | | 2 | T0 | T1 | | | | (7) | (7) | (7) |
| | 1 | 1 | 3 | T0 | | | 6 | 6 | 6 | 6 |
| | 0 | 0 | 2 | 3 | | | 5 | 5 | 5 | 5 |

* Instructions 5 and 6 are not in the IQ in clock cycle 5. Here, the fetch stage shows cache latency.

1. During clock cycle 0, instructions 0 and 1 are dispatched to their respective execution units. Instruction 2 is a branch instruction that updates the CTR. It is predicted as not taken in clock cycle 0. Instruction 3 is a Multiply High Word (**mulhw**) instruction on which instruction 4 depends.

2. In clock cycle 1, instructions 2 and 3 enter the dispatch entries in the IQ. Instruction 4 (a second **bc** instruction) and 5 are fetched. The second **bc** instruction is predicted as taken. It can be folded, but it cannot be resolved until instruction 3 writes back.
3. In clock cycle 2, instruction 4 has been folded and instruction 5 has been flushed from the IQ. The two target instructions, T0 and T1, are both in the BTIC, so they are fetched in this cycle. Note that, even though the first **bc** instruction might not have resolved by this point (we can assume it has), the 750GX allows fetching from a second predicted branch stream. However, these instructions could not be dispatched until the previous branch has resolved.
4. In clock cycle 3, target instructions T2–T5 are fetched as T0 and T1 are dispatched.
5. In clock cycle 4, instruction 3, on which the second branch instruction depended, writes back, and the branch prediction is proven incorrect. Even though T0 is in CQ1, from which it could be written back, it is not written back because the branch prediction was incorrect. All target instructions are flushed from their positions in the pipeline at the end of this clock cycle, as are any results in the Rename Registers.

After one clock cycle required to refetch the original instruction stream, instruction 5, the same instruction that was fetched in clock cycle 1, is brought back into the IQ from the instruction cache, along with three others (not all of which are shown).

### 6.4.2 Integer Unit Execution Timing

The 750GX has two integer units. The IU1 can execute all integer instructions; the IU2 can execute all integer instructions except multiply and divide instructions. As shown in *Figure 6-2* on page 212, each integer unit has one execute pipeline stage. Thus, when a multicycle (for example, divide) integer instruction is being executed, no additional integer instruction can begin to execute in that unit. However, the other unit IU2 can continue to execute integer instructions. *Table 6-7* on page 240 lists integer instruction latencies. Most integer instructions have an execution latency of one clock cycle.

### 6.4.3 Floating-Point Unit Execution Timing

The floating-point unit on the 750GX executes all floating-point instructions. Execution of most floating-point instructions is pipelined within the FPU, allowing up to three instructions to execute in the FPU concurrently. While most floating-point instructions execute with 3-cycle or 4-cycle latency, and 1-cycle or 2-cycle throughput, two instructions, **fdivs** and **fdiv**, execute with latencies of 11 to 33 cycles. The following instructions block the floating-point unit pipeline until they complete execution:

- Floating Divide Single (**fdivs**)
- Floating Divide (**fdiv**)
- Move-to Floating-Point Status and Control Register [FPSCR] Bit 0 (**mtfsb0**)
- Move-to FPSCR Bit 1(**mtfsb1**)
- Move-to FPSCR Field Immediate (**mtfsfi**)
- Move-from FPSCR (**mffs**)
- Move-to FPSCR Fields (**mtfsf**)

Thus, they inhibit the dispatch of additional floating-point instructions. See *Table 6-8* on page 242 for floating-point instruction execution timing.

### 6.4.4 Effect of Floating-Point Exceptions on Performance

For the fastest and most predictable floating-point performance, all exceptions should be disabled in the FPSCR and Machine State Register (MSR).

### 6.4.5 Load/Store Unit Execution Timing

The execution of most load-and-store instructions is pipelined. The LSU has two pipeline stages. The first is for effective address calculation and MMU translation, and the second is for accessing data in the cache. Load-and-store instructions have a 2-cycle latency and 1-cycle throughput. For instructions that store FPR values (Store Floating-Point Double [**stfd**], Store Floating-Point Single [**stfs**], and their variations), the data to be stored is prefetched from the source register during the first pipeline stage. In cases where this register is updated that same cycle, the instruction will stall to get the correct data, resulting in one additional cycle of latency.

If operands are misaligned, additional latency might be required either for an alignment exception to be taken or for additional bus accesses. Load instructions that miss in the cache block require subsequent cache accesses during the cache-line refill. *Table 6-9* on page 244 gives load-and-store instruction execution latencies.

### 6.4.6 Effect of Operand Placement on Performance

The PowerPC virtual environment architecture (VEA) states that the placement (location and alignment) of operands in memory might affect the relative performance of memory accesses, and in some cases affect it significantly. The effects memory operand placement has on performance are shown in *Table 6-2*.

The best performance is guaranteed if memory operands are aligned on natural boundaries. For the best performance across the widest range of implementations, the programmer should assume the performance model described in Chapter 3, “Operand Conventions” in the *PowerPC Microprocessor Family: The Programming Environments Manual*.

The effect of misalignment on memory-access latency is the same for big and little-endian addressing modes except for multiple and string operations that cause an alignment exception in little-endian mode.

*Table 6-2. Performance Effects of Memory Operand Placement* (Page 1 of 2)

| Operand | | Boundary Crossing | | | |
|---|---|---|---|---|---|
| Size | Byte Alignment | None | 8 Byte | Cache Block | Protection Boundary |
| **Integer** | | | | | |
| 4 byte | 4 | Optimal[1] | — | — | — |
| | < 4 | Optimal | Good[2] | Good | Good |
| 2 byte | 2 | Optimal | — | — | — |
| | < 2 | Optimal | Good | Good | Good |
| 1 byte | 1 | Optimal | — | — | — |
| Load Multiple Word (**lmw**),<br>Store Multiple Word (**stmw**)[3] | 4 | Good | Good | Good | Good |
| | < 4 | Poor[4] | Poor | Poor | Poor |
| String[3] | — | Good | Good | Good | Good |

**Note:**

1. Optimal means one EA calculation occurs.
2. Good means multiple EA calculations occur that might cause additional bus activities with multiple bus transfers.
3. Not supported in little-endian mode; causes an alignment exception.
4. Poor means that an alignment exception occurs.

*Table 6-2. Performance Effects of Memory Operand Placement* (Page 2 of 2)

| Operand | | Boundary Crossing | | | |
|---|---|---|---|---|---|
| Size | Byte Alignment | None | 8 Byte | Cache Block | Protection Boundary |
| **Floating-Point** | | | | | |
| 8 byte | 8 | Optimal | — | — | — |
| | 4 | — | Good | Good | Good |
| | < 4 | — | Poor | Poor | Poor |
| 4 byte | 4 | Optimal | — | — | — |
| | < 4 | Poor | Poor | Poor | Poor |

**Note:**

1. Optimal means one EA calculation occurs.
2. Good means multiple EA calculations occur that might cause additional bus activities with multiple bus transfers.
3. Not supported in little-endian mode; causes an alignment exception.
4. Poor means that an alignment exception occurs.

### 6.4.7 Integer Store Gathering

The 750GX performs store gathering for write-through operations to nonguarded space. It performs cache-inhibited stores to nonguarded space for 4-byte, word-aligned stores. These stores are combined in the LSU to form a double word and are sent out on the 60x bus as a single-beat operation. However, stores are gathered only if the successive stores meet the criteria and are queued and pending. Store gathering occurs regardless of the address order of the stores. Store gathering is enabled by setting HID0[SGE]. Stores can be gathered in both endian modes.

Store gathering is not done for:

- Cacheable store operations
- Stores to guarded cache-inhibited or write-through space
- Byte-reverse store operations
- Store Word Conditional Indexed (**stwcx.**) instructions
- External Control Out Word Indexed (**ecowx**) instructions
- A store that occurs during a table-search operation
- Floating-point store operations

If store gathering is enabled and the stores do not fall under the above categories, an Enforce In-Order Execution of I/O (**eieio**) or Synchronize (**sync**) instruction must be used to prevent two stores from being gathered.

### 6.4.8 System Register Unit Execution Timing

Most instructions executed by the SRU either directly access renamed registers or access or modify nonrenamed registers. They generally execute in a serial manner. Results from these instructions are not available to subsequent instructions until the instruction completes and is retired. See *Section 6.3.2.7, Instruction Serialization* for more information on serializing instructions executed by the SRU, and see *Table 6-5* on page 238 and *Table 6-6* on page 240 for SRU instruction execution timings.

## 6.5 Memory Performance Considerations

Because the 750GX can have a maximum instruction throughput of three instructions per clock cycle, lack of memory bandwidth can affect performance. For the 750GX to maximize performance, it must be able to read and write data efficiently. If a system has multiple bus devices, one of them might experience long memory latencies while another bus master (for example, a direct memory-access controller) is using the external bus.

### 6.5.1 Caching and Memory Coherency

To minimize the effect of bus contention, the PowerPC Architecture defines WIM bits that are used to configure memory regions as caching enforced or caching inhibited. Accesses to such memory locations never update the L1 cache. If a cache-inhibited access hits the L1 cache, the cache block is invalidated. If the cache block is marked modified, it is copied back to memory before being invalidated. Where caching is permitted, memory is configured as either write-back or write-through, which are described as follows:

Write-back — Configuring a memory region as write-back lets a processor modify data in the cache without updating system memory. For such locations, memory updates occur only on modified cache-block replacements, cache flushes, or when one processor needs data that is modified in another’s cache. Therefore, configuring memory as write-back can help when bus traffic could cause bottlenecks, especially for multiprocessor systems and for regions in which data, such as local variables, is used often and is coupled closely to a processor.

If multiple devices use data in a memory region marked write-through, snooping must be enabled to allow the copy-back and cache invalidation operations necessary to ensure cache coherency. The 750GX’s snooping hardware keeps other devices from accessing invalid data. For example, when snooping is enabled, the 750GX monitors transactions of other bus devices. If another device needs data that is modified on the 750GX’s cache, the access is delayed so the 750GX can copy the modified data to memory.

Write-through — Store operations to memory marked write-through always update both system memory and the L1 cache on cache hits. Because valid cache contents always match system memory marked write-through, cache hits from other devices do not cause modified data to be copied back as they do for locations marked write-back. However, all write operations are passed to the bus, which can limit performance. Load operations that miss the L1 cache must wait for the external store operation.

Write-through configuration is useful when cached data must agree with external memory (for example, video memory), when shared (global) data might be needed often, or when it is undesirable to allocate a cache block on a cache miss.

*Chapter 3, Instruction-Cache and Data-Cache Operation,* on page 121 describes the caches, memory configuration, and snooping in detail.

### 6.5.2 Effect of TLB Miss

If a page-address translation is not in a translation lookaside buffer (TLB), the 750GX hardware searches the page tables and updates the TLB when a translation is found. *Table 6-3* shows the estimated latency for the hardware TLB load for different cache configurations and conditions.

*Table 6-3. TLB Miss Latencies*

| L1 Condition<br>(Instruction and Data) | L2 Condition | Processor/System Bus<br>Clock Ratio | Estimated Latency<br>(Cycles) |
|---|---|---|---|
| 100% cache hit | — | — | 7 |
| 100% cache miss | 100% cache hit | — | 13 |
| 100% cache miss | 100% cache miss | 2.5:1 (6:3:3:3 memory) | 62 |
| 100% cache miss | 100% cache miss | 4:1 (5:2:2:2 memory) | 77 |

The page table entry (PTE) table search assumes a hit in the first entry of the primary page-table-entry group (PTEG).

## 6.6 Instruction Scheduling Guidelines

The performance of the 750GX can be improved by avoiding resource conflicts and scheduling instructions to take fullest advantage of the parallel execution units. Instruction scheduling on the 750GX can be improved by observing the following guidelines:

- To reduce mispredictions, separate the instruction that sets CR bits from the branch instruction that evaluates them. Because there can be no more than 12 instructions in the processor (with the instruction that sets CR in CQ0 and the dependent branch instruction in IQ5), there is no advantage to having more than 10 instructions between them.
- Likewise, when branching to a location specified by the CTR or LR, separate the **mtspr** instruction that initializes the CTR or LR from the dependent branch instruction. This ensures the register values are available sooner to the branch instruction.
- Schedule instructions so that two can be dispatched at a time.
- Schedule instructions to minimize stalls due to execution units being busy.
- Avoid scheduling high-latency instructions close together. Interspersing single-cycle latency integer instructions between longer-latency instructions minimizes the effect that instructions such as integer and floating-point divide and multiply can have on throughput.
- Avoid using serializing instructions.
- Schedule instructions to avoid dispatch stalls.
  - Six instructions can be tracked in the completion queue. Therefore, only six instructions can be in the execute stages at any one time.
  - There are six GPR Rename Registers. Therefore, only six GPRs can be specified as destination operands at any time. If no Rename Registers are available, instructions cannot enter the execute stage and remain in the reservation station or instruction queue until they become available.

    **Note:** Load with update address instructions use two Rename Registers.
  - Similarly, there are six FPR Rename Registers, so only six FPR destination operands can be in the execute and complete stages at any time.

### 6.6.1 Branch, Dispatch, and Completion-Unit Resource Requirements

This section describes the specific resources required to avoid stalls during branch resolution, instruction dispatching, and instruction completion.

#### *6.6.1.1 Branch-Resolution Resource Requirements*

The following branch instructions and resources are required to avoid stalling the fetch unit in the course of branch resolution:

- The **bclr** instruction requires LR availability.
- The **bcctr** instruction requires CTR availability.
- Branch and link instructions require shadow LR availability.
- The “branch conditional on counter decrement and the CR” condition requires CTR availability or the CR condition must be false, and the 750GX cannot execute instructions after an unresolved predicted branch when the BPU encounters a branch.
- A branch conditional on CR condition cannot be executed following an unresolved predicted branch instruction.

#### *6.6.1.2 Dispatch-Unit Resource Requirements*

The following resources are required to avoid stalls in the dispatch unit. IQ0 and IQ1 are the two dispatch entries in the instruction queue:

- Requirements for dispatching from IQ0 are:
  - Needed execution unit available.
  - Needed GPR Rename Registers available.
  - Needed FPR Rename Registers available.
  - Completion queue is not full.
  - A completion-serialized instruction is not being executed.
- Requirements for dispatching from IQ1 are:
  - Instruction in IQ0 must dispatch.
  - Instruction dispatched by IQ0 is not completion- or refetch-serialized.
  - Needed execution unit is available after dispatch from IQ0.
  - Needed GPR Rename Registers are available after dispatch from IQ0.
  - Needed FPR Rename Register is available after dispatch from IQ0.
  - Completion queue is not full after dispatch from IQ0.

#### *6.6.1.3 Completion-Unit Resource Requirements*

The following is a list of resources required to avoid stalls in the completion unit. Note that the two completion entries are described as CQ0 and CQ1, where CQ0 is the completion queue located at the end of the completion queue (see *Figure 6-4* on page 218).

- Requirements for completing an instruction from CQ0:
  - Instruction in CQ0 must be finished.
  - Instruction in CQ0 must not follow an unresolved predicted branch.
  - Instruction in CQ0 must not cause an exception.

- Requirements for completing an instruction from CQ1:
  - Instruction in CQ0 must complete in same cycle.
  - Instruction in CQ1 must be finished.
  - Instruction in CQ1 must not follow an unresolved predicted branch.
  - Instruction in CQ1 must not cause an exception.
  - Instruction in CQ1 must be an integer or load instruction.
  - Number of CR updates from both CQ0 and CQ1 must not exceed two.
  - Number of GPR updates from both CQ0 and CQ1 must not exceed two.
  - Number of FPR updates from both CQ0 and CQ1 must not exceed two.

## 6.7 Instruction Latency Summary

*Table 6-4* through *Table 6-9* list the latencies associated with instructions executed by each execution unit. *Table 6-4* describes branch instruction latencies.

*Table 6-4. Branch Instructions*

| Instruction | Mnemonic | Primary Opcode | Extended Opcode | Latency |
|---|---|---|---|---|
| Branch | **b**[l][a] | 18 | — | Unless these instructions update either the CTR or the LR, branch operations are folded if they are either taken or predicted as taken. They fall through if they are not taken or predicted as not taken. |
| Branch Conditional | **bc**[l][a] | 16 | — | |
| Branch Conditional to Count Register | **bcctr**[l] | 19 | 528 | |
| Branch Conditional to Link Register | **bclr**[l] | 19 | 16 | |

*Table 6-5* lists system-register instruction latencies.

*Table 6-5. System-Register Instructions* (Page 1 of 2)

| Instruction | Mnemonic | Primary Opcode | Extended Opcode | Unit | Cycles | Serialization |
|---|---|---|---|---|---|---|
| Enforce In-Order Execution of I/O | **eieio** | 31 | 854 | SRU | 1 | — |
| Instruction Synchronize | **isync** | 19 | 150 | SRU | 2 | Completion, refetch |
| Move-from Machine State Register | **mfmsr** | 31 | 83 | SRU | 1 | — |

1. This 3-cycle operation assumes no pending stores in the store queue. If there are pending stores, the **sync** completes after the stores complete to memory. If broadcast is enabled on the 60x bus, **sync** completes only after a successful broadcast.
2. **tlbsync** is dispatched only to the completion buffer (not to any execution unit) and is marked finished as it is dispatched. Upon retirement, it waits for an external TLB Invalidate Synchronize (TLBISYNC) signal to be asserted. In most systems, TLBISYNC is always asserted so the instruction is a no-op.

*Table 6-5. System-Register Instructions* (Page 2 of 2)

| Instruction | Mnemonic | Primary Opcode | Extended Opcode | Unit | Cycles | Serialization |
|---|---|---|---|---|---|---|
| Move-from Special Purpose Register | **mfspr** (data block-address translations [DBATs]) | 31 | 339 | SRU | 3 | Execution |
| | **mfspr** (instruction block-address translations [IBATs]) | 31 | 339 | SRU | 3 | — |
| | **mfspr** (not I/DBATs) | 31 | 339 | SRU | 1 | Execution |
| Move-from Segment Register | **mfsr** | 31 | 595 | SRU | 3 | — |
| Move-from Segment Register Indirect | **mfsrin** | 31 | 659 | SRU | 3 | Execution |
| Move-from Time Base | **mftb** | 31 | 371 | SRU | 1 | — |
| Move-to Machine State Register | **mtmsr** | 31 | 146 | SRU | 1 | Execution |
| Move-to Special Purpose Register | **mtspr** (DBATs) | 31 | 467 | SRU | 2 | Execution |
| | **mtspr** (IBATs) | 31 | 467 | SRU | 2 | Execution |
| | **mtspr** (not I/DBATs) | 31 | 467 | SRU | 2 | Execution |
| Move-to Segment Register | **mtsr** | 31 | 210 | SRU | 2 | Execution |
| Move-to Segment Register Indirect | **mtsrin** | 31 | 242 | SRU | 2 | Execution |
| Move-to Time Base Register | **mttb** | 31 | 467 | SRU | 1 | Execution |
| Return from Interrupt | **rfi** | 19 | 50 | SRU | 2 | Completion, refetch |
| System Call | **sc** | 17 | - -1 | SRU | 2 | Completion, refetch |
| Synchronize | **sync** | 31 | 598 | SRU | 3[1] | — |
| TLB Synchronize | **tlbsync**[2] | 31 | 566 | — | — | |

1. This 3-cycle operation assumes no pending stores in the store queue. If there are pending stores, the **sync** completes after the stores complete to memory. If broadcast is enabled on the 60x bus, **sync** completes only after a successful broadcast.
2. **tlbsync** is dispatched only to the completion buffer (not to any execution unit) and is marked finished as it is dispatched. Upon retirement, it waits for an external TLB Invalidate Synchronize (TLBISYNC) signal to be asserted. In most systems, TLBISYNC is always asserted so the instruction is a no-op.

*Table 6-6* lists condition register logical instruction latencies.

*Table 6-6. Condition Register Logical Instructions*

| Instruction | Mnemonic | Primary Opcode | Extended Opcode | Unit | Cycles | Serialization |
|---|---|---|---|---|---|---|
| Condition Register AND | **crand** | 19 | 257 | SRU | 1 | Execution |
| Condition Register AND with Complement | **crandc** | 19 | 129 | SRU | 1 | Execution |
| Condition Register Equivalent | **creqv** | 19 | 289 | SRU | 1 | Execution |
| Condition Register NAND | **crnand** | 19 | 225 | SRU | 1 | Execution |
| Condition Register NOR | **crnor** | 19 | 33 | SRU | 1 | Execution |
| Condition Register OR | **cror** | 19 | 449 | SRU | 1 | Execution |
| Condition Register OR with Complement | **crorc** | 19 | 417 | SRU | 1 | Execution |
| Condition Register XOR | **crxor** | 19 | 193 | SRU | 1 | Execution |
| Move Condition Register Field | **mcrf** | 19 | 0 | SRU | 1 | Execution |
| Move to Condition Register from XER | **mcrxr** | 31 | 512 | SRU | 1 | Execution |
| Move From Condition Register | **mfcr** | 31 | 19 | SRU | 1 | Execution |
| Move To Condition Register Fields | **mtcrf** | 31 | 144 | SRU | 1 | Execution |

*Table 6-7* shows integer instruction latencies. Note that the IU1 executes all integer arithmetic instructions—multiply, divide, shift, rotate, add, subtract, and compare. The IU2 executes all integer instructions except multiply and divide (shift, rotate, add, subtract, and compare).

*Table 6-7. Integer Instructions* (Page 1 of 3)

| Instruction | Mnemonic | Primary Opcode | Extended Opcode | Unit | Cycles | Serialization |
|---|---|---|---|---|---|---|
| Add Carrying | **addc**[**o**][.] | 31 | 10 | IU1/IU2 | 1 | — |
| Add Extended | **adde**[**o**][.] | 31 | 138 | IU1/IU2 | 1 | Execution |
| Add Immediate | **addi** | 14 | — | IU1/IU2 | 1 | — |
| Add Immediate Carrying | **addic** | 12 | — | IU1/IU2 | 1 | — |
| Add Immediate Carrying and Record | **addic.** | 13 | — | IU1/IU2 | 1 | — |
| Add Immediate Shifted | **addis** | 15 | — | IU1/IU2 | 1 | — |
| Add to Minus One Extended | **addme**[**o**][.] | 31 | 234 | IU1/IU2 | 1 | Execution |
| Add to Zero Extended | **addze**[**o**][.] | 31 | 202 | IU1/IU2 | 1 | Execution |
| Add | **add**[**o**][.] | 31 | 266 | IU1/IU2 | 1 | — |
| AND with Complement | **andc**[.] | 31 | 60 | IU1/IU2 | 1 | — |
| AND Immediate | **andi.** | 28 | — | IU1/IU2 | 1 | — |

*Table 6-7. Integer Instructions* (Page 2 of 3)

| Instruction | Mnemonic | Primary Opcode | Extended Opcode | Unit | Cycles | Serialization |
|---|---|---|---|---|---|---|
| AND Immediate Shifted | **andis.** | 29 | — | IU1/IU2 | 1 | — |
| AND | **and**[.] | 31 | 28 | IU1/IU2 | 1 | — |
| Compare | **cmp** | 31 | 0 | IU1/IU2 | 1 | — |
| Compare Immediate | **cmpi** | 11 | — | IU1/IU2 | 1 | — |
| Compare Logical | **cmpl** | 31 | 32 | IU1/IU2 | 1 | — |
| Compare Logical Immediate | **cmpli** | 10 | — | IU1/IU2 | 1 | — |
| Count Leading Zeros Word | **cntlzw**[.] | 31 | 26 | IU1/IU2 | 1 | — |
| Divide Word Unsigned | **divwu**[**o**][.] | 31 | 459 | IU1 | 19 | — |
| Divide Word | **divw**[**o**][.] | 31 | 491 | IU1 | 19 | — |
| Equivalent | **eqv**[.] | 31 | 284 | IU1/IU2 | 1 | — |
| Extend Sign Byte | **extsb**[.] | 31 | 954 | IU1/IU2 | 1 | — |
| Extend Sign Halfword | **extsh**[.] | 31 | 922 | IU1/IU2 | 1 | — |
| Multiply High Word Unsigned | **mulhwu**[.] | 31 | 11 | IU1 | 2, 3, 4, 5, 6 | — |
| Multiply High Word | **mulhw**[.] | 31 | 75 | IU1 | 2, 3, 4, 5 | — |
| Multiply Low Immediate | **mulli** | 7 | — | IU1 | 2, 3 | — |
| Multiply Low Word | **mullw**[**o**][.] | 31 | 235 | IU1 | 2, 3, 4, 5 | — |
| NAND | **nand**[.] | 31 | 476 | IU1/IU2 | 1 | — |
| Negate | **neg**[**o**][.] | 31 | 104 | IU1/IU2 | 1 | — |
| NOR | **nor**[.] | 31 | 124 | IU1/IU2 | 1 | — |
| OR with Complement | **orc**[.] | 31 | 412 | IU1/IU2 | 1 | — |
| OR Immediate | **ori** | 24 | — | IU1/IU2 | 1 | — |
| OR Immediate Shifted | **oris** | 25 | — | IU1/IU2 | 1 | — |
| OR | **or**[.] | 31 | 444 | IU1/IU2 | 1 | — |
| Rotate Left Word Immediate then Mask Insert | **rlwimi**[.] | 20 | — | IU1/IU2 | 1 | — |
| Rotate Left Word Immediate then AND with Mask | **rlwinm**[.] | 21 | — | IU1/IU2 | 1 | — |
| Rotate Left Word then AND with Mask | **rlwnm**[.] | 23 | — | IU1/IU2 | 1 | — |
| Shift Left Word | **slw**[.] | 31 | 24 | IU1/IU2 | 1 | — |
| Shift Right Algebraic Word Immediate | **srawi**[.] | 31 | 824 | IU1/IU2 | 1 | — |
| Shift Right Algebraic Word | **sraw**[.] | 31 | 792 | IU1/IU2 | 1 | — |
| Shift Right Word | **srw**[.] | 31 | 536 | IU1/IU2 | 1 | — |

*Table 6-7. Integer Instructions* (Page 3 of 3)

| Instruction | Mnemonic | Primary Opcode | Extended Opcode | Unit | Cycles | Serialization |
|---|---|---|---|---|---|---|
| Subtract From Carrying | **subfc**[**o**][**.**] | 31 | 8 | IU1/IU2 | 1 | — |
| Subtract From Extended | **subfe**[**o**][**.**] | 31 | 136 | IU1/IU2 | 1 | Execution |
| Subtract From Immediate Carrying | **subfic** | 8 | — | IU1/IU2 | 1 | — |
| Subtract From Minus One Extended | **subfme**[**o**][**.**] | 31 | 232 | IU1/IU2 | 1 | Execution |
| Subtract From Zero Extended | **subfze**[**o**][**.**] | 31 | 200 | IU1/IU2 | 1 | Execution |
| Subtract From | **subf**[**.**] | 31 | 40 | IU1/IU2 | 1 | — |
| Trap Word | **tw** | 31 | 4 | IU1/IU2 | 2 | — |
| Trap Word Immediate | **twi** | 3 | — | IU1/IU2 | 2 | — |
| XOR Immediate | **xori** | 26 | — | IU1/IU2 | 1 | — |
| XOR Immediate Shifted | **xoris** | 27 | — | IU1/IU2 | 1 | — |
| XOR | **xor**[**.**] | 31 | 316 | IU1/IU2 | 1 | — |

*Table 6-8* shows latencies for floating-point instructions. Pipelined floating-point instructions are shown with the number of clocks in each pipeline stage separated by dashes. Floating-point instructions with a single entry in the cycles column are not pipelined. When the FPU executes these nonpipelined instructions, it remains busy for the full duration of the instruction execution and is not available for subsequent instructions.

*Table 6-8. Floating-Point Instructions* (Page 1 of 2)

| Instruction | Mnemonic | Primary Opcode | Extended Opcode | Unit | Cycles | Serialization |
|---|---|---|---|---|---|---|
| Floating Absolute Value | **fabs**[**.**] | 63 | 264 | FPU | 1-1-1 | — |
| Floating Add Single | **fadds**[**.**] | 59 | 21 | FPU | 1-1-1 | — |
| Floating Add | **fadd**[**.**] | 63 | 21 | FPU | 1-1-1 | — |
| Floating Compare Ordered | **fcmpo** | 63 | 32 | FPU | 1-1-1 | — |
| Floating Compare Unordered | **fcmpu** | 63 | 0 | FPU | 1-1-1 | — |
| Floating Convert To Integer Word with Round toward Zero | **fctiwz**[**.**] | 63 | 15 | FPU | 1-1-1 | — |
| Floating Convert To Integer Word | **fctiw**[**.**] | 63 | 14 | FPU | 1-1-1 | — |
| Floating Divide Single | **fdivs**[**.**] | 59 | 18 | FPU | 17 | — |
| Floating Divide | **fdiv**[**.**] | 63 | 18 | FPU | 31 | — |
| Floating Multiply-Add Single | **fmadds**[**.**] | 59 | 29 | FPU | 1-1-1 | — |
| Floating Multiply-Add | **fmadd**[**.**] | 63 | 29 | FPU | 2-1-1 | — |
| Floating Move Register | **fmr**[**.**] | 63 | 72 | FPU | 1-1-1 | — |

*Table 6-8. Floating-Point Instructions* (Page 2 of 2)

| Instruction | Mnemonic | Primary Opcode | Extended Opcode | Unit | Cycles | Serialization |
|---|---|---|---|---|---|---|
| Floating Multiply-Subtract Single | **fmsubs**[.] | 59 | 28 | FPU | 1-1-1 | — |
| Floating Multiply-Subtract | **fmsub**[.] | 63 | 28 | FPU | 2-1-1 | — |
| Floating Multiply Single | **fmuls**[.] | 59 | 25 | FPU | 1-1-1 | — |
| Floating Multiply | **fmul**[.] | 63 | 25 | FPU | 2-1-1 | — |
| Floating Negative Absolute Value | **fnabs**[.] | 63 | 136 | FPU | 1-1-1 | — |
| Floating Negate | **fneg**[.] | 63 | 40 | FPU | 1-1-1 | — |
| Floating Negative Multiply-Add Single | **fnmadds**[.] | 59 | 31 | FPU | 1-1-1 | — |
| Floating Negative Multiply-Add | **fnmadd**[.] | 63 | 31 | FPU | 2-1-1 | — |
| Floating Negative Multiply-Subtract Single | **fnmsubs**[.] | 59 | 30 | FPU | 1-1-1 | — |
| Floating Negative Multiply-Subtract | **fnmsub**[.] | 63 | 30 | FPU | 2-1-1 | — |
| Floating Reciprocal Estimate Single | **fres**[.] | 59 | 24 | FPU | 2-1-1 | — |
| Floating Round to Single-Precision | **frsp**[.] | 63 | 12 | FPU | 1-1-1 | — |
| Floating Reciprocal Square Root Estimate | **frsqrte**[.] | 63 | 26 | FPU | 2-1-1 | — |
| Floating Select | **fsel**[.] | 63 | 23 | FPU | 1-1-1 | — |
| Floating Subtract Single | **fsubs**[.] | 59 | 20 | FPU | 1-1-1 | — |
| Floating Subtract | **fsub**[.] | 63 | 20 | FPU | 1-1-1 | — |
| Move to Condition Register from FPSCR | **mcrfs** | 63 | 64 | FPU | 1-1-1 | Execution |
| Move From FPSCR | **mffs**[.] | 63 | 583 | FPU | 1-1-1 | Execution |
| Move To FPSCR Bit 0 | **mtfsb0**[.] | 63 | 70 | FPU | 3 | — |
| Move To FPSCR Bit 1 | **mtfsb1**[.] | 63 | 38 | FPU | 3 | — |
| Move To FPSCR Field Immediate | **mtfsfi**[.] | 63 | 134 | FPU | 3 | — |
| Move To FPSCR Fields | **mtfsf**[.] | 63 | 711 | FPU | 3 | — |

*Table 6-9* shows load-and-store instruction latencies. Pipelined load/store instructions are shown with cycles of total latency and throughput cycles separated by a colon.

*Table 6-9. Load-and-Store Instructions* (Page 1 of 4)

| Instruction | Mnemonic | Primary Opcode | Extended Opcode | Unit | Cycles | Serialization |
|---|---|---|---|---|---|---|
| Data Cache Block Flush | **dcbf** | 31 | 86 | LSU | 3:5[1] | Execution |
| Data Cache Block Invalidate | **dcbi** | 31 | 470 | LSU | 3:3[1] | Execution |
| Data Cache Block Store | **dcbst** | 31 | 54 | LSU | 3:5[1] | Execution |
| Data Cache Block Touch | **dcbt** | 31 | 278 | LSU | 2:1 | — |
| Data Cache Block Touch for Store | **dcbtst** | 31 | 246 | LSU | 2:1 | — |
| Data Cache Block set to Zero | **dcbz** | 31 | 1014 | LSU | 3:6[12] | Execution |
| External Control In Word Indexed | **eciwx** | 31 | 310 | LSU | 2:1 | — |
| External Control Out Word Indexed | **ecowx** | 31 | 438 | LSU | 2:1 | — |
| Instruction Cache Block Invalidate | **icbi** | 31 | 982 | LSU | 3:4[1] | Execution |
| Load Byte and Zero | **lbz** | 34 | — | LSU | 2:1 | — |
| Load Byte and Zero with Update | **lbzu** | 35 | — | LSU | 2:1 | — |
| Load Byte and Zero with Update Indexed | **lbzux** | 31 | 119 | LSU | 2:1 | — |
| Load Byte and Zero Indexed | **lbzx** | 31 | 87 | LSU | 2:1 | — |
| Load Floating-Point Double | **lfd** | 50 | — | LSU | 2:1 | — |
| Load Floating-Point Double with Update | **lfdu** | 51 | — | LSU | 2:1 | — |
| Load Floating-Point Double with Update Indexed | **lfdux** | 31 | 631 | LSU | 2:1 | — |
| Load Floating-Point Double Indexed | **lfdx** | 31 | 599 | LSU | 2:1 | — |
| Load Floating-Point Single | **lfs** | 48 | — | LSU | 2:1 | — |
| Load Floating-Point Single with Update | **lfsu** | 49 | — | LSU | 2:1 | — |
| Load Floating-Point Single with Update Indexed | **lfsux** | 31 | 567 | LSU | 2:1 | — |

1. For cache operations, the first number indicates the latency in finishing a single instruction; the second indicates the throughput for back-to-back cache operations. Throughput might be larger than the initial latency, as more cycles might be needed to complete the instruction to the cache, which stays busy keeping subsequent cache operations from executing.
2. The throughput number of six cycles for **dcbz** assumes it is to nonglobal (M = 0) address space. For global address space, throughput is at least 11 cycles.
3. Load/store multiple/string instruction cycles are represented as a fixed number of cycles plus a variable number of cycles, where *n* is the number of words accessed by the instruction.

*Table 6-9. Load-and-Store Instructions* (Page 2 of 4)

| Instruction | Mnemonic | Primary Opcode | Extended Opcode | Unit | Cycles | Serialization |
|---|---|---|---|---|---|---|
| Load Floating-Point Single Indexed | **lfsx** | 31 | 535 | LSU | 2:1 | — |
| Load Halfword Algebraic | **lha** | 42 | — | LSU | 2:1 | — |
| Load Halfword Algebraic with Update | **lhau** | 43 | — | LSU | 2:1 | — |
| Load Halfword Algebraic with Update Indexed | **lhaux** | 31 | 375 | LSU | 2:1 | — |
| Load Halfword Algebraic Indexed | **lhax** | 31 | 343 | LSU | 2:1 | — |
| Load Halfword Byte-Reverse Indexed | **lhbrx** | 31 | 790 | LSU | 2:1 | — |
| Load Halfword and Zero | **lhz** | 40 | — | LSU | 2:1 | — |
| Load Halfword and Zero with Update | **lhzu** | 41 | — | LSU | 2:1 | — |
| Load Halfword and Zero with Update Indexed | **lhzux** | 31 | 311 | LSU | 2:1 | — |
| Load Halfword and Zero Indexed | **lhzx** | 31 | 279 | LSU | 2:1 | — |
| Load Multiple Word | **lmw** | 46 | — | LSU | 2 + *n*[3] | Completion, execution |
| Load String Word Immediate | **lswi** | 31 | 597 | LSU | 2 + *n* [3] | Completion, execution |
| Load String Word Indexed | **lswx** | 31 | 533 | LSU | 2 + *n* [3] | Completion, execution |
| Load Word And Reserve Indexed | **lwarx** | 31 | 20 | LSU | 3:1 | Execution |
| Load Word Byte-Reverse Indexed | **lwbrx** | 31 | 534 | LSU | 2:1 | — |
| Load Word and Zero | **lwz** | 32 | — | LSU | 2:1 | — |
| Load Word and Zero with Update | **lwzu** | 33 | — | LSU | 2:1 | — |
| Load Word and Zero with Update Indexed | **lwzux** | 31 | 55 | LSU | 2:1 | — |
| Load Word and Zero Indexed | **lwzx** | 31 | 23 | LSU | 2:1 | — |
| Store Byte | **stb** | 38 | — | LSU | 2:1 | — |
| Store Byte with Update | **stbu** | 39 | — | LSU | 2:1 | — |
| Store Byte with Update Indexed | **stbux** | 31 | 247 | LSU | 2:1 | — |
| Store Byte Indexed | **stbx** | 31 | 215 | LSU | 2:1 | — |

1. For cache operations, the first number indicates the latency in finishing a single instruction; the second indicates the throughput for back-to-back cache operations. Throughput might be larger than the initial latency, as more cycles might be needed to complete the instruction to the cache, which stays busy keeping subsequent cache operations from executing.
2. The throughput number of six cycles for **dcbz** assumes it is to nonglobal (M = 0) address space. For global address space, throughput is at least 11 cycles.
3. Load/store multiple/string instruction cycles are represented as a fixed number of cycles plus a variable number of cycles, where *n* is the number of words accessed by the instruction.

*Table 6-9. Load-and-Store Instructions* (Page 3 of 4)

| Instruction | Mnemonic | Primary Opcode | Extended Opcode | Unit | Cycles | Serialization |
|---|---|---|---|---|---|---|
| Store Floating-Point Double | **stfd** | 54 | — | LSU | 2:1 | — |
| Store Floating-Point Double with Update | **stfdu** | 55 | — | LSU | 2:1 | — |
| Store Floating-Point Double with Update Indexed | **stfdux** | 31 | 759 | LSU | 2:1 | — |
| Store Floating-Point Double Indexed | **stfdx** | 31 | 727 | LSU | 2:1 | — |
| Store Floating-Point as Integer Word Indexed | **stfiwx** | 31 | 983 | LSU | 2:1 | — |
| Store Floating-Point Single | **stfs** | 52 | — | LSU | 2:1 | — |
| Store Floating-Point Single with Update | **stfsu** | 53 | — | LSU | 2:1 | — |
| Store Floating-Point Single with Update Indexed | **stfsux** | 31 | 695 | LSU | 2:1 | — |
| Store Floating-Point Single Indexed | **stfsx** | 31 | 663 | LSU | 2:1 | — |
| Store Halfword | **sth** | 44 | — | LSU | 2:1 | — |
| Store Halfword Byte-Reverse Indexed | **sthbrx** | 31 | 918 | LSU | 2:1 | — |
| Store Halfword with Update | **sthu** | 45 | — | LSU | 2:1 | — |
| Store Halfword with Update Indexed | **sthux** | 31 | 439 | LSU | 2:1 | — |
| Store Halfword Indexed | **sthx** | 31 | 407 | LSU | 2:1 | — |
| Store Multiple Word | **stmw** | 47 | — | LSU | 2 + *n* [3] | Execution |
| Store String Word Immediate | **stswi** | 31 | 725 | LSU | 2 + *n* [3] | Execution |
| Store String Word Indexed | **stswx** | 31 | 661 | LSU | 2 + *n* [3] | Execution |
| Store Word | **stw** | 36 | — | LSU | 2:1 | — |
| Store Word Byte-Reverse Indexed | **stwbrx** | 31 | 662 | LSU | 2:1 | — |
| Store Word Conditional Indexed | **stwcx.** | 31 | 150 | LSU | 8:8 | Execution |
| Store Word with Update | **stwu** | 37 | — | LSU | 2:1 | — |

1. For cache operations, the first number indicates the latency in finishing a single instruction; the second indicates the throughput for back-to-back cache operations. Throughput might be larger than the initial latency, as more cycles might be needed to complete the instruction to the cache, which stays busy keeping subsequent cache operations from executing.
2. The throughput number of six cycles for **dcbz** assumes it is to nonglobal (M = 0) address space. For global address space, throughput is at least 11 cycles.
3. Load/store multiple/string instruction cycles are represented as a fixed number of cycles plus a variable number of cycles, where *n* is the number of words accessed by the instruction.

*Table 6-9. Load-and-Store Instructions* (Page 4 of 4)

| Instruction | Mnemonic | Primary Opcode | Extended Opcode | Unit | Cycles | Serialization |
|---|---|---|---|---|---|---|
| Store Word with Update Indexed | **stwux** | 31 | 183 | LSU | 2:1 | — |
| Store Word Indexed | **stwx** | 31 | 151 | LSU | 2:1 | — |
| TLB Invalidate Entry | **tlbie** | 31 | 306 | LSU | 3:4[1] | Execution |

1. For cache operations, the first number indicates the latency in finishing a single instruction; the second indicates the throughput for back-to-back cache operations. Throughput might be larger than the initial latency, as more cycles might be needed to complete the instruction to the cache, which stays busy keeping subsequent cache operations from executing.
2. The throughput number of six cycles for **dcbz** assumes it is to nonglobal (M = 0) address space. For global address space, throughput is at least 11 cycles.
3. Load/store multiple/string instruction cycles are represented as a fixed number of cycles plus a variable number of cycles, where *n* is the number of words accessed by the instruction.

# 7. Signal Descriptions

This chapter describes the 750GX microprocessor's external signals. It contains a concise description of individual signals, showing behavior when the signal is asserted and negated and when the signal is an input and an output.

**Note:** A bar over a signal name indicates that the signal is active low—for example, $\overline{\text{ARTRY}}$ (address retry) and $\overline{\text{TS}}$ (transfer start). Active-low signals are referred to as asserted (active) when they are low and negated when they are high. Signals that are not active low, such as A[0–31] (address-bus signals) and TT[0–4] (transfer type signals) are referred to as asserted when they are high and negated when they are low.

The 750GX's signals are grouped as follows:

Address arbitration — The 750GX uses these signals to arbitrate for address-bus mastership.

Address transfer start — Indicate that a bus master has begun a transaction on the address bus.

Address transfer — These signals include the address bus. They are used to transfer the address.

Transfer attribute — Provide information about the type of transfer, such as the transfer size and whether the transaction is burst, write-through, or cache-inhibited.

Address transfer termination — Acknowledge the end of the address phase of the transaction. They also indicate whether a condition exists that requires the address phase to be repeated.

Data arbitration — The 750GX uses these signals to arbitrate for data-bus mastership.

Data transfer — These signals, which consist of the data bus, are used to transfer the data.

Data-transfer termination — Data termination signals are required after each data beat in a data transfer. In a single-beat transaction, the data termination signals also indicate the end of the tenure; while in burst accesses, the data termination signals apply to individual beats and indicate the end of the tenure only after the final data beat. They also indicate whether a condition exists that requires the data phase to be repeated.

Interrupts/resets — These signals include the external interrupt signal, checkstop signals, and both soft reset and hard reset signals. They are used to interrupt and, under various conditions, to reset the processor.

Processor status and control — These signals are used to set the reservation coherency bit, enable the time base, and for other functions. They are also used in conjunction with such resources as secondary caches and the time-base facility.

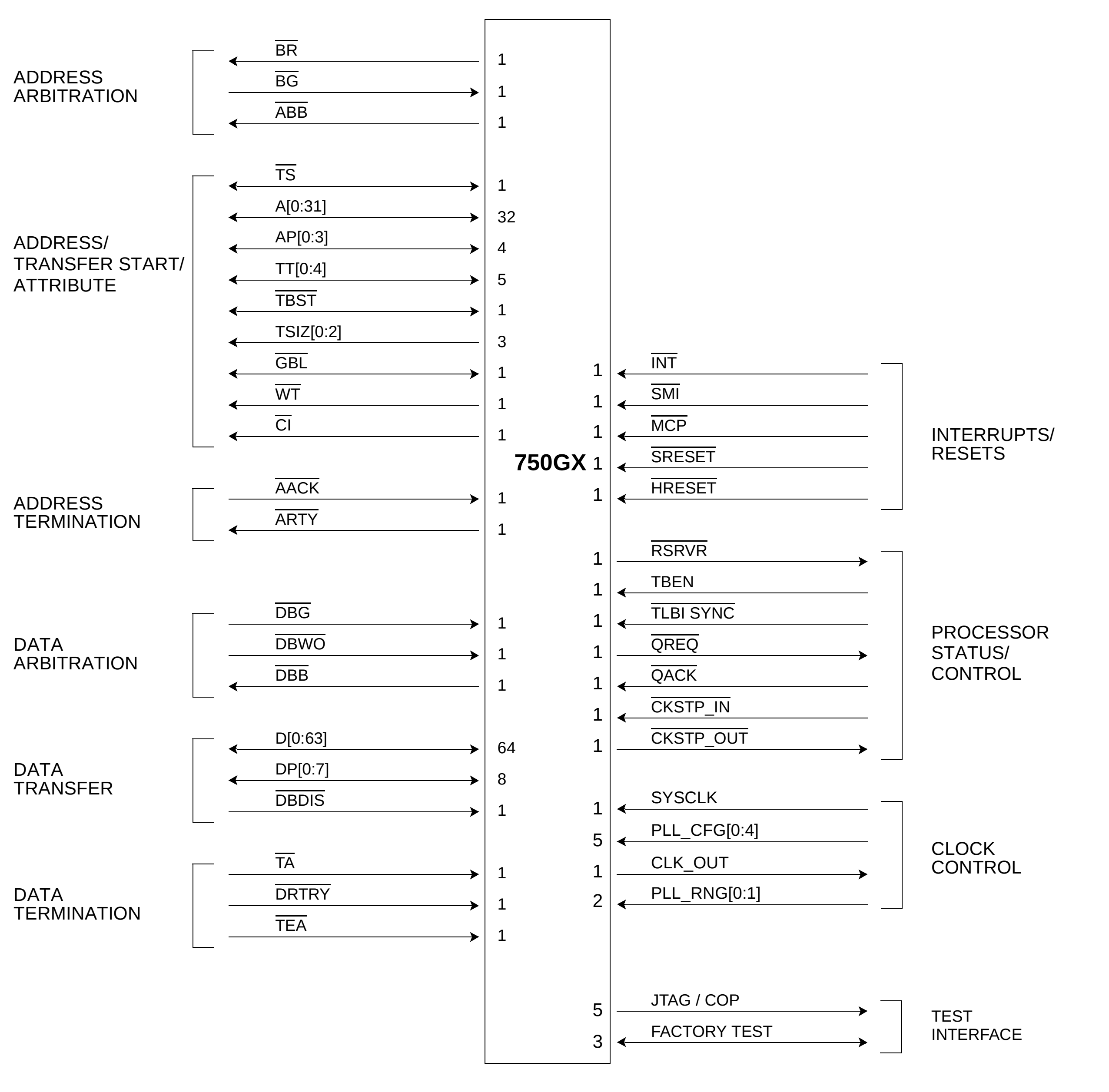
Clock control — Determine the system clock frequency. These signals can also be used to synchronize multiprocessor systems.

Test interface — The Joint Test Action Group (JTAG) (IEEE 1149.1a-1993) interface and the common on-chip processor (COP) unit provide a serial interface to the system for performing board-level boundary-scan interconnect tests.

## 7.1 Signal Configuration

*Figure 7-1* illustrates the 750GX’s signal configuration, showing how the signals are grouped. A pinout showing pin numbers is included in the *PowerPC 750GX RISC Microprocessor Datasheet.*

*Figure 7-1. 750GX Signal Groups*



## 7.2 Signal Descriptions

This section summarizes the functions of individual signals on the 750GX, grouped according to *Figure 7-1*. *Chapter 8, Bus Interface Operation,* on page 279 describes many of these signals in greater detail, both with respect to how individual signals function and to how the groups of signals interact. The information in the remainder of this chapter applies to the basic transfer protocol of the 60x bus. This is the normal transfer protocol used by 60x devices. The extended transfer protocol (also referred to as ETP or PLL 0 internal configuration [PI0] protocol) is not supported by the 750GX.

**Note:** In the following tables, “cycle” or “clock” refers to a single bus clock period, which corresponds to one or more internal processor clocks depending on the clock mode programmed for the 750GX.

**Note:** In phase-locked loop (PLL)-bypass mode, the SYSCLK input signal clocks the internal processor directly, the PLL is disabled, and the bus mode is set for 1:1 mode operation. This mode is intended for factory use only.

### 7.2.1 Address-Bus Arbitration Signals

The address arbitration signals are the input and output signals the 750GX uses to request the address bus, recognize when the request is granted, and indicate to other devices when mastership is granted.

For a detailed description of how these signals interact, see *Section 8.3.1, Address-Bus Arbitration,* on page 290.

#### *7.2.1.1 Bus Request ($\overline{\text{BR}}$)—Output*

| | | |
|---|---|---|
| **State** | *Asserted* | Indicates that the 750GX has a bus transaction to perform, and that it is waiting for a qualified bus grant (BG) to begin the address tenure. BR might be asserted even if all four (five with snoop) possible address tenures have already been granted. |
| | *Negated* | Indicates that the 750GX does not have a bus transaction to perform, or, if parked, that it is potentially ready to start a bus transaction on the next clock (with proper qualification, see BG). |
| **Timing** | *Assertion* | Occurs on any cycle. Will not occur if the 750GX is parked and the address bus is idle (BG asserted and address bus busy [ABB] input negated). |
| | *Negation* | Occurs during the cycle TS is asserted even if another transaction is pending. Also occurs the cycle after any *qualified* ARTRY on the bus unless this chip asserted the ARTRY and requires it to perform a snoop copyback. Will also occur if the bus request is internally cancelled before receiving a qualified BG. |
| | *High Impedance* | Occurs during a hard reset or checkstop condition. |

### *7.2.1.2 Bus Grant ($\overline{\text{BG}}$)—Input*

**State** *Asserted* Indicates that the 750GX may, with proper qualification, assume mastership of the address bus. A qualified bus grant occurs when $\overline{\text{BG}}$ is asserted and $\overline{\text{ABB}}$ and $\overline{\text{ARTRY}}$ are not asserted on the bus cycled following the assertion of $\overline{\text{AACK}}$.

Note that the assertion of $\overline{\text{BR}}$ is not required for a qualified bus grant (to allow bus parking).

*Negated* Indicates that the 750GX is not granted next address-bus ownership.

**Timing** *Assertion* May occur on any cycle. Once the 750GX has assumed address-bus ownership, it will not begin checking for $\overline{\text{BG}}$ again until the cycle after $\overline{\text{AACK}}$.

*Negation* Must occur whenever the 750GX must be prevented from starting a bus transaction. The 750GX will still assume address-bus ownership on the cycle $\overline{\text{BG}}$ is negated if $\overline{\text{BG}}$ was asserted in the previous cycle with other bus grant qualifications.

### *7.2.1.3 Address Bus Busy ($\overline{\text{ABB}}$)*

The address bus busy ($\overline{\text{ABB}}$) signal is both an input and an output signal.

*Address Bus Busy ($\overline{\text{ABB}}$)—Output*

**State** *Asserted* Indicates that the 750GX is the current address-bus owner. The 750GX will not assume address-bus ownership if the bus request is internally cancelled by the cycle a qualified $\overline{\text{BG}}$ would have been recognized.

*Negated* Indicates that the 750GX is not the current address-bus owner.

**Timing** *Assertion* Occurs the cycle after a qualified $\overline{\text{BG}}$ is accepted by the 750GX, and remains asserted for the duration of the address tenure.

*Negation* Negates for a fraction of a bus cycle (one-half minimum, depends on clock mode) starting the cycle following the assertion of $\overline{\text{AACK}}$. Then releases to the high impedance state.

*Address Bus Busy (ABB)—Input*

| | | |
|---|---|---|
| **State** | *Asserted* | Indicates that another master is the current address-bus owner. |
| | *Negated* | Indicates that the address bus might be available for use by the 750GX (see BG).<br>The 750GX will also track the state of ABB on the bus from the TS and AACK inputs. (See *Section 8.3.1, Address-Bus Arbitration,* on page 290.) |
| **Timing** | *Assertion* | Must occur whenever the 750GX must be prevented from using the address bus. |
| | *Negation* | May occur whenever the 750GX can use the address bus. |

### 7.2.2 Address Transfer Start Signals

Address transfer start signals are input and output signals that indicate that an address-bus transfer has begun. The transfer start (TS) signal identifies the operation as a memory transaction.

For detailed information about how TS interacts with other signals, see *Section 8.3.2, Address Transfer,* on page 292.

#### *7.2.2.1 Transfer Start (TS)*

The TS signal is both an input and an output signal on the 750GX.

*Transfer Start (TS)—Output*

| | | |
|---|---|---|
| **State** | *Asserted* | Indicates that the 750GX has begun a memory bus transaction, and that the address-bus and transfer attribute signals are valid. When asserted with the appropriate TT[0–4] signals, it is also an implied data-bus request for a memory transaction (unless it is an address-only operation). |
| | *Negated* | Indicates that a bus transaction is not being started. |
| **Timing** | *Assertion* | Occurs on the first cycle of ABB assertion. |
| | *Negation* | Occurs one bus clock cycle after TS is asserted. |
| | *High Impedance* | Occurs the bus cycle following AACK (same cycle as ABB negation). |

*Transfer Start (TS)—Input*

| | | |
|---|---|---|
| **State** | *Asserted* | Indicates that another master has begun a bus transaction, and that the address-bus and transfer attribute signals are valid for snooping (see *Section 7.2.4.6, Global (GBL),* on page 261. |
| | *Negated* | Indicates that a bus cycle is not being started. |
| **Timing** | *Assertion/ Negation* | Must be asserted for one cycle only, and then immediately negated. Assertion may occur at any time during the assertion of ABB. |

### 7.2.3 Address Transfer Signals

The address transfer signals are used to transmit the address and to generate and monitor parity for the address transfer. For a detailed description of how these signals interact, see *Section 8.3.2, Address Transfer,* on page 292.

#### *7.2.3.1 Address Bus (A[0–31])*

The address bus (A[0–31]) consists of 32 signals that are both input and output signals.

*Address Bus (A[0–31])—Output*

**State** *Asserted/Negated* Represents the physical address (real address in the architecture specification) of the data to be transferred. On burst transfers, the address bus presents the double-word-aligned address containing the critical code/data that missed the cache on a read operation, or the first double word of the cache line on a write operation. Note that the address output during burst operations is not incremented. See *Section 8.3.2, Address Transfer,* on page 292.

**Timing** *Assertion/Negation* Occurs on the bus clock cycle after a qualified bus grant (coincides with assertion of $\overline{\text{TS}}$). Remains driven/valid for the duration of the address tenure.

*High Impedance* Occurs one bus clock cycle following the assertion of $\overline{\text{AACK}}$; no precharge action is performed on release.

*Address Bus (A[0–31])—Input*

**State** *Asserted/Negated* Represents the physical address of a snoop operation.

**Timing** *Assertion/Negation* Must occur on the same bus clock cycle as the assertion of $\overline{\text{TS}}$; is sampled by the 750GX only on this cycle.

#### *7.2.3.2 Address-Bus Parity (AP[0–3])*

The address-bus parity (AP[0–3]) signals are both input and output signals reflecting 1 bit of odd-byte parity for each of the 4 bytes of address when a valid address is on the bus.

*Address-Bus Parity (AP[0–3])—Output*

| | | |
|---|---|---|
| **State** | *Asserted/ Negated* | Represents odd parity for each of the 4 bytes of the physical address for a transaction. Odd parity means that an odd number of bits, including the parity bit, are driven high. Address parity is generated by the 750GX as address-bus master (unless disabled through Hardware-Implementation-Dependent Register 0 [HID0]). The signal assignments correspond to the following:<br>AP0 A[0–7]<br>AP1 A[8–15]<br>AP2 A[16–23]<br>AP3 A[24–31] |
| **Timing** | *Assertion/ Negation/ High Impedance* | The same as A[0–31]. |

*Address-Bus Parity (AP[0–3])—Input*

| | | |
|---|---|---|
| **State** | *Asserted/ Negated* | Represents odd parity for each of the 4 bytes of the physical address for snooping operations. Detected even parity causes the processor to take a machine check exception or enter the checkstop state if address-parity checking is enabled in the HID0 register. See *Section 2.1.2.2, Hardware-Implementation-Dependent Register 0 (HID0),* on page 65. |
| **Timing** | *Assertion/ Negation* | The same as A[0–31]. |

### 7.2.4 Address Transfer Attribute Signals

The transfer attribute signals are a set of signals that further characterize the transfer—such as the size of the transfer, whether it is a read or write operation, and whether it is a burst or single-beat transfer. For a detailed description of how these signals interact, see *Section 8.3.2, Address Transfer,* on page 292.

**Note:** Some signal functions vary depending on whether the transaction is a memory access or an I/O access.

#### *7.2.4.1 Transfer Type (TT[0–4])*

The transfer type (TT[0–4]) signals consist of five input/output signals on the 750GX. For a complete description of TT[0–4] signals and for transfer type encodings, see *Table 7-1*.

*Transfer Type (TT[0–4])—Output*

| | | |
|---|---|---|
| **State** | *Asserted/ Negated* | Indicates the type of transfer in progress. |
| **Timing** | *Assertion/ Negation/ High Impedance* | The same as A[0–31]. |

*Transfer Type (TT[0–4])—Input*

| | | |
|---|---|---|
| **State** | *Asserted/ Negated* | Indicates the type of transfer in progress (see *Table 7-1*). |
| **Timing** | *Assertion/ Negation* | The same as A[0–31]. |

*Table 7-1* describes the transfer encodings for the 750GX bus master.

*Table 7-1. Transfer Type Encodings for PowerPC 750GX Bus Master* (Page 1 of 2)

| 750GX Bus Master Transaction | Transaction Source | TT0 | TT1 | TT2 | TT3 | TT4 | 60x Bus Specification Command | Transaction |
|---|---|---|---|---|---|---|---|---|
| Address only[1] | Data Cache Block Store (**dcbst**) | 0 | 0 | 0 | 0 | 0 | Clean block | Address only |
| Address only[1] | Data Cache Block Flush (**dcbf**) | 0 | 0 | 1 | 0 | 0 | Flush block | Address only |
| Address only[1] | Synchronize (**sync**) | 0 | 1 | 0 | 0 | 0 | **sync** | Address only |
| Address only | Data Cache Block Set to Zero (**dcbz**) | 0 | 1 | 1 | 0 | 0 | Kill block | Address only |
| Address only[1] | Data Cache Block Invalidate (**dcbi**) | 0 | 1 | 1 | 0 | 0 | Kill block | Address only |
| Address only[1] | Enforce In-Order Execution of I/O (**eieio**) | 1 | 0 | 0 | 0 | 0 | **eieio** | Address only |
| Single-beat write (nonGBL) | External Control Out Word Indexed (**ecowx**) | 1 | 0 | 1 | 0 | 0 | External control word write | Single-beat write |
| N/A | N/A | 1 | 1 | 0 | 0 | 0 | Translation Lookaside Buffer (TLB) invalidate | Address only |
| Single-beat read (nonGBL) | External Control In Word Indexed (**eciwx**) | 1 | 1 | 1 | 0 | 0 | External control word read | Single-beat read |

[1] Address-only transaction occurs if enabled by setting the HID0[ABE] bit to 1.

*Table 7-1. Transfer Type Encodings for PowerPC 750GX Bus Master* (Page 2 of 2)

| 750GX Bus Master Transaction | Transaction Source | TT0 | TT1 | TT2 | TT3 | TT4 | 60x Bus Specification Command | Transaction |
|---|---|---|---|---|---|---|---|---|
| N/A | N/A | 0 | 0 | 0 | 0 | 1 | Load Word And Reserve Indexed (**lwarx**) reservation set | Address only |
| N/A | N/A | 0 | 0 | 1 | 0 | 1 | Reserved | — |
| N/A | N/A | 0 | 1 | 0 | 0 | 1 | TLB Synchronize (**tlbsync**) | Address only |
| N/A | N/A | 0 | 1 | 1 | 0 | 1 | Instruction Cache Block Invalidate (**icbi**) | Address only |
| N/A | N/A | 1 | X | X | 0 | 1 | Reserved | — |
| Single-beat write | Caching-inhibited or write-through store | 0 | 0 | 0 | 1 | 0 | Write-with-flush | Single-beat write or burst |
| Burst (non$\overline{\text{GBL}}$) | Castout, or snoop copy-back | 0 | 0 | 1 | 1 | 0 | Write-with-kill | Burst |
| Single-beat read | Caching-inhibited load or instruction fetch | 0 | 1 | 0 | 1 | 0 | Read | Single-beat read or burst |
| Burst | Load miss, store miss, or instruction fetch | 0 | 1 | 1 | 1 | 0 | Read-with-intent-to-modify | Burst |
| Single-beat write | Store Word Conditional Indexed (**stwcx.**) | 1 | 0 | 0 | 1 | 0 | Write-with-flush-atomic | Single-beat write |
| N/A | N/A | 1 | 0 | 1 | 1 | 0 | Reserved | N/A |
| Single-beat read | **lwarx** (caching-inhibited load) | 1 | 1 | 0 | 1 | 0 | Read-atomic | Single-beat read or burst |
| Burst | **lwarx**<br>(load miss) | 1 | 1 | 1 | 1 | 0 | Read-with-intent-to-modify-atomic | Burst |
| N/A | N/A | 0 | 0 | 0 | 1 | 1 | Reserved | — |
| N/A | N/A | 0 | 0 | 1 | 1 | 1 | Reserved | — |
| N/A | N/A | 0 | 1 | 0 | 1 | 1 | Read-with-no-intent-to-cache | Single-beat read or burst |
| N/A | N/A | 0 | 1 | 1 | 1 | 1 | Reserved | — |
| N/A | N/A | 1 | X | X | 1 | 1 | Reserved | — |

1. Address-only transaction occurs if enabled by setting the HID0[ABE] bit to 1.

*Table 7-2* describes the 60x bus specification transfer encodings and the 750GX bus snoop response on an address hit.

*Table 7-2. PowerPC 750GX Snoop Hit Response* (Page 1 of 2)

| 60x Bus Specification Command | Transaction | TT0 | TT1 | TT2 | TT3 | TT4 | PowerPC 750GX Bus Snooper; Action on Hit |
|---|---|---|---|---|---|---|---|
| Clean block | Address only | 0 | 0 | 0 | 0 | 0 | N/A |
| Flush block | Address only | 0 | 0 | 1 | 0 | 0 | N/A |
| **sync** | Address only | 0 | 1 | 0 | 0 | 0 | N/A |
| Kill block | Address only | 0 | 1 | 1 | 0 | 0 | Flush, cancel reservation |

*Table 7-2. PowerPC 750GX Snoop Hit Response* (Page 2 of 2)

| 60x Bus Specification Command | Transaction | TT0 | TT1 | TT2 | TT3 | TT4 | PowerPC 750GX Bus Snooper; Action on Hit |
|---|---|---|---|---|---|---|---|
| **eieio** | Address only | 1 | 0 | 0 | 0 | 0 | N/A |
| External control word write | Single-beat write | 1 | 0 | 1 | 0 | 0 | N/A |
| TLB Invalidate | Address only | 1 | 1 | 0 | 0 | 0 | N/A |
| External control word read | Single-beat read | 1 | 1 | 1 | 0 | 0 | N/A |
| **lwarx**<br>reservation set | Address only | 0 | 0 | 0 | 0 | 1 | N/A |
| Reserved | — | 0 | 0 | 1 | 0 | 1 | N/A |
| **tlbsync** | Address only | 0 | 1 | 0 | 0 | 1 | N/A |
| **icbi** | Address only | 0 | 1 | 1 | 0 | 1 | N/A |
| Reserved | — | 1 | X | X | 0 | 1 | N/A |
| Write-with-flush | Single-beat write or burst | 0 | 0 | 0 | 1 | 0 | Flush, cancel reservation |
| Write-with-kill | Single-beat write or burst | 0 | 0 | 1 | 1 | 0 | Kill, cancel reservation |
| Read | Single-beat read or burst | 0 | 1 | 0 | 1 | 0 | Clean or flush |
| Read-with-intent-to-modify | Burst | 0 | 1 | 1 | 1 | 0 | Flush |
| Write-with-flush-atomic | Single-beat write | 1 | 0 | 0 | 1 | 0 | Flush, cancel reservation |
| Reserved | N/A | 1 | 0 | 1 | 1 | 0 | N/A |
| Read-atomic | Single-beat read or burst | 1 | 1 | 0 | 1 | 0 | Clean or flush |
| Read-with-intent-to modify-atomic | Burst | 1 | 1 | 1 | 1 | 0 | Flush |
| Reserved | — | 0 | 0 | 0 | 1 | 1 | N/A |
| Reserved | — | 0 | 0 | 1 | 1 | 1 | N/A |
| Read-with-no-intent-to-cache | Single-beat read or burst | 0 | 1 | 0 | 1 | 1 | Clean |
| Reserved | — | 0 | 1 | 1 | 1 | 1 | N/A |
| Reserved | — | 1 | X | X | 1 | 1 | N/A |

### *7.2.4.2 Transfer Size (TSIZ[0–2])—Output*

**State** *Asserted/ Negated*

For memory accesses, these signals, along with the transfer burst (TBST) signal, indicate the data-transfer size for the current bus operation, as shown in *Table 7-3* on page 259. *Table 8-4* on page 296 shows how the transfer size signals are used with the address signals for aligned transfers. *Table 8-5* on page 298 shows how the transfer size signals are used with the address signals for misaligned transfers.

For external control instructions (**eciwx** and **ecowx**), TSIZ[0–2] are used to output bits 29–31 of the External Access Register (EAR), which are used to form the resource ID ($\overline{\text{TBST}}$||TSIZ0–TSIZ2).

**Timing** *Assertion/ Negation/ High Impedance* The same as A[0–31].

*Table 7-3. Data-Transfer Size*

| $\overline{\text{TBST}}$ | TSIZ[0–2] | Transfer Size |
|---|---|---|
| Asserted | 010 | Burst (32 bytes) |
| Negated | 000 | 8 bytes |
| Negated | 001 | 1 byte |
| Negated | 010 | 2 bytes |
| Negated | 011 | 3 bytes |
| Negated | 100 | 4 bytes |
| Negated | 101 | 5 bytes[1] |
| Negated | 110 | 6 bytes[1] |
| Negated | 111 | 7 bytes[1] |

[1] Not generated by the 750GX.

### *7.2.4.3 Transfer Burst ($\overline{\textit{TBST}}$)*

The transfer burst ($\overline{\text{TBST}}$) signal is an input/output signal on the 750GX.

*Transfer Burst ($\overline{\text{TBST}}$)—Output*

**State** *Asserted* Indicates that a burst transfer is in progress.

*Negated* Indicates that a burst transfer is not in progress.

For external control instructions (**eciwx** and **ecowx**), $\overline{\text{TBST}}$ is used to output bit 28 of the EAR, which is used to form the resource ID ($\overline{\text{TBST}}$||TSIZ0–TSIZ2).

**Timing** *Assertion/ Negation/ High Impedance* The same as A[0–31].

*Transfer Burst ($\overline{\text{TBST}}$)—Input*

**State** *Asserted/ Negated* Used when snooping for single-beat reads (read with no intent to cache).

**Timing** *Assertion/ Negation* The same as A[0–31].

### *7.2.4.4 Cache Inhibit ($\overline{\text{CI}}$)—Output*

The cache inhibit ($\overline{\text{CI}}$) signal is an output signal on the 750GX.

| | | |
|---|---|---|
| **State** | *Asserted* | Indicates that a single-beat transfer will not be cached, reflecting the setting of the I bit for the block or page that contains the address of the current transaction. |
| | *Negated* | Indicates that a burst transfer will allocate the 750GX data-cache block. |
| **Timing** | *Assertion/ Negation/ High Impedance* | The same as A[0–31]. |

### *7.2.4.5 Write-Through ($\overline{\text{WT}}$)—Output*

The write-through ($\overline{\text{WT}}$) signal is an output signal on the 750GX.

| | | |
|---|---|---|
| **State** | *Asserted* | Indicates that a single-beat write transaction is write-through, reflecting the value of the W bit for the block or page that contains the address of the current transaction. Assertion during a read operation indicates a data load. |
| | *Negated* | Indicates that a write transaction is not write-through. During a read operation, negation indicates an instruction load. |
| **Timing** | *Assertion/ Negation/ High Impedance* | The same as A[0–31]. |

### *7.2.4.6 Global ($\overline{\text{GBL}}$)*

The global ($\overline{\text{GBL}}$) signal is an input/output signal on the 750GX.

*Global ($\overline{\text{GBL}}$)—Output*

**State** *Asserted* Indicates that the transaction is global and should be snooped by other masters. $\overline{\text{GBL}}$ reflects the M bit (WIMG bits) from the memory management unit (MMU) except during certain transactions. Copybacks are always nonglobal. Instruction accesses do not reflect the M bit when the HID0[IFEM] bit (HID0 bit 23) is '0' and the instruction address translation bit (bit 26) in the Machine State Register is '1' (MSR[IR] = '1'); or if HID0[IFEM] is '1' and MSR[IR] is '0'. In either of these cases, the M bit is ignored and the access is nonglobal.

*Negated* Indicates that the transaction is not global and should not be snooped by other masters.

**Timing** *Assertion/ Negation/ High Impedance* The same as A[0–31].

*Global ($\overline{\text{GBL}}$)—Input*

**State** Asserted Indicates that a transaction must be snooped by the 750GX.

Negated Indicates that a transaction should not be snooped by the 750GX. (In addition, certain nonglobal transactions are snooped for reservation coherency. See *Table 7-1* on page 256.)

**Timing** Assertion/ Negation The same as A[0–31].

### 7.2.5 Address Transfer Termination Signals

The address transfer termination signals are used to indicate either that the address phase of the transaction has completed successfully or must be repeated, and when it should be terminated. For detailed information about how these signals interact, see *Chapter 8, Bus Interface Operation,* on page 279.

#### *7.2.5.1 Address Acknowledge (AACK)—Input*

The address acknowledge (AACK) signal is an input-only signal on the 750GX.

| | | |
|---|---|---|
| **State** | *Asserted* | Indicates that the address tenure of a transaction should be terminated. On the following cycle, the 750GX, as address-bus master, will release the address and attribute signals to high impedance, and sample ARTRY to determine a qualified ARTRY condition. Note that the address tenure will not be terminated until the assertion of AACK, even if the associated data tenure has completed. As snooper, the 750GX requires an assertion of AACK for every assertion of TS that it detects. |
| | *Negated* | During ABB, indicates that the address tenure must remain active, and that the address and attribute signals remain driven. |
| **Timing** | *Assertion/<br>Negation* | May occur as early as the bus clock cycle after TS is asserted. Assertion can be delayed to allow adequate address access time for slow devices. For example, if an implementation supports slow snooping devices, an external arbiter can postpone the assertion of AACK.<br>**Note:** If configured for 1x or 1.5x clock modes, the 750GX requires AACK to be asserted no sooner than the second cycle following the assertion of TS (one address wait state) in order to generate a snoop response (via ARTRY). |

### *7.2.5.2 Address Retry ($\overline{\text{ARTRY}}$)*

The address retry ($\overline{\text{ARTRY}}$) signal is both an input and output signal on the 750GX.

*Address Retry ($\overline{\text{ARTRY}}$)—Output*

| | | |
|---|---|---|
| **State** | *Asserted* | The 750GX as snooper indicates that the 750GX requires the snooped transaction to be rerun. The 750GX might require a snooped transaction to rerun to perform a snoop copyback first, because it is currently performing a cache reload for that line (pipeline collision on bus), or because it was unable to service the snooped address at that time. |
| | *Negated/ High Impedance* | Indicates that the 750GX does not need the snooped address tenure to be retried. |
| **Timing** | *Assertion* | Driven and asserted the second cycle following the assertion of $\overline{\text{TS}}$ if a retry is required. Remains asserted until the cycle following the assertion of $\overline{\text{AACK}}$. |
| | *Negation* | Occurs the second bus cycle after the assertion of $\overline{\text{AACK}}$. Since this signal might be simultaneously driven by multiple devices, it negates in a unique fashion. First the buffer goes to high impedance for a minimum of one-half processor cycle (dependent on the clock mode), and then it is driven negated for one-half bus cycle before returning to high impedance. This special method of negation can be disabled by setting precharge disable in HID0[PAR]. |

*Address Retry ($\overline{\text{ARTRY}}$)—Input*

**State** *Asserted* If the 750GX is the address-bus master, $\overline{\text{ARTRY}}$ indicates that the 750GX must retry the preceding address tenure and immediately negate $\overline{\text{BR}}$ (if asserted). If the associated data tenure has already started, the 750GX also cancels the data tenure immediately, even if the burst data has been received. If the 750GX is not the address-bus master, this input indicates that the 750GX should immediately negate $\overline{\text{BR}}$ to allow an opportunity for a copy-back operation to main memory after a snooping bus master asserts ARTRY. Note that the subsequent address presented on the address bus might not be the same one associated with the assertion of the ARTRY signal.

*Negated/ High Impedance* Indicates that the 750GX does not need to retry the last address tenure.

**Timing** *Assertion* May occur as early as the second cycle following the assertion of $\overline{\text{TS}}$, and must occur by the bus clock cycle immediately following the assertion of $\overline{\text{AACK}}$ if an address retry is required.

*Negation/ High Impedance* Must occur the second cycle following the assertion of $\overline{\text{AACK}}$ (if $\overline{\text{ARTRY}}$ was asserted).

**Note:** During the second cycle following the assertion of $\overline{\text{AACK}}$, $\overline{\text{ARTRY}}$ is first set to the high impedance state by the asserting masters, and might be sampled in an *indeterminate* state.

### 7.2.6 Data-Bus Arbitration Signals

Like the address-bus arbitration signals, data-bus arbitration signals maintain an orderly process for determining data-bus mastership. Note that there is no data-bus arbitration signal equivalent to the address-bus arbitration signal $\overline{\text{BR}}$ (bus request), because, except for address-only transactions, $\overline{\text{TS}}$ implies data-bus requests. For a detailed description of how these signals interact, see *Section 8.4.1, Data-Bus Arbitration,* on page 301.

One special signal, $\overline{\text{DBWO}}$, allows the 750GX to be configured dynamically to write data out of order with respect to read data. For detailed information about using the $\overline{\text{DBWO}}$, see *Section 8.9, Using Data-Bus Write-Only, on page 320*.

#### *7.2.6.1 Data-Bus Grant ($\overline{\text{DBG}}$)—Input*

The data-bus grant ($\overline{\text{DBG}}$) signal is an input-only signal on the 750GX.

**State** *Asserted* Indicates that the 750GX may, with proper qualification, assume mastership of the data bus. A qualified bus grant occurs when $\overline{\text{DBG}}$ is asserted and $\overline{\text{DBB}}$, $\overline{\text{DRTRY}}$, and $\overline{\text{ARTRY}}$ are not asserted. $\overline{\text{ARTRY}}$ is only for the address-bus tenure associated with the data-bus tenure about to be granted (that is, not from another address tenure due to address pipelining).

*Negated* Indicates that the 750GX is not granted next data-bus ownership.

**Timing** *Assertion* Might occur on any cycle; not recognized until the cycle $\overline{\text{TS}}$ is asserted, or later.

*Negation* Might occur on any cycle to indicate the 750GX cannot assume data-bus ownership.

### 7.2.6.2 Data-Bus Write-Only ($\overline{\text{DBWO}}$)

The data-bus write-only ($\overline{\text{DBWO}}$) signal is an input-only signal on the 750GX.

**State** *Asserted* If two or more data tenure requests are pending for the 750GX due to address pipelining, indicates that the 750GX should run the data-bus tenure for the next pipelined write transaction *even if a read address tenure was pipelined on the bus before the write address tenure.* $\overline{\text{DBWO}}$ allows write data tenures to be run ahead of read data tenures. However, it does not allow write data tenures to be run ahead of other write data tenures. If no write requests are pending, the 750GX will ignore $\overline{\text{DBWO}}$ and assume data-bus ownership for the next pending read request.

*Negated* Indicates that the 750GX must run its read and write data-bus tenures in the same order as their address tenures.

**Timing** *Assertion/ Negation* Sampled by the 750GX only on the clock that a qualified $\overline{\text{DBG}}$ is recognized.

*Start-Up* See *Table 7-6, Summary of Mode Select Signals*, on page 274 for a description of the start-up function.

### 7.2.6.3 Data Bus Busy ($\overline{\text{DBB}}$)

The data bus busy ($\overline{\text{DBB}}$) signal is both an input and output signal on the 750GX.

*Data Bus Busy ($\overline{\text{DBB}}$)—Output*

**State** *Asserted* Indicates that the 750GX is the current data-bus owner. The 750GX will always assume data-bus ownership if it needs the data bus and determines a qualified data-bus grant (see $\overline{\text{DBG}}$).

*Negated* Indicates that the 750GX is not the current data-bus owner, unless the data retry ($\overline{\text{DRTRY}}$) signal has extended the data tenure for the last or only data beat.

**Timing** *Assertion* Occurs the cycle following a qualified $\overline{\text{DBG}}$. Remains asserted for the duration of the data tenure.

*Negation* Negates for a fraction of a bus cycle (one-half minimum, depends on clock mode) starting the cycle following the final assertion of the transfer acknowledge ($\overline{\text{TA}}$) signal, or following the transfer error acknowledge ($\overline{\text{TEA}}$) signal or certain $\overline{\text{ARTRY}}$ cases. Then releases to the high impedance state.

*Data Bus Busy ($\overline{\text{DBB}}$)—Input*

**State** *Asserted* Indicates that another master is the current data-bus owner.

*Negated* Indicates that the data bus might be available for use by the 750GX (see $\overline{\text{DBG}}$).

**Timing** *Assertion* May occur when the 750GX must be prevented from using the data bus.

*Negation* May occur whenever the 750GX can use the data bus.

### 7.2.7 Data-Transfer Signals

Like the address transfer signals, the data-transfer signals are used to transmit data and to generate and monitor parity for the data transfer. For a detailed description of how the data-transfer signals interact, see *Section 8.4.3, Data Transfer,* on page 303. Data parity is optional on the 750GX.

#### *7.2.7.1 Data Bus (DH[0–31], DL[0–31])*

The data bus (DH[0–31] and DL[0–31]) consists of 64 signals that are both inputs and outputs on the 750GX.

**State** The data bus has two halves—data bus high (DH) and data bus low (DL). See *Table 7-4* on page 266 for the data-bus lane assignments.

**Timing** The data bus is driven once for noncached transactions and four times for cache transactions (bursts).

*Table 7-4. Data-Bus Lane Assignments*

| Data-Bus Signals | Byte Lane |
|---|---|
| DH[0–7] | 0 |
| DH[8–15] | 1 |
| DH[16–23] | 2 |
| DH[24–31] | 3 |
| DL[0–7] | 4 |
| DL[8–15] | 5 |
| DL[16–23] | 6 |
| DL[24–31] | 7 |

*Data Bus (DH[0–31], DL[0–31])—Output*

**State** *Asserted/ Negated* Represents the state of data during a data write. For single-beat (cache inhibited or write through) writes, byte lanes not selected for data transfer will not supply valid data (no data mirroring).

**Timing** *Assertion/ Negation* First or only beat begins on the cycle of $\overline{\text{DBB}}$ assertion and, for bursts, transitions on the cycle following each initially qualified assertion of $\overline{\text{TA}}$.

*High Impedance* Occurs on the bus clock cycle after the final assertion of $\overline{\text{TA}}$, following the assertion of $\overline{\text{TEA}}$, or in certain $\overline{\text{ARTRY}}$ cases.

*Data Bus (DH[0–31], DL[0–31])—Input*

**State** *Asserted/ Negated* Represents the state of data during a data read transaction.

**Timing** *Assertion/ Negation* Data must be valid on the same bus clock cycle that $\overline{\text{TA}}$ is asserted, even if during the last assertion cycle of $\overline{\text{DRTRY}}$.

### *7.2.7.2 Data-Bus Parity (DP[0–7])*

The eight data-bus parity (DP[0–7]) signals are both input and output signals.

*Data-Bus Parity (DP[0–7])—Output*

**State** *Asserted/ Negated* Represents odd parity for each of the 8 bytes of data write transactions. Odd parity means that an odd number of bits, including the parity bit, are driven high. The generation of parity is enabled through HID0. The signal assignments are listed in *Table 7-5*.

**Timing** *Assertion/ Negation/ High Impedance* The same as DL[0–31].

*Table 7-5. DP[0–7] Signal Assignments*

| Signal Name | Signal Assignments |
|---|---|
| DP0 | DH[0–7] |
| DP1 | DH[8–15] |
| DP2 | DH[16–23] |
| DP3 | DH[24–31] |
| DP4 | DL[0–7] |
| DP5 | DL[8–15] |
| DP6 | DL[16–23] |
| DP7 | DL[24–31] |

*Data-Bus Parity (DP[0–7])—Input*

| | | |
|---|---|---|
| **State** | *Asserted/ Negated* | Represents odd parity for each byte of read data. Parity is checked on all data byte lanes, regardless of the size of the transfer. Detected even parity causes a checkstop if data-parity errors are enabled in the HID0 register. |
| **Timing** | *Assertion/ Negation* | The same as DL[0–31]. |

### *7.2.7.3 Data Bus Disable (DBDIS)—Input*

| | | |
|---|---|---|
| **State** | *Asserted* | Indicates (for a write transaction) that the 750GX must release the data bus and data-bus parity to high impedance during the following cycle. The data tenure will remain active, DBB will remain driven, and the transfer termination signals will still be monitored by the 750GX. |
| | *Negated* | Indicates the data bus should remain driven if it otherwise would have been. DBDIS is ignored during read transactions. |
| **Timing** | *Assertion/ Negation* | May be asserted on any clock; will not otherwise affect the operation of the bus if the 750GX is not running a bus transaction or if the 750GX is running a read transaction. |
| | *Start-Up* | See *Table 7-6, Summary of Mode Select Signals*, on page 274 for a description of the start-up function. |

### 7.2.8 Data-Transfer Termination Signals

Data termination signals are required after each data beat in a data transfer. Note that in a single-beat transaction, the data termination signals also indicate the end of the tenure, while in burst accesses, the data termination signals apply to individual beats and indicate the end of the tenure only after the final data beat.

For a detailed description of how these signals interact, see *Section 8.4.4, Data-Transfer Termination,* on page 303.

### *7.2.8.1 Transfer Acknowledge (TA)—Input*

| | | |
|---|---|---|
| **State** | *Asserted* | Indicates that data on the data bus has been provided or accepted by the system. On the following cycle, the 750GX will terminate the data beat (unless DRTRY extends a read data beat), and if a burst, advance to the next data beat. If it is the last or only data beat, the 750GX will also terminate the data tenure (unless DRTRY extends a read data beat). TA must always be asserted on the same cycle as valid data on the data bus, even if during the final assertion cycle of DRTRY for that beat. |
| | *Negated* | Indicates that the 750GX must extend the current data beat (by inserting wait states) until data can be provided or accepted by the system. TA might also be negated anytime during a DRTRY assertion except on the last cycle of the DRTRY assertion. |

**Timing** *Assertion* Might occur on any cycle during the normal or extended data-bus tenure for the 750GX (see $\overline{\text{DBB}}$ and $\overline{\text{DRTRY}}$). Must not occur two cycles or more before $\overline{\text{ARTRY}}$ assertion if $\overline{\text{ARTRY}}$ cancellation is to be used.

*Negation* For a burst, must occur the cycle after the assertion of $\overline{\text{TA}}$ unless another assertion of $\overline{\text{TA}}$ is immediately required for the next data beat.

**Note:** It is the responsibility of the system to ensure that $\overline{\text{TA}}$ is negated by the start of the next data-bus tenure.

***Warning***: If configured for 1x clock mode and performing a data (not instruction) burst read, the 750GX requires one wait state between the assertion of $\overline{\text{TS}}$ and the first assertion of $\overline{\text{TA}}$. If No-DRTRY mode is also selected, the 750GX requires two wait states for 1x clock mode, or one wait state for 1.5x clock mode.

### *7.2.8.2 Data Retry ($\overline{\text{DRTRY}}$)—Input*

**State** *Asserted* During a read transaction, indicates that the 750GX must cancel data received on the previous cycle with a valid $\overline{\text{TA}}$, and extend that data beat until new valid data with a new $\overline{\text{TA}}$ is provided. While asserted, $\overline{\text{DRTRY}}$ also extends the data-bus tenure of the current transaction if the last or only data beat was retried and $\overline{\text{DBB}}$ has already negated.

*Negated* Indicates that read data presented with $\overline{\text{TA}}$ on the previous bus cycle is valid.

$\overline{\text{DRTRY}}$ is ignored as a data termination control during write transactions.

**Timing** *Assertion* Must occur the cycle following the assertion of $\overline{\text{TA}}$, if a data retry is required. Once asserted must remain asserted until a valid $\overline{\text{TA}}$ and data are provided.

*Negation* Must occur the cycle following the presentation of valid data and a $\overline{\text{TA}}$ to the 750GX. This might occur several cycles after the negation of $\overline{\text{DBB}}$.

*Start-Up* Sampled at the negation of the hard reset ($\overline{\text{HRESET}}$) signal to select DRTRY-enabled mode if negated, or No-DRTRY mode if asserted. See *Table 7-6, Summary of Mode Select Signals*, on page 274 for a description of the start-up function.

### *7.2.8.3 Transfer Error Acknowledge ($\overline{\text{TEA}}$)—Input*

**State** *Asserted* Indicates that a data-bus error has occurred. On the following cycle, the 750GX must terminate the data tenure. Internally, the 750GX will also take a machine-check interrupt or enter the checkstop state (see *Chapter 10, Power and Thermal Management,* on page 335). For reads, a $\overline{\text{TEA}}$ will not invalidate data entering the General Purpose Registers (GPRs) or the caches.

*Negated* Indicates that no bus error was detected.

**Timing** *Assertion/ Negation* Assertion might occur on any cycle during the normal or extended data-bus tenure for the 750GX (during $\overline{\text{DBB}}$, and the cycle after $\overline{\text{TA}}$ during reads). Assertion should occur for one cycle only.

It is the responsibility of the system to ensure that $\overline{\text{TEA}}$ is negated by the start of the next data-bus tenure.

### 7.2.9 System Status Signals

Most system status signals are input signals that indicate when exceptions are received, when checkstop conditions have occurred, and when the 750GX must be reset. The 750GX generates the output signal, $\overline{\text{CKSTP\_OUT}}$, when it detects a checkstop condition.

#### *7.2.9.1 Interrupt ($\overline{\text{INT}}$)— Input*

**State** *Asserted* The 750GX initiates an interrupt if MSR[EE] is set. Otherwise, the 750GX ignores the interrupt. To guarantee that the 750GX will take the external interrupt, $\overline{\text{INT}}$ must be held active until the 750GX takes the interrupt. Otherwise, whether the 750GX takes an external interrupt depends on whether the MSR[EE] bit was set while the $\overline{\text{INT}}$ signal was held active.

*Negated* Indicates that normal operation should proceed.

**Timing** *Assertion* May occur at any time and may be asserted asynchronously to the input clocks. The $\overline{\text{INT}}$ input is level-sensitive.

*Negation* Should not occur until an interrupt is taken.

#### *7.2.9.2 System Management Interrupt ($\overline{\text{SMI}}$)—Input*

**State** *Asserted* The 750GX initiates a system management interrupt operation if the MSR[EE] is set. Otherwise, the 750GX ignores the exception condition. The system must hold $\overline{\text{SMI}}$ active until the exception is taken.

*Negated* Indicates that normal operation should proceed.

**Timing** *Assertion* May occur at any time and may be asserted asynchronously to the input clocks. The $\overline{\text{SMI}}$ input is level-sensitive.

*Negation* Should not occur until an interrupt is taken.

### 7.2.9.3 Machine-Check Interrupt (MCP)—Input

**State** *Asserted* The 750GX initiates a machine-check interrupt operation if MSR[ME] and HID0[EMCP] are set. If MSR[ME] is cleared and HID0[EMCP] is set, the 750GX must terminate operation by internally gating off all clocks, and releasing all outputs (except CKSTP_OUT) to the high-impedance state. If HID0[EMCP] is cleared, the 750GX ignores the interrupt condition. The MCP signal must be held asserted for two bus clock cycles.

*Negated* Indicates that normal operation should proceed.

**Timing** *Assertion* May occur at any time and may be asserted asynchronously to the input clocks. The MCP input is negative edge-sensitive.

*Negation* May be negated two bus cycles after assertion.

### 7.2.9.4 Checkstop Input (CKSTP_IN)—Input

**State** *Asserted* Indicates that the 750GX must enter the checkstop state and terminate operation. The 750GX will internally gate off all clocks and remain in this state while CKSTP_IN is asserted. The 750GX will also release all outputs (except CKSTP_OUT) to the high-impedance state. CKSTP_IN is not maskable. Once CKSTP_IN has been asserted it must remain asserted until the system has been reset.

*Negated* Indicates that normal operation should proceed.

**Timing** *Assertion* May occur at any time and may be asserted asynchronously to the input clocks.

*Negation* May occur any time after the CKSTP_IN output has been asserted.

### 7.2.9.5 Checkstop Output (CKSTP_OUT)—Output

Note that the CKSTP_OUT signal is an open-drain type output, and requires an external pull-up resistor (for example, 10 kΩ to $V_{DD}$) to assure proper deassertion of the CKSTP_OUT signal.

**State** *Asserted* Indicates that a checkstop condition has been detected, and the processor has ceased operation.

*Negated* Indicates that the processor is operating normally.

**Timing** *Assertion* Might occur at any time and can be asserted asynchronously to input clocks.

*High Impedance* Requires HRESET.

**Note:** CKSTP_OUT operates as an open-drain output. It will either be in the asserted or high-impedance state.

### 7.2.10 Reset Signals

There are two reset signals on the 750GX—hard reset ($\overline{\text{HRESET}}$) and soft reset ($\overline{\text{SRESET}}$). Descriptions of the reset signals follows.

#### *7.2.10.1 Hard Reset ($\overline{\text{HRESET}}$)—Input*

The hard reset ($\overline{\text{HRESET}}$) signal must be used at power-on in conjunction with the test reset ($\overline{\text{TRST}}$) signal to properly reset the processor.

| | | |
|---|---|---|
| **State** | *Asserted* | Initiates a complete hard reset operation when this input transitions from asserted to negated. Causes a reset exception as described in *Section 4.5.1, System Reset Exception (0x00100),* on page 163 Output drivers are released to high impedance within five clocks after the assertion of $\overline{\text{HRESET}}$. |
| | *Negated* | Indicates that normal operation should proceed. |
| **Timing** | *Assertion* | May occur at any time and may be asserted asynchronously to the 750GX input clock. Must be held asserted for a minimum of 255 clock cycles after the PLL lock time has been met. See the 750GX hardware specifications for further timing comments. Falling-edge activated. |
| | *Negation* | May occur any time after the minimum reset pulse width has been met. |

#### *7.2.10.2 Soft Reset ($\overline{\text{SRESET}}$)—Input*

The soft reset input provides warm reset capability. This input can be used to avoid forcing the 750GX to complete the cold start sequence.

| | | |
|---|---|---|
| **State** | *Asserted* | Initiates processing for a reset exception as described in *Section 4.5.1, System Reset Exception (0x00100),* on page 163. |
| | *Negated* | Indicates that normal operation should proceed. |
| **Timing** | *Assertion* | May occur at any time and may be asserted asynchronously to the 750GX input clock. The $\overline{\text{SRESET}}$ input is negative edge-sensitive. |
| | *Negation* | May be negated two bus cycles after assertion. |

### 7.2.11 Processor Status Signals

Processor status signals indicate the state of the processor. They include the memory reservation signal, machine quiesce control signals, time-base enable signal, and TLB Invalidate Synchronize ($\overline{\text{TLBISYNC}}$) signal.

#### *7.2.11.1 Quiescent Request ($\overline{\text{QREQ}}$)—Output*

| | | |
|---|---|---|
| **State** | *Asserted* | Indicates that the 750GX is requesting all bus activity normally required to be snooped to terminate or to pause so the 750GX can enter a quiescent (low power) state. When the 750GX has entered a quiescent state, it no longer snoops bus activity. See *Chapter 10, Power and Thermal Management,* on page 335. |
| | *Negated* | Indicates that the 750GX is not making a request to enter the quiescent state. |
| **Timing** | *Assertion/ Negation* | Might occur on any cycle. $\overline{\text{QREQ}}$ will remain asserted for the duration of the quiescent state. |

#### *7.2.11.2 Quiescent Acknowledge ($\overline{\text{QACK}}$)—Input*

| | | |
|---|---|---|
| **State** | *Asserted* | Indicates that all bus activity has terminated or paused, and that the 750GX might enter nap or sleep mode. |
| | *Negated* | Indicates that the 750GX cannot enter nap or sleep mode, or that it must return to doze mode from nap mode in order to snoop. |
| **Timing** | *Assertion/ Negation* | May occur on any cycle following the assertion of $\overline{\text{QREQ}}$. Must be negated for at least eight bus cycles prior to performing any snoop cycles to ensure that the 750GX has returned to doze mode from nap mode. |
| | *Start-Up* | See *Table 7-6, Summary of Mode Select Signals*, on page 274 for a description of the start-up function. |

#### *7.2.11.3 Reservation (RSRV)—Output*

| | | |
|---|---|---|
| **State** | *Asserted/ Negated* | Represents the state of the internal reservation coherency bit used by the **lwarx** and **stwcx.** instructions. See *Section 8.7.1, Support for the lwarx and stwcx. Instruction Pair,* on page 319. |
| **Timing** | *Assertion/ Negation* | Might occur on any cycle. Will occur immediately following a transition of the reservation coherency bit. |

### *7.2.11.4 Time Base Enable (TBEN)—Input*

**State** *Asserted* Indicates that the time base and decrementer should continue clocking. This signal is essentially a “count enable” control for the time base and decrementer counter.

*Negated* Indicates that the time base and decrementer should stop clocking.

**Timing** *Assertion/ Negation* May occur on any cycle. The sampling of this signal is synchronous with SYSCLK.

### *7.2.11.5 TLB Invalidate Synchronize (TLBISYNC)—Input*

The TLB Invalidate Synchronize (TLBISYNC) signal is an input-only signal.

**State** *Asserted* Prevents execution of a **tlbsync** instruction from completing.

*Negated* Enables execution of a **tlbsync** to complete.

**Timing** *Assertion/ Negation* Might occur on any cycle.

*Start-Up* See *Table 7-6, Summary of Mode Select Signals*, on page 274 for a description of the start-up function.

### 7.2.12 Processor Mode Selection Signals

*Table 7-6* summarizes the processor mode select signals of the 750GX. The mode select signals establish the operating modes of the processor after reset (for example, 32-bit or 64-bit data mode or DRTRY mode). Mode select signals are sampled at the rising transition of HRESET.

*Table 7-6. Summary of Mode Select Signals*

| Pin | Description | Pin state at HRESET transition | |
|---|---|---|---|
| | | '0' | '1' |
| DRTRY | Selects DRTRY mode. | No-DRTRY mode | DRTRY mode |
| QACK | QACK selects normal or full cycle precharge on ABB, DBB, and ARTRY. | Full cycle precharge | Normal precharge |
| TLBISYNC | TLBISYNC selects 32-bit or 64-bit bus mode. | 32-bit mode | 64-bit mode |
| DBWO | Factory usage mode only. Must be tied high at HRESET transition. | N/A | Required |
| DBDIS | Factory usage mode only. Must be tied high at HRESET transition. | N/A | Required |
| L2_TSTCLK | Factory usage mode only. Must be tied high at HRESET transition. | N/A | Required |

### 7.2.13 I/O Voltage Select Signals

*Table 7-7* shows the settings for the I/O voltage signals.

*Table 7-7. Bus Voltage Selection Settings*

| Voltage Selection | $OV_{DD}$ Select #1<br>BVSEL | $OV_{DD}$ Select #2<br>L1TSTCLK |
|---|---|---|
| Reserved | 0 | 0 |
| 1.8 V | 0 | 1 |
| 2.5 V | 1 | 1 |
| 3.3 V | 1 | 0 |

### 7.2.14 Test Interface Signals

The processor provides two sets of pins for controlling JTAG and level-sensitive scan design (LSSD) testing.

#### *7.2.14.1 IEEE 1149.1a-1993 Interface Description*

The 750GX has five dedicated JTAG signals, which are described in *Table 7-8*. The test data input (TDI) and test data output (TDO) scan ports are used to scan instructions, as well as data into the various scan registers for JTAG operations. The scan operation is controlled by the test access port (TAP) controller, which in turn is controlled by the test mode select (TMS) input sequence. The scan data is latched in at the rising edge of the test clock (TCK). Test reset ($\overline{\text{TRST}}$) is a JTAG optional signal, which is used to reset the TAP controller asynchronously. The $\overline{\text{TRST}}$ signal assures that the JTAG logic does not interfere with the normal operation of the chip, and must be asserted and deasserted coincident with the assertion of the $\overline{\text{HRESET}}$ signal.

*Table 7-8. IEEE Interface Pin Descriptions*

| Signal Name | Input/Output | Weak Pullup Provided | IEEE 1149.1a-1993 Function | Timing Comments |
|---|---|---|---|---|
| TDI | Input | Yes | Serial scan input signal | Asserted/Negated—Not used during normal operation. TMS, TDI, and $\overline{\text{TRST}}$ have internal pullups provided; TCK does not. For normal operation, TMS and TDI may be left unconnected, TCK must be set high or low, and $\overline{\text{TRST}}$ must be asserted sometime during power-up for JTAG logic initialization. |
| TDO | Output | No | Serial scan output signal | |
| TMS | Input | Yes | TAP controller mode signal | |
| TCK | Input | No | Scan clock | |
| $\overline{\text{TRST}}$ | Input | Yes | TAP controller reset | |

#### *7.2.14.2 $\overline{\text{LSSD\_MODE}}$*

**State** *Asserted* LSSD test enable. The LSSD test enable signal is an input-only signal.

**Timing** *Assertion/Negation* Must be set high by the system during normal operation.

### *7.2.14.3 L1_TSTCLK*

**State** LSSD test clock in test mode, and bus voltage select in functional mode. See *Table 7-7, Bus Voltage Selection Settings*, on page 275.

**Timing** *Assertion/ Negation* Signal should be held to a constant value for I/O voltage selection.

### *7.2.14.4 L2_TSTCLK*

**State** Reserved pin that must be negated for system operation.

**Timing** *Assertion/ Negation* Must be held constant for system operation.

*Start-Up* See *Table 7-6, Summary of Mode Select Signals*, on page 274 for a description of the start-up function.

### *7.2.14.5 BVSEL*

**State** I/O voltage is selectable through using the BVSEL pin and L1_TSTCLK pin. See *Table 7-7, Bus Voltage Selection Settings*, on page 275.

**Timing** Signal should be held to a constant value for I/O voltage selection.

## 7.2.15 Clock Signals

The 750GX requires a single system clock input (SYSCLK). This input represents the frequency at which the bus interface for the 750GX will operate. Internally, the 750GX uses a PLL circuit to generate a master core clock that is frequency-multiplied and phase-locked to the SYSCLK input. This master core clock is the clock actually used by the 750GX to operate the internal circuitry. The PLL samples the master clock at the latch boundary (that is, end of clock tree) and minimizes the clock skew between the rising edge of SYSCLK and the master clock at the latch boundary. This mechanism provides I/O timings accurate to the rising edge of SYSCLK. However, if the chip is operated in bypass mode (PLL not used), this phase correcting circuitry cannot be used, and the I/O timings are unreliable.

The PLL is configured by the PLL_CFG(0:4) pins. These pins select the multiplier that the PLL will use to multiply the SYSCLK frequency up to the internal core frequency. In addition, the pins PLL_RNG(0:1) must be set to select the appropriate frequency operating range of the PLL. See the *PowerPC 750GX Datasheet* for more information.

### *7.2.15.1 System Clock (SYSCLK)—Input*

The 750GX requires a single system clock (SYSCLK) input. This input sets the frequency of operation for the bus interface. Internally, the 750GX uses a PLL circuit to generate a master clock for all of the CPU circuitry (including the bus interface circuitry) which is phase-locked to the SYSCLK input.

**State** *Asserted/Negated* The primary clock input for the 750GX. SYSCLK represents the bus clock frequency for bus operation. Internally, the processor core will be operating at an integer or half-integer multiple (≥ 1.0) of the bus clock frequency.

**Timing** *Assertion/Negation* See the *IBM PowerPC 750GX RISC Microprocessor Datasheet* for timing comments. Loose duty cycle allowed.

**Note:** SYSCLK is used as a frequency reference for the internal PLL clock regenerator. It must not be suspended or varied during normal operation to ensure proper PLL operation.

### *7.2.15.2 Clock Out (CLK_OUT)—Output*

The clock out (CLK_OUT) signal is an output signal.

**State** *Asserted/Negated* PLL clock output for PLL testing or monitoring. (See HID1 for select and enable.)

The CLK_OUT signal is provided for testing only.

**Timing** *Assertion/Negation* See the *IBM PowerPC 750GX RISC Microprocessor Datasheet*. Driven with a bus-rate clock during the assertion of $\overline{\text{HRESET}}$. The default state during normal operation is high impedance.

### *7.2.15.3 PLL Configuration (PLL_CFG[0:4])—Input*

The PLL is configured by the PLL_CFG[0:4] signals. For a given SYSCLK (bus) frequency, the PLL configuration signals set the internal CPU frequency of operation. See the *PowerPC 750GX Datasheet* for PLL configuration.

**State** *Asserted/Negated* Configures the operation of the PLL and the internal processor clock frequency. Settings are based on the desired bus and internal frequency of operation.

**Timing** *Assertion/Negation* Must remain stable during operation; should only be changed during the assertion of $\overline{\text{HRESET}}$. These bits can be read through the PCE[0–4] bits in the HID1 register.

#### *7.2.15.4 PLL Range (PLL_RNG[0:1])—Input*

| | | |
|---|---|---|
| **State** | *Asserted/ Negated* | Configures the PLL operating-frequency range. Internal core clock frequency must be within the specified range. |
| **Timing** | *Asserted/ Negated* | Must remain stable during normal operation; should only be changed during the assertion of $\overline{\text{HRESET}}$. These bits are readable through bits PRE[5:6] in the HID1. |

### 7.2.16 Power and Ground Signals

The 750GX provides the following connections for power and ground:

- $V_{DD}$—The $V_{DD}$ signals provide the supply voltage connection for the processor core.
- $OV_{DD}$—The $OV_{DD}$ signals provide the supply voltage connection for the system interface drivers.
- $AV_{DD}$—The $AV_{DD}$ power signal provides power to the clock generation phase-locked loop. See the *PowerPC 750GX Datasheet* for information on how to use this signal.
- GND and OGND—The GND and OGND signals provide the connection for grounding the 750GX. On the 750GX, there is no electrical distinction between the GND and OGND signals.

# 8. Bus Interface Operation

This chapter describes the PowerPC 750GX microprocessor’s bus interface and its operation. It shows how the 750GX signals, defined in *Chapter 7, Signal Descriptions,* on page 249, interact to perform address and data transfers.

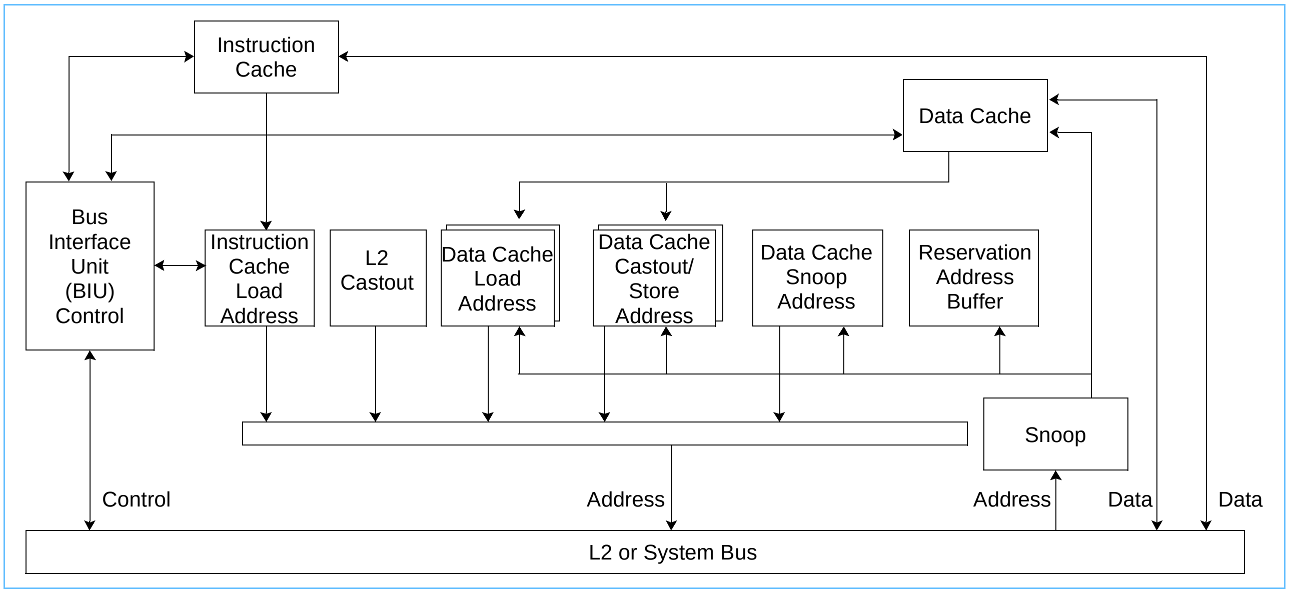
The bus interface buffers bus requests from the instruction and data caches, and executes the requests per the 60x bus protocol. It includes Address Register queues, prioritizing logic, and bus control logic. It captures snoop addresses for snooping in the cache and in the Address Register queues. It also snoops for reservations and holds the touch load address for the cache. All data storage for the Address Register buffers (load-and-store data buffers) are located in the cache section. The data buffers are considered temporary storage for the cache and not part of the bus interface.

The general functions and features of the bus interface are:

- Eight Address Register buffers that include the following:
  - Instruction-cache load address buffer
  - Four data-cache load address buffers
  - Two data-cache castout/store address buffers
  - Data-cache snoop copy-back address buffer (associated data block buffer located in cache)
  - Reservation address buffer for snoop monitoring
  - L2 castout buffer
- Pipeline collision detection for data-cache buffers
- Reservation address snooping for Load Word and Reserve Indexed (**lwarx**) and Store Word Conditional Indexed (**stwcx.**) instructions
- Address pipelining for four load/store transactions and one snoop transaction
- Load ahead of store capability

*Figure 8-1* on page 280 provides a conceptual block diagram of the bus interface. The Address Register queues in the figure hold transaction requests that the bus interface can issue on the bus independently of the other requests. The bus interface can have up to four load/store transactions operating on the bus at any given time through the use of address pipelining. Enabling MuM allows four cache reloads or cache inhibited loads to be pipelined in a continuous fashion on the 60x bus.  If there is a miss in the L2 cache, then the request is passed on to the BIU via three additional L2-to-BIU reload-request queues.  Data returned from the bus is loaded into the data-cache reload buffer, one of the L2 reload buffers, and the critical word is forwarded to the load/store unit.  For a D-cache-line load due to the cache miss of a load instruction, the critical double word is simultaneously written to the 256-bit line fill buffer and forwarded to the requesting load/store unit.

*Figure 8-1. Bus Interface Address Buffers*



## 8.1 Bus Interface Overview

The bus 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 bus 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 four levels of pipelining for load/store transactions. That is, with certain restrictions discussed later, there can be four outstanding load/store 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 (a maximum of four per cycle) 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 750GX encounters an instruction or data access, it calculates the logical address (effective address in the architecture specification) and uses the low-order address bits to check for a hit in the L1 32-KB 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 (real address in the architecture specification). 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 750GX performs hardware table-search operations following translation lookaside buffer (TLB) misses, L2 cache castout operations when the least-recently used (LRU) 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 1-1, 750GX Microprocessor Block Diagram,* on page 25 shows the address path from the execution units and instruction fetcher, through the translation logic to the caches and bus interface logic.

The 750GX uses separate address and data buses and a variety of control and status signals to perform reads and writes. The address bus is 32 bits wide, and the data bus is 64 bits wide. The interface is synchronous—all 750GX inputs are sampled at, and all outputs are driven from, the rising edge of the bus clock. The processor runs at a multiple of the bus-clock speed.

### 8.1.1 Operation of the Instruction and Data L1 Caches

The 750GX provides independent instruction and data L1 caches. Each cache is a physically-addressed, 32-KB cache with 8-way set associativity. Both caches consist of 128 sets of eight cache lines, with eight words in each cache line.

Because the data cache on the 750GX is an on-chip, write-back primary cache, the predominant type of transaction for most applications is burst-read memory operations, followed by burst-write memory operations and single-beat (noncacheable or write-through) memory read and write operations. Additionally, there can be address-only operations, variants of the burst and single-beat operations (that is, global memory operations that are snooped, and atomic memory operations), and address retry activity (that is, when a snooped read access hits a modified line in the cache).

Since the 750GX data-cache tags are single ported, simultaneous load or store and snoop accesses cause resource contention. Snoop accesses have the highest priority and are given first access to the tags, unless the snoop access coincides with a tag write, in which case the snoop is retried and must rearbitrate for access to the cache. Loads or stores that are deferred due to snoop accesses are performed on the clock cycle following the snoop.

The 750GX supports a 3-state coherency protocol that supports the modified, exclusive, and invalid (MEI) cache states. The protocol is a subset of the modified, exclusive, shared, and invalid (MESI) 4-state protocol and operates coherently in systems that contain 4-state caches. With the exception of the Data Cache Block Set to Zero (**dcbz**) instruction,[1] the 750GX does not broadcast cache-control instructions. The cache-control instructions are intended for the management of the local cache, but not for other caches in the system.

Instruction-cache lines in the 750GX are loaded in four beats of 64 bits each. The burst load is performed as critical double word first. The critical double word is simultaneously written to the cache and forwarded to the instruction prefetch unit, thus minimizing stalls due to load delays. If subsequent loads follow in sequential order, the instructions will be forwarded to the requesting unit as the cache block is written.

Data-cache lines in the 750GX are loaded into the cache in one cycle of 256 bits. For a cache-line load due to the cache miss of a load instruction, the critical double word is simultaneously written to the 256-bit line fill buffer and forwarded to the requesting load/store unit. If subsequent loads follow in sequential order, the data will be forwarded to the load/store unit as the cache block is written into the cache.

[1] And the Data Cache Block Invalidate (**dcbi**), Data Cache Block Store (**dcbst**), and Data Cache Block Flush (**dcbf**) instructions, if the address broadcast enable bit in Hardware-Implementation-Dependent 0 Register (HID0[ABE]) is enabled.

Cache lines are selected for replacement based on a pseudo least-recently-used (PLRU) algorithm. Each time a cache line is accessed, it is tagged as the most-recently-used line of the set. When a miss occurs, and all eight lines in the set are marked as valid, the least recently used line is replaced with the new data. When data to be replaced is in the modified state, the modified data is written into a write-back buffer while the missed data is being read from memory. When the load completes, the 750GX then pushes the replaced line from the write-back buffer to the L2 cache (if enabled), or to main memory in a burst write operation.

### 8.1.2 Operation of the Bus Interface

Memory accesses can occur in single-beat (1, 2, 3, 4, and 8 bytes) and 4-beat (32 bytes) burst data transfers. The address and data buses are independent for memory accesses to support pipelining and split transactions. The 750GX can pipeline as many as four load/store transactions and has limited support for out-of-order split-bus transactions.

Access to the bus interface is granted through an external arbitration mechanism that allows devices to compete for bus mastership. This arbitration mechanism is flexible, allowing the 750GX to be integrated into systems that implement various fairness and bus-parking procedures to avoid arbitration overhead.

Typically, memory accesses are weakly ordered to maximize the efficiency of the bus without sacrificing coherency of the data. The 750GX allows load operations to bypass store operations (except when a dependency exists). In addition, the 750GX can be configured to reorder high-priority store operations ahead of lower-priority store operations. Because the processor can dynamically optimize run-time ordering of load/store traffic, overall performance is improved.

**Note:** The Synchronize (**sync**) and Enforce In-Order Execution of I/O (**eieio**) instructions can be used to enforce strong ordering.

The following sections describe how the 750GX interface operates and provide detailed timing diagrams that illustrate how the signals interact. A collection of more general timing diagrams are included as examples of typical bus operations. *Figure 8-2* on page 283 is a legend of the conventions used in the timing diagrams.

This is a synchronous interface—all 750GX input signals are sampled and output signals are driven on the rising edge of the bus clock cycle (see the *PowerPC 750GX Datasheet* for exact timing information).

### 8.1.3 Bus Signal Clocking

All signals for the 750GX bus interface are specified with respect to the rising-edge of the external system clock input (SYSCLK), and they are guaranteed to be sampled as inputs or changed as outputs with respect to that edge.

**System Implementation Note:** Since the same clock edge is referenced for driving or sampling the bus signals, the possibility of clock skew exists between various modules in a system due to routing or the use of multiple clock lines. It is the responsibility of the system to handle any such clock skew problems that could occur.

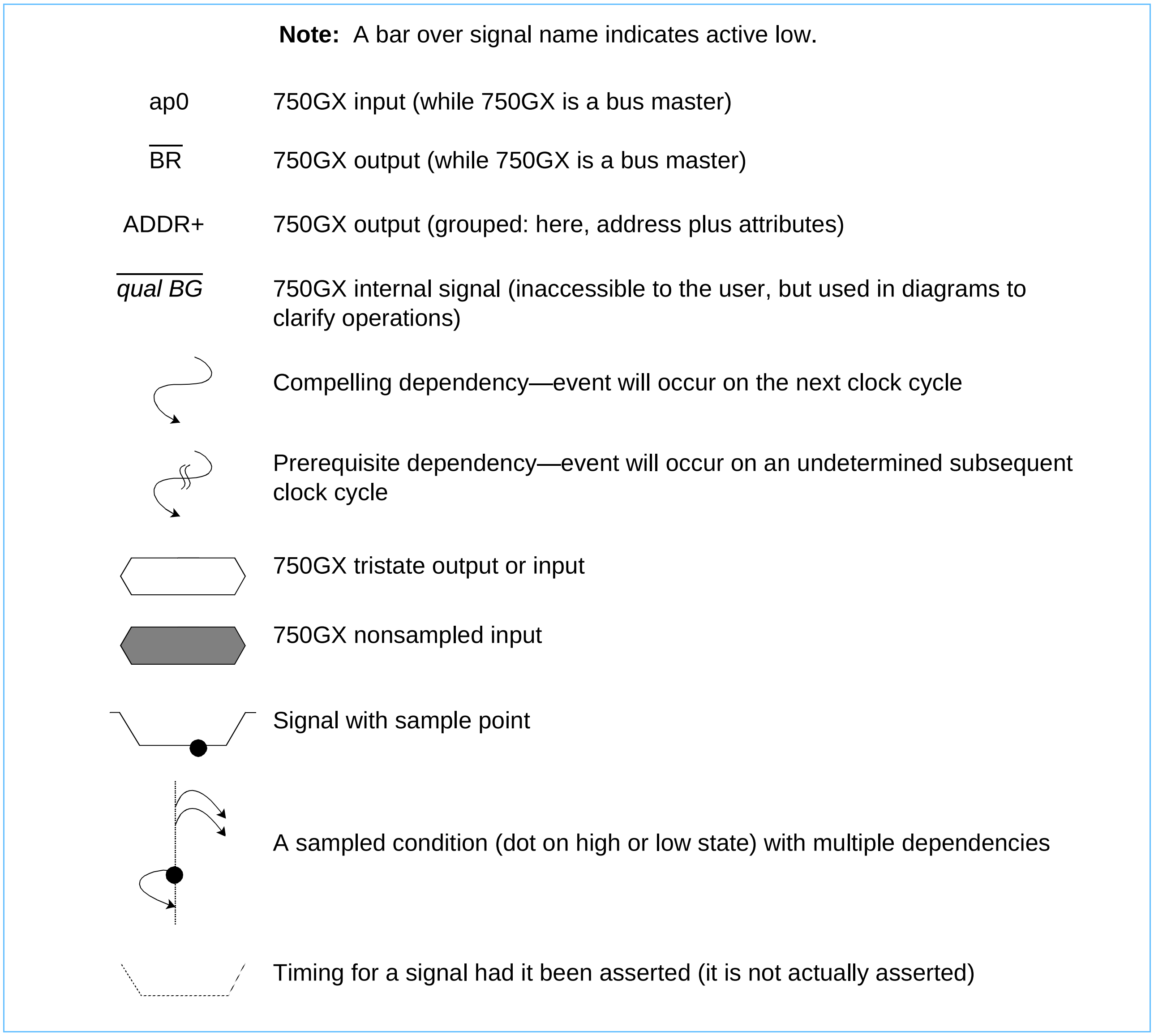
### 8.1.4 Optional 32-Bit Data Bus Mode

The 750GX supports an optional 32-bit data bus mode. 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. The number of data beats required for a data tenure in the 32-bit data bus mode is one, two, or eight beats depending on the size of the program transaction and the cache mode for the address. For additional information about 32-bit data bus mode, see *Section 8.6.1, 32-Bit Data Bus Mode,* on page 316.”

### 8.1.5 Direct-Store Accesses

The 750GX does not support the extended transfer protocol for accesses to the direct-store storage space. The transfer protocol used for any given access is selected by the T bit in the MMU Segment Registers. If the T bit is set, the memory access is a direct-store access. An attempt to access instructions or data in a direct-store segment will result in the 750GX taking an instruction storage interrupt (ISI) or data-storage interrupt (DSI) exception.

*Figure 8-2. Timing Diagram Legend*

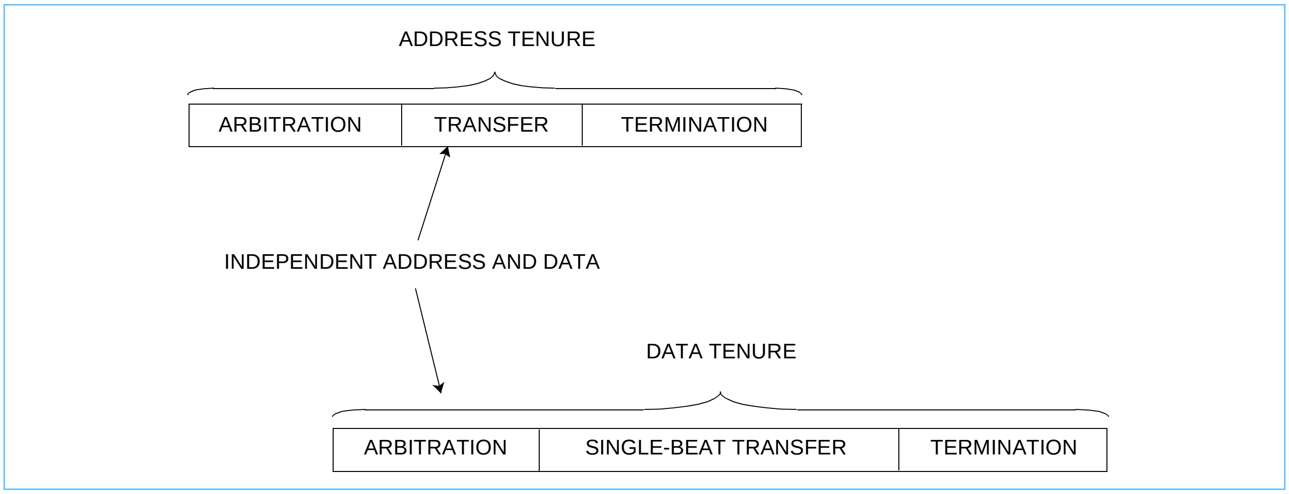

**Note:** A bar over signal name indicates active low.
ap0
750GX input (while 750GX is a bus master)
$\overline{\text{BR}}$
750GX output (while 750GX is a bus master)
ADDR+
750GX output (grouped: here, address plus attributes)
$\overline{\textit{qual BG}}$
750GX internal signal (inaccessible to the user, but used in diagrams to clarify operations)
Compelling dependency—event will occur on the next clock cycle
Prerequisite dependency—event will occur on an undetermined subsequent clock cycle
750GX tristate output or input
750GX nonsampled input
Signal with sample point
A sampled condition (dot on high or low state) with multiple dependencies
Timing for a signal had it been asserted (it is not actually asserted)

## 8.2 Memory-Access Protocol

Memory accesses are divided into address and data tenures. Each tenure has three phases—bus arbitration, transfer, and termination. The 750GX also supports address-only transactions. Note that address and data tenures can overlap, as shown in *Figure 8-3*.

*Figure 8-3* shows that the address and data tenures are distinct from one another and that both consist of three phases—arbitration, transfer, and termination. Address and data tenures are independent (indicated in *Figure 8-3* by the fact that the data tenure begins before the address tenure ends), which allows split-bus transactions to be implemented at the system level in multiprocessor systems. The figure also shows a data transfer that consists of a single-beat transfer of as many as 64 bits. Four-beat burst transfers of 32-byte cache lines require data-transfer termination signals for each beat of data.

*Figure 8-3. Overlapping Tenures on the* 750GX *Bus for a Single-Beat Transfer*



The basic functions of the address and data tenures are as follows.

*Address tenure*:

Arbitration — During arbitration, address-bus arbitration signals are used to gain mastership of the address bus.

Transfer — After the 750GX is the address-bus master, it transfers the address on the address bus. The address signals and the transfer attribute signals control the address transfer. The address parity and address-parity error signals ensure the integrity of the address transfer.

Termination — After the address transfer, the system signals that the address tenure is complete or that it must be repeated.

*Data tenure:*

Arbitration To begin the data tenure, the 750GX arbitrates for mastership of the data bus.

Transfer After the 750GX is the data-bus master, it samples the data bus for read operations or drives the data bus for write operations. The data parity and data-parity error signals ensure the integrity of the data transfer.

Termination Data termination signals are required after each data beat in a data transfer. Note that in a single-beat transaction, the data termination signals also indicate the end of the tenure. However, in burst accesses, the data termination signals apply to individual beats and indicate the end of the tenure only after the final data beat.

The 750GX generates an address-only bus transfer during the execution of the **dcbz** instruction (and for the **dcbi**, **dcbf**, **dcbst**, **sync**, and **eieio** instructions, if HID0[ABE] is enabled), which uses only the address bus with no data transfer involved. Additionally, the 750GX retry capability provides an efficient snooping protocol for systems with multiple memory systems (including caches) that must remain coherent.

### 8.2.1 Arbitration Signals

Arbitration for both address-bus and data-bus mastership is performed by a central, external arbiter and, minimally, by the arbitration signals shown in *Section 7.2.1, Address-Bus Arbitration Signals,* on page 251. Most arbiter implementations require additional signals to coordinate bus master/slave/snooping activities. Note that address-bus busy ($\overline{ABB}$) and data bus busy ($\overline{DBB}$) are bidirectional signals. These signals are inputs unless the 750GX has mastership of one or both of the respective buses. They must be connected high through pull-up resistors so that they remain negated when no devices have control of the buses.

The following list describes the address arbitration signals:

$\overline{BR}$ (bus request) Assertion indicates that the 750GX is requesting mastership of the address bus.

$\overline{BG}$ (bus grant) Assertion indicates that the 750GX might, with the proper qualification, assume mastership of the address bus. A qualified bus grant occurs when $\overline{BG}$ is asserted and $\overline{ABB}$ and address retry ($\overline{ARTRY}$) are negated.

If the 750GX is parked, $\overline{BR}$ need not be asserted for the qualified bus grant.

$\overline{ABB}$ (address bus busy) Assertion by the 750GX indicates that the 750GX is the address-bus master.

The following list describes the data arbitration signals:

$\overline{DBG}$ (data-bus grant) Indicates that the 750GX might, with the proper qualification, assume mastership of the data bus. A qualified data-bus grant occurs when $\overline{DBG}$ is asserted while $\overline{DBB}$, date retry ($\overline{DRTRY}$), and $\overline{ARTRY}$ are negated.

The $\overline{ARTRY}$ signal is driven from the bus and is only for the address-bus tenure associated with the current data-bus tenure (that is, not from another address tenure).

$\overline{\text{DBWO}}$ (data-bus write-only)

Assertion indicates that the 750GX might perform the data-bus tenure for an outstanding write address even if a read address is pipelined before the write address. If $\overline{\text{DBWO}}$ is asserted, the 750GX will assume data-bus mastership for a pending data-bus write operation. The 750GX will take the data bus for a pending read operation if this input is asserted along with $\overline{\text{DBG}}$ and no write is pending. Care must be taken with $\overline{\text{DBWO}}$ to ensure the desired write is queued (for example, a cache-line snoop push-out operation).

$\overline{\text{DBB}}$ (data bus busy)

Assertion by the 750GX indicates that the 750GX is the data-bus master. The 750GX always assumes data-bus mastership if it needs the data bus and is given a qualified data-bus grant (see $\overline{\text{DBG}}$).

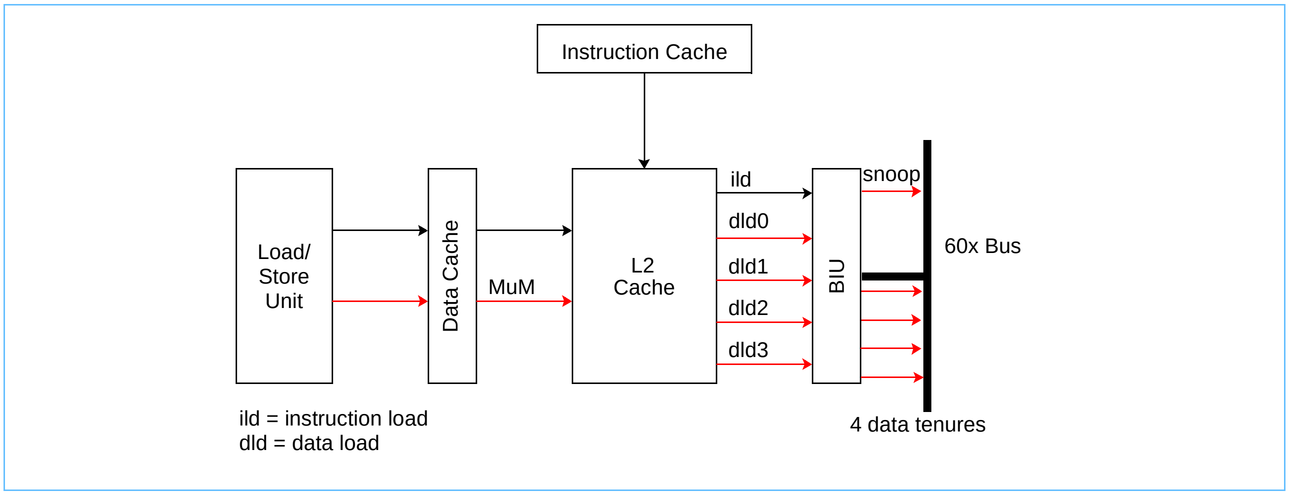
For more detailed information on the arbitration signals, see *Section 7.2.1, Address-Bus Arbitration Signals,* on page 251, and *Section 7.2.6, Data-Bus Arbitration Signals,* on page 264.

### 8.2.2 Miss-under-Miss

To improve processor performance, a feature called miss-under-miss (MuM) has been added which makes better use of the address pipelining function of the 60x bus and memory subsystem. It does this by looking deeper into the L/S unit and starting the process of fetching pending load misses in a pipelined fashion on the bus. Past versions of the 750 family supported limited pipelining where cache-inhibited stores and castouts could be pipelined with instruction and data reloads. Reloads, however, consumed a majority of the bus cycles and were not pipelined into other reloads, stalling the processor for instructions or data. Enabling MuM now allows four reloads or cache-inhibited loads to be pipelined in a continuous fashion on the 60x bus, with the BIU keeping track of requests, addreesses, and pipelining.

**Note:** The MuM only works because the the 60x bus is an in-order transfer. If it was a transaction type bus (out of order with tags), then the L/S and Dcache would need the extra queues as well.

*Figure 8-4. Cache Diagram for Miss-under-Miss Feature*



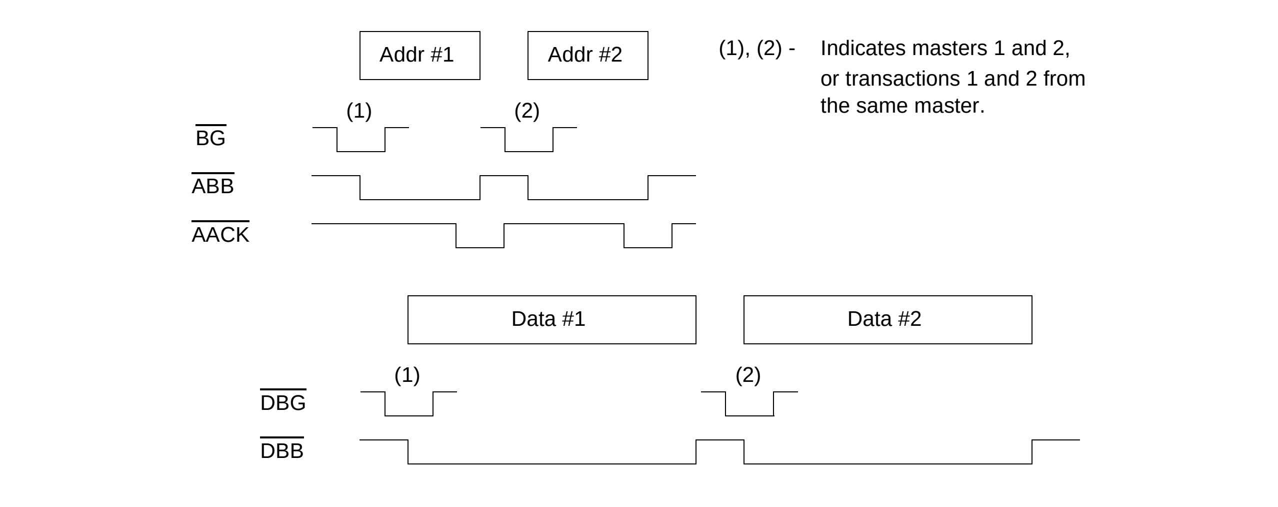
The data cache allows hits under one miss, but stalls for a second miss until the first miss is reloaded. The MuM feature enables a second request queue to the L2 cache for handling up to four misses. If there is a hit in the L2 cache for a data-cache miss-under-miss, then the L2 data is held until needed for allocation in the data cache. If there is a miss in the L2 cache, then the request is passed on to the bus interface unit (BIU) via three additional L2-to-BIU reload-request queues. Data returned from the bus is loaded into the data-cache reload buffer, one of the L2 reload buffers, and the critical word is forwarded to the load/store unit.

A dedicated snoop copyback queue has been added, which enables a fifth transaction to pipeline on the bus. It supports enveloped write transactions with the assertion of $\overline{\text{DBWO}}$. All snoop copybacks are issued from this queue.

A maximum of four reloads can be in progress through the L2 cache. The instruction cache will only request one reload at a time, and the data cache can request up to four. There can be a maximum of one instruction cache and three data cache reloads, or four data cache reloads.

An example of 1-level address pipelining is shown in *Figure 8-5* on page 287. Note that to support address pipelining, the memory system must no longer require the first address on the bus in order to complete the first data tenure, and possibly can also queue the second address to maximize the parallelism on the bus.

*Figure 8-5. First Level Address Pipelining*



### 8.2.2.1 Miss-under-Miss and System Performance

The MuM feature allows loads and stores that miss in the L1 cache to continue to the L2 cache, even though the L1 cache is busy reloading a prior miss. Hence the name, miss-under-miss (MuM). If MuM requests also miss in the L2 cache, they will proceed to the 60x bus in a pipelined fashion. A performance benefit is realized when pipelining on the 60x bus because the penalty for large memory latency only occurs with the first memory access. The greatest performance advantage is achieved if MuM requests can be sustained for as long as possible. Load/store instruction sequences affect how much benefit the MuM feature will produce.

The best sequence is a series of load instructions that reference a different cache-line index (EA[20:26]). Blocks of memory can be efficiently loaded into the data cache with tight loops that increment the address by x'20', as in the following example.

The L/S to data cache has two lines to indicate the normal request path and the MuM request path. MuM can serially request up to three more loads (hold,Eib0, and Eib1)  but the address queues are really in the BIU which can hold up to 4 loads.  MuM will be throttled by other events such as a full 3 entry Store queue in the L/S.

The BIU has both AR buffers and a 4-deep reload-request queue. So, the BIU operation for the MuM support is not dependent on the LSU queue, as it has enough buffers and queue depth to manage the outstanding transactions. The LSU has no additional queues for MuM. MuM just uses what is already there. If there are four data cache reload requests, the data cache does a lookup, reports a miss, and passes the request on to the L2/BIU, and it does not store any information.

```
//////////////////////////////////////////////////////////////////////////////////////////
example:1
            addis   r15,0,0x4200        # Base Data Address in r15
            addi    r16,0,0x0020        # Loop Count is 32
            mtctr   r16
lptop:      lwz     r20,0x0000(r15)
            lwz     r21,0x0020(r15)
            lwz     r22,0x0040(r15)
            lwz     r23,0x0060(r15)
            addi    r15,r15,0x0080      # Modify Base Data Address
            bdnz    lptop
            b       finish
//////////////////////////////////////////////////////////////////////////////////////////
```

There are several conditions, listed below, that can stall, limit, or prevent MuM so that the performance advantage on the bus will not be realized.

1. Sequential cacheable loads to the same index.

   Sequential cacheable loads or stores that reference the same L1 cache index will not be pipelined. The index bits are EA(20:26). A load to the same cache-line index as an outstanding load miss will prohibit all further MuM, even for successive loads to a different cache line, until the outstanding load miss is complete. The MuM feature does not look deeper into the load/store request queue for other loads that do not reference the same cache line once an index conflict exists.

   Cache-inhibited loads do not have this restriction.

2. TLB miss resulting in a table walk.

   When address translation is enabled using paging, a TLB miss occurs for load or store instructions that reference a page of memory not described in the TLB. A hardware table walk is then started, which reads page table entries (PTEs) from the caches or the bus. During this process, MuM will be prohibited until the table walk is complete.

3. Load request for a graphics instruction, such as External Control In Word Indexed (**eciwx**).

   Graphics instructions, such as **eciwx**, halt MuM, but once the **eciwx** is active on the bus, other qualified loads can initiate an MuM. An **eciwx** load will not be pipelined into other loads, but other loads can pipeline into it.

4. There is a load to guarded memory.

   Guarded loads are not allowed to be pipelined into other loads, and other loads are not pipelined into it.

   **Note:** Real mode, the default if address translation is not enabled, defines the write-through, caching-inhibited, memory coherency, guarded (WIMG) bits to b'0011'. The guarded attribute is set, and, therefore, MuM will not occur. Address translation must be enabled, and it must set the guarded bit (of WIMG) to zero for MuM.

5. Load multiple and load string instructions limit MuM.

Load multiple and load string instructions allow one MuM (two outstanding miss requests) to pipeline on the 60x bus.

6. A load is aliased to a store in the store queue, which means it references a byte to the same index and word. Loads are normally allowed to bypass stores in the 3-deep store queue. However, a load that aliases a store must allow the store to proceed ahead of it (in program order). The aliased store can start an MuM, but load MuM will wait until the alias condition is complete.
7. There is a cache inhibit (CI), **eieio**, or **sync** instruction in the store queue.

   Loads can bypass stores in the store queue, providing the store queue does not contain a CI, **eieio**, or **sync** instruction. MuM is prohibited while these instructions are executing.
8. Store queue in LSU is full (three entries).

   Once the store queue is full, and another store is dispatched, then the stores must be allowed to empty. In this state, MuM will drop down to two outstanding misses.
9. Exception, DSI, or alignment error.

   Any load or store instruction resulting in an exception, DSI, or alignment error will not be serviced for an MuM.
10. The **lwarx**, Data Cache Block Touch (**dcbt**), Data Cache Block Touch for Store (**dcbtst**), **dcbst**, **dcbf**, **dcbz**, **dcbi**, TLB Invalidate Entry (**tlbie**), and **eieio** instructions stall MuM requests.

    These instructions represent special cache and synchronizing mechanisms that will prevent MuM requests from starting until they have completed.
11. Cacheable loads will not MuM into cache-inhibited loads, and vice versa.

    Mixing cacheable loads with CI loads in the instruction stream prevents MuM requests from executing. Cacheable loads pipeline into other cacheable loads, but not into cache-inhibited loads. The two types of loads allow MuM requests only to their respective type.
12. Store misses pipeline to a maximum depth of two outstanding misses as shown below.

    ```
    Load word A miss
    Store word B    MuM
    .... no further load or store MuM
    Store word A miss
    Store word B MuM
    .... no further load or store MuM
    ```
13. Load miss pipeline in BIU to a maximum depth of four outstanding misses (same as the reload-request queue).

    If there are two outstanding requests, two more MuM requests can be issued if the above conditions are not true and the following conditions are true:

    a. The transaction is a load.

    b. The load transaction preceding a new MuM request is aligned on a cache-line boundary.

    c. There are no outstanding dependencies. For MuM, all operands for the address calculation must be valid.

    d. No preceding MuM request is an L2 cache hit. Once it is determined that an MuM request is an L2 hit, then no more MuM requests will proceed.

    e. The limit for reloads (four) has not been reached.
14. In little-endian mode, MuM is constrained to a maximum depth of two outstanding misses.

#### *8.2.2.2 Speculative Loads and Conditional Branches*

Loads that are dispatched before a preceding conditional branch is resolved are speculative. Mispredicted branches cause the speculative loads to be canceled. Normally, the cancellation is confined to the load/store unit, and no additional cycles are wasted. However, this is not the case when MuM is enabled. The speculative loads might be MuM requests that have started on the 60x bus. All outstanding MuM requests must complete, since there is no way to cancel them once they are started on the 60x bus. The load/store unit is now stalled until all outstanding loads have completed. The data cache is not reloaded for any MuM request that is canceled. However, the MuM reload is loaded into the L2 cache if enabled.

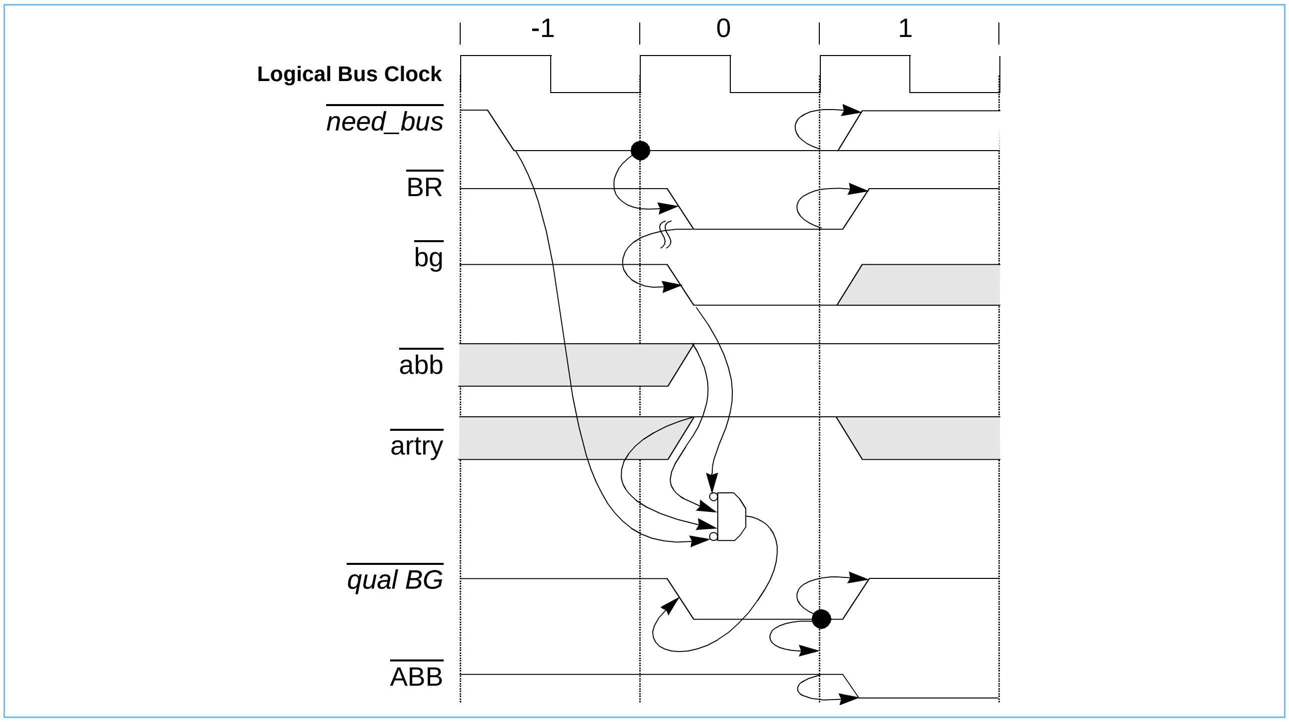
## 8.3 Address-Bus Tenure

This section describes the three phases of the address tenure—address-bus arbitration, address transfer, and address termination.

### 8.3.1 Address-Bus Arbitration

When the 750GX needs access to the external bus and it is not parked (BG is negated), it asserts the bus request (BR) signal until it is granted mastership of the bus and the bus is available (see *Figure 8-6*). The external arbiter must grant master-elect status to the potential master by asserting the bus grant (BG) signal. The 750GX requesting the bus determines that the bus is available when the ABB input is negated. When the address bus is not busy (ABB input is negated), BG is asserted and the address retry (ARTRY) input is negated. This is referred to as a qualified bus grant. The potential master assumes address-bus mastership by asserting ABB when it receives a qualified bus grant.

*Figure 8-6. Address-Bus Arbitration*



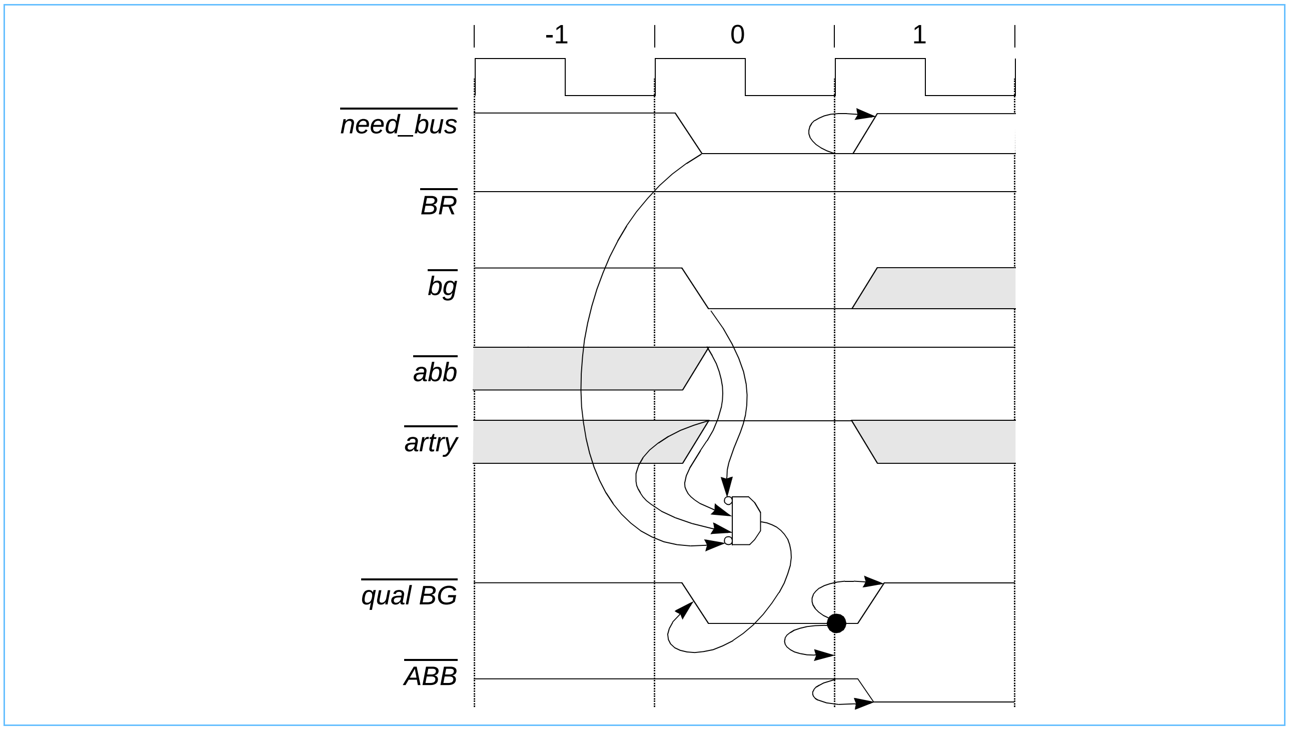
External arbiters must allow only one device at a time to be the address-bus master. For implementations in which no other device can be a master, BG can be grounded (always asserted) to continually grant mastership of the address bus to the 750GX.

**Note:** Arbiter designs must ensure that no more than one address-bus master can be granted the bus at one time (that is, bus grants must be mutually exclusive).

If the 750GX asserts BR before the external arbiter asserts BG, the 750GX is considered to be unparked, as shown in *Figure 8-6*. *Figure 8-7* shows the parked case, where a qualified bus grant exists on the clock edge following a need_bus condition. Notice that the bus clock cycle required for arbitration is eliminated if the 750GX is parked, reducing overall memory latency for a transaction. The 750GX always negates ABB for at least one bus clock cycle after the address acknowledge (AACK) signal is asserted, even if it is parked and has another transaction pending.

Typically, bus parking is provided to the device that was the most recent bus master. However, system designers might choose other schemes, such as providing unrequested bus grants in situations where it is easy to correctly predict the next device requesting bus mastership.

*Figure 8-7. Address-Bus Arbitration Showing Bus Parking*



When the 750GX receives a qualified bus grant, it assumes address-bus mastership by asserting ABB and negating the BR output signal. Meanwhile, the 750GX drives the address for the requested access onto the address bus and asserts transfer start (TS) to indicate the start of a new transaction.

When designing external bus arbitration logic, note that the 750GX might assert BR without using the bus after it receives the qualified bus grant. For example, in a system using bus snooping, if the 750GX asserts BR to perform a replacement copy-back operation, another device can invalidate that line before the 750GX is granted mastership of the bus. Once the 750GX is granted the bus, it no longer needs to perform the copy-back operation. Therefore, the 750GX does not assert ABB and does not use the bus for the copy-back operation. Note that the 750GX asserts BR for at least one clock cycle in these instances.

System designers should note that it is possible to ignore the ABB signal, and regenerate the state of ABB locally within each device by monitoring the TS and AACK input signals. The 750GX allows this operation by using both the ABB input signal and a locally regenerated version of ABB to determine if a qualified bus grant state exists (both sources are internally ORed together). The ABB signal can only be ignored if ABB and TS are asserted simultaneously by all masters, or where arbitration (through assertion of BG) is properly managed in cases where the regenerated ABB might not properly track the ABB signal on the bus. If the 750GX’s ABB signal is ignored by the system, it must be connected to a pull-up resistor to ensure proper operation. Additionally, the 750GX will not qualify a bus grant during the cycle that TS is asserted on the bus by any master. Address-bus arbitration without the use of the ABB signal requires that every assertion of TS be acknowledged by an assertion of AACK while the processor is not in sleep mode.

### 8.3.2 Address Transfer

During the address transfer, the physical address and all attributes of the transaction are transferred from the bus master to the slave devices. Snooping logic can monitor the transfer to enforce cache coherency; see the discussion of snooping in *Section 8.3.3, Address Transfer Termination,* on page 300.

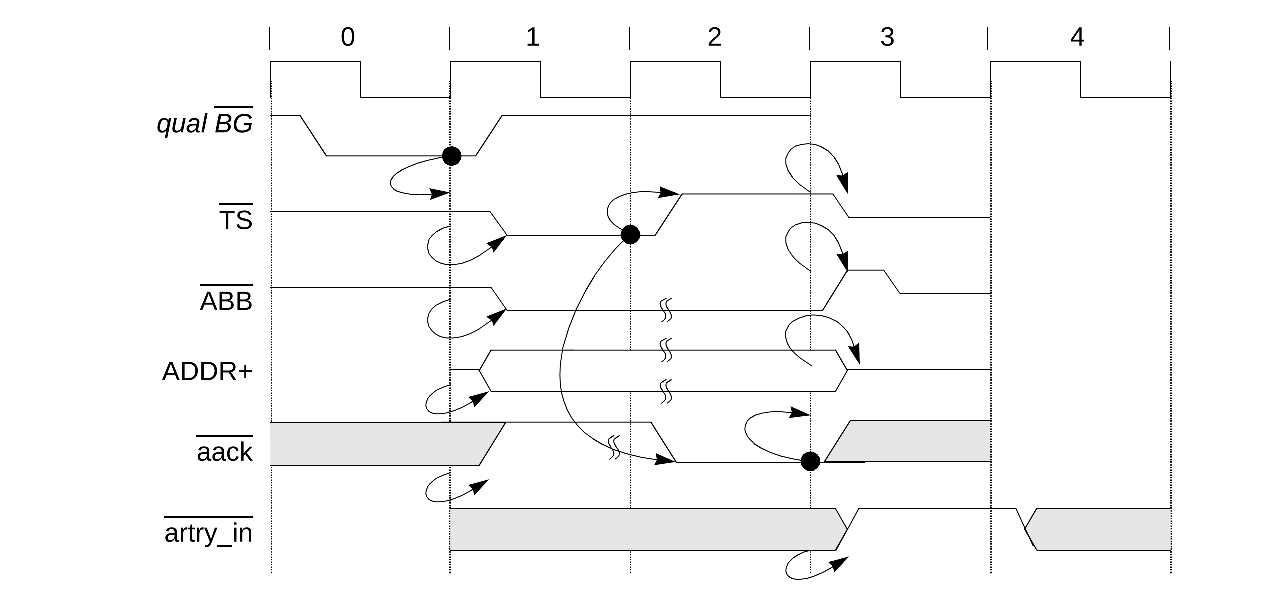
The signals used in the address transfer include the following signal groups:

- Address transfer start signal: transfer start (TS)
- Address transfer signals: address bus (A[0–31]), and address parity (AP[0–3])
- Address transfer attribute signals: transfer type (TT[0–4]), transfer size (TSIZ[0–2]), transfer burst (TBST), cache inhibit (CI), write-through (WT), and global (GBL)

*Figure 8-8* on page 293 shows that the timing for all of these signals, except TS, is identical. All of the address transfer and address transfer attribute signals are combined into the ADDR+ grouping in *Figure 8-8*. The TS signal indicates that the 750GX has begun an address transfer and that the address and transfer attributes are valid (within the context of a synchronous bus). The 750GX always asserts TS coincident with ABB. As an input, TS need not coincide with the assertion of ABB on the bus (that is, TS can be asserted with, or on, a subsequent clock cycle after ABB is asserted; the 750GX tracks this transaction correctly).

In *Figure 8-8*, the address transfer occurs during bus clock cycles 1 and 2 (arbitration occurs in bus clock cycle 0 and the address transfer is terminated in bus clock 3). In this diagram, the address-bus termination input, AACK, is asserted to the 750GX on the bus clock following assertion of TS (as shown by the dependency line). This is the minimum duration of the address transfer for the 750GX; the duration can be extended by delaying the assertion of AACK for one or more bus clocks.

*Figure 8-8. Address-Bus Transfer*



### *8.3.2.1 Address-Bus Parity*

The 750GX always generates 1 bit of correct odd-byte parity for each of the 4 bytes of address when a valid address is on the bus. The calculated values are placed on the AP[0–3] outputs when the 750GX is the address-bus master. If the 750GX is not the master and TS and GBL are asserted together (qualified condition for snooping memory operations), the calculated values are compared with the AP[0–3] inputs. If there is an error, and address-parity checking is enabled (HID0[EBA] set to 1), a machine-check exception is generated. An address-bus parity error causes a checkstop condition if MSR[ME] is cleared to 0. For more information about checkstop conditions, see *Chapter 4, Exceptions,* on page 151.

### *8.3.2.2 Address Transfer Attribute Signals*

The transfer attribute signals include several encoded signals such as the transfer type (TT[0–4]) signals, transfer burst (TBST) signal, transfer size (TSIZ[0–2]) signals, write-through (WT), and cache inhibit (CI). *Section 7.2.4, Address Transfer Attribute Signals,* on page 255 describes the encodings for the address transfer attribute signals.

#### *Transfer Type (TT[0–4]) Signals*

Snooping logic should fully decode the transfer type signals if the GBL signal is asserted. Slave devices can sometimes use the individual transfer type signals without fully decoding the group. For a complete description of the encoding for TT[0–4], see *Table 8-1* and *Table 8-2* on page 295.

#### *Transfer Size (TSIZ[0–2]) Signals*

The TSIZ[0–2] signals indicate the size of the requested data transfer as shown in *Table 8-1*. The TSIZ[0–2] signals can be used along with TBST and A[29–31] to determine which portion of the data bus contains valid data for a write transaction or which portion of the bus should contain valid data for a read transaction. Note that, for a burst transaction (as indicated by the assertion of TBST), TSIZ[0–2] are always set to 0b010. Therefore, if the TBST signal is asserted, the memory system should transfer a total of eight words (32 bytes), regardless of the TSIZ[0–2] encodings.

*Table 8-1. Transfer Size Signal Encodings*

| TBST | TSIZ0 | TSIZ1 | TSIZ2 | Transfer Size |
|---|---|---|---|---|
| Asserted | 0 | 1 | 0 | 8-word burst |
| Negated | 0 | 0 | 0 | 8 bytes |
| Negated | 0 | 0 | 1 | 1 byte |
| Negated | 0 | 1 | 0 | 2 bytes |
| Negated | 0 | 1 | 1 | 3 bytes |
| Negated | 1 | 0 | 0 | 4 bytes |
| Negated | 1 | 0 | 1 | 5 bytes (N/A) |
| Negated | 1 | 1 | 0 | 6 bytes (N/A) |
| Negated | 1 | 1 | 1 | 7 bytes (N/A) |

The basic coherency size of the bus is defined to be 32 bytes (corresponding to one cache line). Data transfers that cross an aligned, 32-byte boundary either must present a new address onto the bus at that boundary (for coherency consideration) or must operate as noncoherent data with respect to the 750GX. The 750GX never generates a bus transaction with a transfer size of 5 bytes, 6 bytes, or 7 bytes.

#### *Write-Through ($\overline{\text{WT}}$) Signal*

The 750GX provides the $\overline{\text{WT}}$ signal to indicate a write-through operation as determined by the WIM bit settings during address translation by the MMU. The $\overline{\text{WT}}$ signal is also asserted for burst writes due to the execution of the **dcbf** and **dcbst** instructions, and snoop push operations. The $\overline{\text{WT}}$ signal is deasserted for accesses caused by the execution of the External Control Out Word Indexed (**ecowx**) instruction. During read operations, the 750GX uses the $\overline{\text{WT}}$ signal to indicate whether the transaction is an instruction fetch ($\overline{\text{WT}}$ negated) or a data read operation ($\overline{\text{WT}}$ asserted).

#### *Cache Inhibit ($\overline{\text{CI}}$) Signal*

The 750GX indicates the caching-inhibited status of a transaction (determined by the setting of the WIM bits by the MMU) through the use of the $\overline{\text{CI}}$ signal. The $\overline{\text{CI}}$ signal is asserted even if the L1 caches are disabled or locked. This signal is also asserted for bus transactions caused by the execution of **eciwx** and **ecowx** instructions independent of the address translation.

### 8.3.2.3 Burst Ordering During Data Transfers

During burst data-transfer operations, 32 bytes of data (one cache line) are transferred to or from the cache in order. Burst write transfers are always performed zero double word first, but since burst reads are performed critical double word first, a burst read transfer might not start with the first double word of the cache line, and the cache-line fill might wrap around the end of the cache line.

*Table 8-2. Burst Ordering—64-Bit Bus*

| Data Transfer | Double-Word Starting Address: | | | |
|---|---|---|---|---|
| | A[27–28] = 00 | A[27–28] = 01 | A[27–28] = 10 | A[27–28] = 11 |
| First data beat | DW0 | DW1 | DW2 | DW3 |
| Second data beat | DW1 | DW2 | DW3 | DW0 |
| Third data beat | DW2 | DW3 | DW0 | DW1 |
| Fourth data beat | DW3 | DW0 | DW1 | DW2 |
| **Note:** A[29–31] are always 0b000 for burst transfers by the 750GX. | | | | |

*Table 8-3. Burst Ordering—32-Bit Bus*

| Data Transfer | For Starting Address: | | | |
|---|---|---|---|---|
| | A[27–28] = 00 | A[27–28] = 01 | A[27–28] = 10 | A[27–28] = 11 |
| First data beat | DW0-U | DW1-U | DW2-U | DW3-U |
| Second data beat | DW0-L | DW1-L | DW2-L | DW3-L |
| Third data beat | DW1-U | DW2-U | DW3-U | DW0-U |
| Fourth data beat | DW1-L | DW2-L | DW3-L | DW0-L |
| Fifth data beat | DW2-U | DW3-U | DW0-U | DW1-U |
| Sixth data beat | DW2-L | DW3-L | DW0-L | DW1-L |
| Seventh data beat | DW3-U | DW0-U | DW1-U | DW2-U |
| Eighth data beat | DW3-L | DW0-L | DW1-L | DW2-L |

**Note:**
A[29–31] are always 0b000 for burst transfers by the 750GX.
"U" and "L" represent the upper and lower word of the double word respectively.

### 8.3.2.4 Effect of Alignment in Data Transfers

*Table 8-4* lists the aligned transfers that can occur on the 750GX bus. These are transfers in which the data is aligned to an address that is an integral multiple of the size of the data. For example, *Table 8-4* shows that 1-byte data is always aligned. However, for a 4-byte word to be aligned, it must be oriented on an address that is a multiple of four.

*Table 8-4. Aligned Data Transfers* (Page 1 of 2)

| Transfer Size | TSIZ0 | TSIZ1 | TSIZ2 | A[29–31] | Data-Bus Byte Lane(s) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Byte | 0 | 0 | 1 | 000 | x | — | — | — | — | — | — | — |
| | 0 | 0 | 1 | 001 | — | x | — | — | — | — | — | — |
| | 0 | 0 | 1 | 010 | — | — | x | — | — | — | — | — |
| | 0 | 0 | 1 | 011 | — | — | — | x | — | — | — | — |
| | 0 | 0 | 1 | 100 | — | — | — | — | x | — | — | — |
| | 0 | 0 | 1 | 101 | — | — | — | — | — | x | — | — |
| | 0 | 0 | 1 | 110 | — | — | — | — | — | — | x | — |
| | 0 | 0 | 1 | 111 | — | — | — | — | — | — | — | x |
| Half word | 0 | 1 | 0 | 000 | x | x | — | — | — | — | — | — |
| | 0 | 1 | 0 | 010 | — | — | x | x | — | — | — | — |
| | 0 | 1 | 0 | 100 | — | — | — | — | x | x | — | — |
| | 0 | 1 | 0 | 110 | — | — | — | — | — | — | x | x |

**Note:** The entries with an "x" indicate the byte portions of the requested operand that are read or written during a bus transaction. The entries with a "–" are not required and are ignored during read transactions, and they are driven with undefined data during all write transactions.

*Table 8-4. Aligned Data Transfers* (Page 2 of 2)

| Transfer Size | TSIZ0 | TSIZ1 | TSIZ2 | A[29–31] | Data-Bus Byte Lane(s) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Word | 1 | 0 | 0 | 000 | x | x | x | x | — | — | — | — |
| | 1 | 0 | 0 | 100 | — | — | — | — | x | x | x | x |
| Double word | 0 | 0 | 0 | 000 | x | x | x | x | x | x | x | x |

**Note:** The entries with an “x” indicate the byte portions of the requested operand that are read or written during a bus transaction. The entries with a “–” are not required and are ignored during read transactions, and they are driven with undefined data during all write transactions.

The 750GX supports misaligned memory operations, although their use can substantially degrade performance. Misaligned memory transfers address memory that is not aligned to the size of the data being transferred (such as, a word read of an odd byte address). Although most of these operations hit in the primary cache (or generate burst memory operations if they miss), the 750GX interface supports misaligned transfers within a word (32-bit aligned) boundary, as shown in *Table 8-5* on page 298.

**Note:** The 4-byte transfer in *Table 8-5* is only one example of misalignment. As long as the attempted transfer does not cross a word boundary, the 750GX can transfer the data on the misaligned address (for example, a half-word read from an odd byte-aligned address). An attempt to address data that crosses a word boundary requires two bus transfers to access the data.

Due to the performance degradations associated with misaligned memory operations, they are best avoided. In addition to the double-word straddle boundary condition, the address-translation logic can generate substantial exception overhead when the load/store multiple and load/store string instructions access misaligned data. It is strongly recommended that software attempt to align data where possible.

*Table 8-5. Misaligned Data Transfers (4-Byte Examples)*

| Transfer Size<br>(Four Bytes) | TSIZ[0–2] | A[29–31] | Data-Bus Byte Lanes | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Aligned | 1 0 0 | 0 0 0 | A | A | A | A | — | — | — | — |
| Misaligned—first access | 0 1 1 | 0 0 1 | | A | A | A | — | — | — | — |
| second access | 0 0 1 | 1 0 0 | — | — | — | — | A | — | — | — |
| Misaligned—first access | 0 1 0 | 0 1 0 | — | — | A | A | — | — | — | — |
| second access | 0 1 1 | 1 0 0 | — | — | — | — | A | A | — | — |
| Misaligned—first access | 0 0 1 | 0 1 1 | — | — | — | A | — | — | — | — |
| second access | 0 1 1 | 1 0 0 | — | — | — | — | A | A | A | — |
| Aligned | 1 0 0 | 1 0 0 | — | — | — | — | A | A | A | A |
| Misaligned—first access | 0 1 1 | 1 0 1 | — | — | — | — | — | A | A | A |
| second access | 0 0 1 | 0 0 0 | A | — | — | — | — | — | — | — |
| Misaligned—first access | 0 1 0 | 1 1 0 | — | — | — | — | — | — | A | A |
| second access | 0 1 0 | 0 0 0 | A | A | — | — | — | — | — | — |
| Misaligned—first access | 0 0 1 | 1 1 1 | — | — | — | — | — | — | — | A |
| second access | 0 1 1 | 0 0 0 | A | A | A | — | — | — | — | — |

**Note:**
A: Byte lane used
—: Byte lane not used

### *Effect of Alignment in Data Transfers (32-Bit Bus)*

The aligned data-transfer cases for 32-bit data-bus mode are shown in *Table 8-6* on page 298. All of the transfers require a single data beat (if caching-inhibited or write-through) except for double-word cases which require two data beats. The double-word case is only generated by the 750GX for load or store double operations to or from the floating-point General Purpose Registers (GPRs). All caching-inhibited instruction fetches are performed as word operations.

*Table 8-6. Aligned Data Transfers (32-Bit Bus Mode)* (Page 1 of 2)

| Transfer Size | TSIZ0 | TSIZ1 | TSIZ2 | A[29–31] | Data-Bus Byte Lanes | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Byte | 0 | 0 | 1 | 000 | A | — | — | — | x | x | x | x |
| | 0 | 0 | 1 | 001 | — | A | x | — | x | x | x | x |
| | 0 | 0 | 1 | 010 | — | — | A | — | x | x | x | x |
| | 0 | 0 | 1 | 011 | — | — | — | A | x | x | x | x |
| | 0 | 0 | 1 | 100 | A | — | — | — | x | x | x | x |
| | 0 | 0 | 1 | 101 | — | A | — | — | x | x | x | x |
| | 0 | 0 | 1 | 110 | — | — | A | — | x | x | x | x |
| | 0 | 0 | 1 | 111 | — | — | — | A | x | x | x | x |

*Table 8-6. Aligned Data Transfers (32-Bit Bus Mode)* (Page 2 of 2)

| Transfer Size | TSIZ0 | TSIZ1 | TSIZ2 | A[29–31] | Data-Bus Byte Lanes | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Half word | 0 | 1 | 0 | 000 | A | A | — | — | x | x | x | x |
| | 0 | 1 | 0 | 010 | — | — | A | A | x | x | x | x |
| | 0 | 1 | 0 | 100 | A | A | — | — | x | x | x | x |
| | 0 | 1 | 0 | 110 | — | — | A | A | x | x | x | x |
| Word | 1 | 0 | 0 | 000 | A | A | A | A | x | x | x | x |
| | 1 | 0 | 0 | 100 | A | A | A | A | x | x | x | x |
| Double word | 0 | 0 | 0 | 000 | A | A | A | A | x | x | x | x |
| Second beat | 0 | 0 | 0 | 000 | A | A | A | A | x | x | x | x |

**Note:**
A: Byte lane used
—: Byte lane not used
x: Byte lane not used in 32-bit bus mode

Misaligned data transfers when the 750GX is configured with a 32-bit data bus operate in the same way as when configured with a 64-bit data bus, with the exception that only the DH[0–31] data bus is used. See *Table 8-7* on page 299 for an example of a 4-byte misaligned transfer starting at each possible byte address within a double word.

*Table 8-7. Misaligned 32-Bit Data-Bus Transfer (4-Byte Examples)*

| Transfer Size<br>(Four Bytes) | TSIZ[0–2] | A[29–31] | Data-Bus Byte Lanes | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Aligned | 1 0 0 | 0 0 0 | A | A | A | A | x | x | x | x |
| Misaligned—first access | 0 1 1 | 0 0 1 | | A | A | A | x | x | x | x |
| second access | 0 0 1 | 1 0 0 | A | — | — | — | x | x | x | x |
| Misaligned—first access | 0 1 0 | 0 1 0 | — | — | A | A | x | x | x | x |
| second access | 0 1 0 | 1 0 0 | A | A | — | x | x | x | x | x |
| Misaligned—first access | 0 0 1 | 0 1 1 | — | — | — | A | x | x | x | x |
| second access | 0 1 1 | 1 0 0 | A | A | A | — | x | x | x | x |
| Aligned | 1 0 0 | 1 0 0 | A | A | A | A | x | x | x | x |
| Misaligned—first access | 0 1 1 | 1 0 1 | — | A | A | A | x | x | x | x |
| second access | 0 0 1 | 0 0 0 | A | — | — | — | x | x | x | x |
| Misaligned—first access | 0 1 0 | 1 1 0 | — | — | A | A | x | x | x | x |
| second access | 0 1 0 | 0 0 0 | A | A | — | — | x | x | x | x |
| Misaligned—first access | 0 0 1 | 1 1 1 | — | — | — | A | x | x | x | x |
| second access | 0 1 1 | 0 0 0 | A | A | A | — | x | x | x | x |

**Note:**
A: Byte lane used
—: Byte lane not used
x: Byte lane not used in 32-bit bus mode

#### *8.3.2.5 Alignment of External Control Instructions*

The size of the data transfer associated with the **eciwx** and **ecowx** instructions is always 4 bytes. If the **eciwx** or **ecowx** instruction is misaligned and crosses any word boundary, the 750GX will generate an alignment exception.

### 8.3.3 Address Transfer Termination

The address tenure of a bus operation is terminated when completed with the assertion of AACK, or retried with the assertion of ARTRY. The 750GX does not terminate the address transfer until the AACK input is asserted. Therefore, the system can extend the address transfer phase by delaying the assertion of AACK to the 750GX. The assertion of AACK can be as early as the bus clock cycle following TS (see *Figure 8-9* on page 301), which allows a minimum address tenure of two bus cycles. As shown in *Figure 8-9*, these signals are asserted for one bus clock cycle, tristated for half of the next bus clock cycle, driven high until the following bus cycle, and finally tristated. Note that AACK must be asserted for only one bus clock cycle.

The address transfer can be terminated with the requirement to retry if ARTRY is asserted anytime during the address tenure and through the cycle following AACK. The assertion causes the entire transaction (address and data tenure) to be rerun. As a snooping device, the 750GX asserts ARTRY for a snooped transaction that hits modified data in the data cache that must be written back to memory, or if the snooped transaction could not be serviced. As a bus master, the 750GX responds to an assertion of ARTRY by canceling the bus transaction and rerequesting the bus. Note that after recognizing an assertion of ARTRY and canceling the transaction in progress, the 750GX is not guaranteed to run the same transaction the next time it is granted the bus due to internal reordering of load-and-store operations.

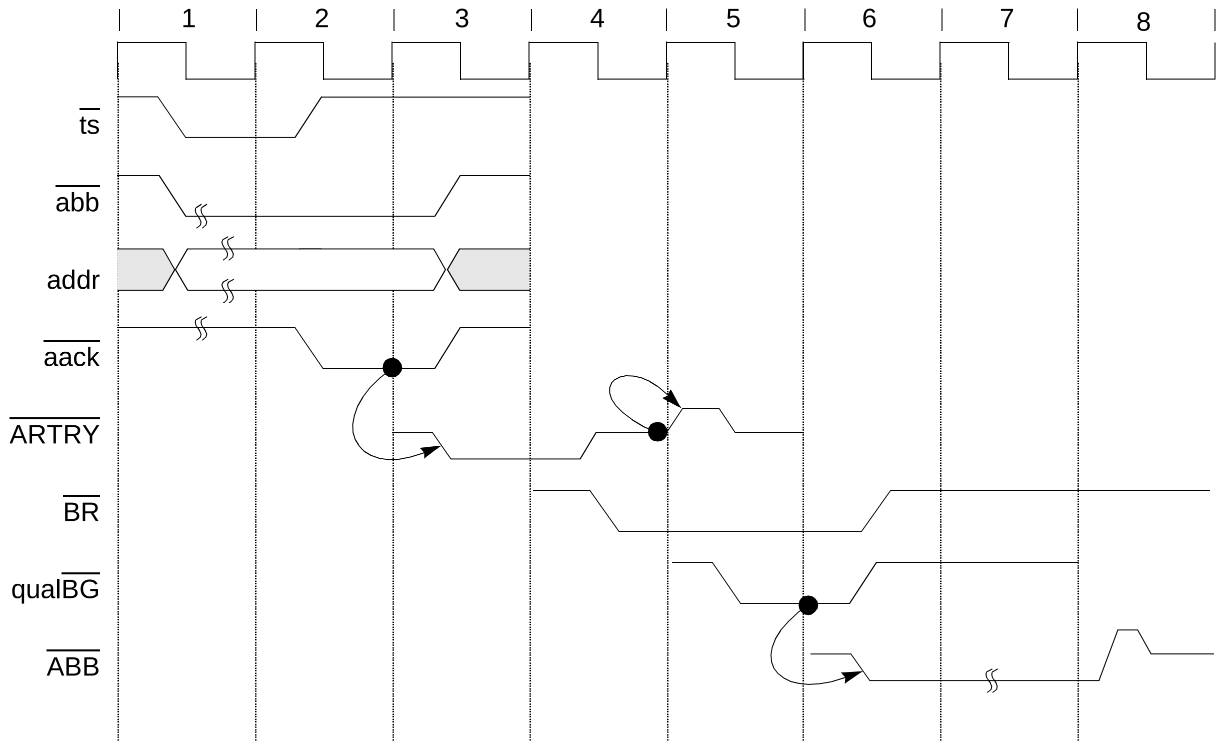
If an address retry is required, the ARTRY response will be asserted by a bus snooping device as early as the second cycle after the assertion of TS. Once asserted, ARTRY must remain asserted through the cycle after the assertion of AACK. The assertion of ARTRY during the cycle after the assertion of AACK is referred to as a qualified ARTRY. An earlier assertion of ARTRY during the address tenure is referred to as an early ARTRY.

As a bus master, the 750GX recognizes either an early or qualified ARTRY and prevents the data tenure associated with the retried address tenure. If the data tenure has already begun, the 750GX cancels and terminates the data tenure immediately even if the burst data has been received. If the assertion of ARTRY is received up to or on the bus cycle following the first (or only) assertion of TA for the data tenure, the 750GX ignores the first data beat, and if it is a load operation, does not forward data internally to the cache and execution units. If ARTRY is asserted after the first (or only) assertion of TA, improper operation of the bus interface can result.

During the clock of a qualified ARTRY, the 750GX also determines if it should negate BR and ignore BG on the following cycle. On the following cycle, only the snooping master that asserted ARTRY and needs to perform a snoop copy-back operation is allowed to assert BR. This guarantees the snooping master an opportunity to request and be granted the bus before the just-retried master can restart its transaction. Note that a nonclocked bus arbiter might detect the assertion of address-bus request by the bus master that asserted ARTRY, and return a qualified bus grant one cycle earlier than shown in *Figure 8-9*.

Note that if the 750GX asserts ARTRY due to a snoop operation, and asserts BR in the bus cycle following ARTRY in order to perform a snoop push to memory, the 750GX might not be able to accept a BG until several bus cycles later. (The delay in responding to the assertion of BG only occurs during snoop pushes from the L2 cache.) The bus arbiter should keep BG asserted until it detects BR negated or TS asserted from the 750GX, which indicates that the snoop copy-back has begun. The system should ensure that no other address tenures occur until the current snoop push from the 750GX is completed. Snoop push delays can also be avoided by operating the L2 cache in write-through mode so no snoop pushes are required by the L2 cache.

*Figure 8-9. Snooped Address Cycle with $\overline{\text{ARTRY}}$*



## 8.4 Data-Bus Tenure

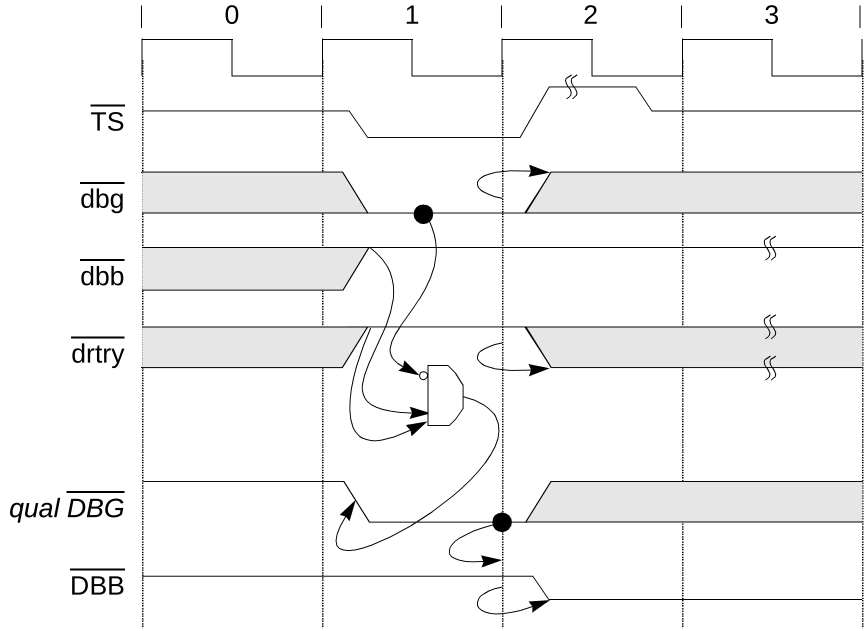
This section describes the data-bus arbitration, transfer, and termination phases defined by the 750GX memory-access protocol. The phases of the data tenure are identical to those of the address tenure, underscoring the symmetry in the control of the two buses.

### 8.4.1 Data-Bus Arbitration

Data-bus arbitration uses the data arbitration signal group—$\overline{\text{DBG}}$, $\overline{\text{DBWO}}$, and $\overline{\text{DBB}}$. Additionally, the combination of $\overline{\text{TS}}$ and TT[0–4] provides information about the data-bus request to external logic.

The $\overline{\text{TS}}$ signal is an implied data-bus request from the 750GX. The arbiter must qualify $\overline{\text{TS}}$ with the transfer type (TT) encodings to determine if the current address transfer is an address-only operation, which does not require a data-bus transfer (see *Figure 8-9*). If the data bus is needed, the arbiter grants data-bus mastership by asserting the $\overline{\text{DBG}}$ input to the 750GX. As with the address-bus arbitration phase, the 750GX must qualify the $\overline{\text{DBG}}$ input with a number of input signals before assuming bus mastership, as shown in *Figure 8-10*.

*Figure 8-10. Data-Bus Arbitration*



A qualified data-bus grant can be expressed as the following:

> QDBG = $\overline{\text{DBG}}$ asserted while $\overline{\text{DBB}}$, $\overline{\text{DRTRY}}$, and $\overline{\text{ARTRY}}$ (associated with the data-bus operation) are negated.

When a data tenure overlaps with its associated address tenure, a qualified $\overline{\text{ARTRY}}$ assertion coincident with a data-bus grant signal does not result in data-bus mastership ($\overline{\text{DBB}}$ is not asserted). Otherwise, the 750GX always asserts $\overline{\text{DBB}}$ on the bus clock cycle after recognition of a qualified data-bus grant. Since the 750GX can pipeline transactions, there might be an outstanding data-bus transaction when a new address transaction is retried. In this case, the 750GX becomes the data-bus master to complete the outstanding transaction.

#### *8.4.1.1 Using the $\overline{\text{DBB}}$ Signal*

The $\overline{\text{DBB}}$ signal should be connected between masters if data tenure scheduling is left to the masters. Optionally, the memory system can control data tenure scheduling directly with $\overline{\text{DBG}}$. However, it is possible to ignore the $\overline{\text{DBB}}$ signal in the system if the $\overline{\text{DBB}}$ input is not used as the final data-bus allocation control between data-bus masters, and if the memory system can track the start and end of the data tenure. If $\overline{\text{DBB}}$ is not used to signal the end of a data tenure, $\overline{\text{DBG}}$ is only asserted to the next bus master the cycle before the cycle that the next bus master might actually begin its data tenure, rather than asserting it earlier (usually during another master’s data tenure) and allowing the negation of $\overline{\text{DBB}}$ to be the final gating signal for a qualified data-bus grant. Even if $\overline{\text{DBB}}$ is ignored in the system, the 750GX always recognizes its own assertion of $\overline{\text{DBB}}$, and requires one cycle after data tenure completion to negate its own $\overline{\text{DBB}}$ before recognizing a qualified data-bus grant for another data tenure. If $\overline{\text{DBB}}$ is ignored in the system, it must still be connected to a pull-up resistor on the 750GX to ensure proper operation.

### 8.4.2 Data-Bus Write-Only

As a result of address pipelining, the 750GX can have up to two data tenures queued to perform when it receives a qualified DBG. Generally, the data tenures should be performed in strict order (the same order as their address tenures were performed). The 750GX, however, also supports a limited out-of-order capability with the data-bus write-only (DBWO) input. When recognized on the clock of a qualified DBG, DBWO can direct the 750GX to perform the next pending data write tenure even if a pending read tenure would have normally been performed first. For more information on the operation of DBWO, see *Section 8.9, Using Data-Bus Write-Only,* on page 320.

If the 750GX has any data tenures to perform, it always accepts data-bus mastership to perform a data tenure when it recognizes a qualified DBG. If DBWO is asserted with a qualified DBG and no write tenure is queued to run, the 750GX still takes mastership of the data bus to perform the next pending read data tenure.

Generally, DBWO should only be used to allow a copy-back operation (burst write) to occur before a pending read operation. If DBWO is used for single-beat write operations, it can negate the effect of the **eieio** instruction by allowing a write operation to precede a program-scheduled read operation.

### 8.4.3 Data Transfer

The data-transfer signals include the data bus high (DH[0–31]), data bus low (DL[0–31]), and data bus parity (DP[0–7]) signals. For memory accesses, the DH and DL signals form a 64-bit data path for read and write operations.

The 750GX transfers data in either single-beat or 4-beat burst transfers. Single-beat operations can transfer from 1 to 8 bytes at a time and can be misaligned; see *Section 8.3.2.4, Effect of Alignment in Data Transfers,* on page 296. Burst operations always transfer eight words and are aligned on 8-word address boundaries. Burst transfers can achieve significantly higher bus throughput than single-beat operations.

The type of transaction initiated by the 750GX depends on whether the code or data is cacheable and, for store operations, whether the cache is in write-back or write-through mode, which software controls on either a page or block basis. Burst transfers support cacheable operations only. That is, memory structures must be marked as cacheable (and write-back for data store operations) in the respective page or block descriptor to take advantage of burst transfers.

The 750GX output TBST indicates to the system whether the current transaction is a single-beat or 4-beat transfer (except during **eciwx** and **ecowx** transactions, when it signals the state of bit 28 of the External Access Register (EAR[28]). A burst transfer has an assumed address order. For load or store operations that miss in the cache (and are marked as cacheable and, for stores, write-back in the MMU), the 750GX uses the double-word-aligned address associated with the critical code or data that initiated the transaction. This minimizes latency by allowing the critical code or data to be forwarded to the processor before the rest of the cache line is filled. For all other burst operations, however, the cache line is transferred beginning with the 8-word-aligned data.

### 8.4.4 Data-Transfer Termination

Four signals are used to terminate data-bus transactions—TA, DRTRY, transfer error acknowledge (TEA), and ARTRY. The TA signal indicates normal termination of data transactions. It must always be asserted on the bus cycle coincident with the data that it is qualifying. It can be withheld by the slave for any number of clocks until valid data is ready to be supplied or accepted. DRTRY indicates invalid read data in the previous bus clock cycle. DRTRY extends the current data beat and does not terminate it. If it is asserted after the last (or only) data beat, the 750GX negates DBB but still considers the data beat active and waits for another assertion of TA. DRTRY is ignored on write operations. TEA indicates a nonrecoverable bus error event. Upon receiving a final (or only) termination condition, the 750GX always negates DBB for one cycle.

If DRTRY is asserted by the memory system to extend the last (or only) data beat past the negation of DBB, the memory system should tristate the data bus on the clock after the final assertion of TA, even though it will negate DRTRY on that clock. This is to prevent a potential momentary data-bus conflict if a write access begins on the following cycle.

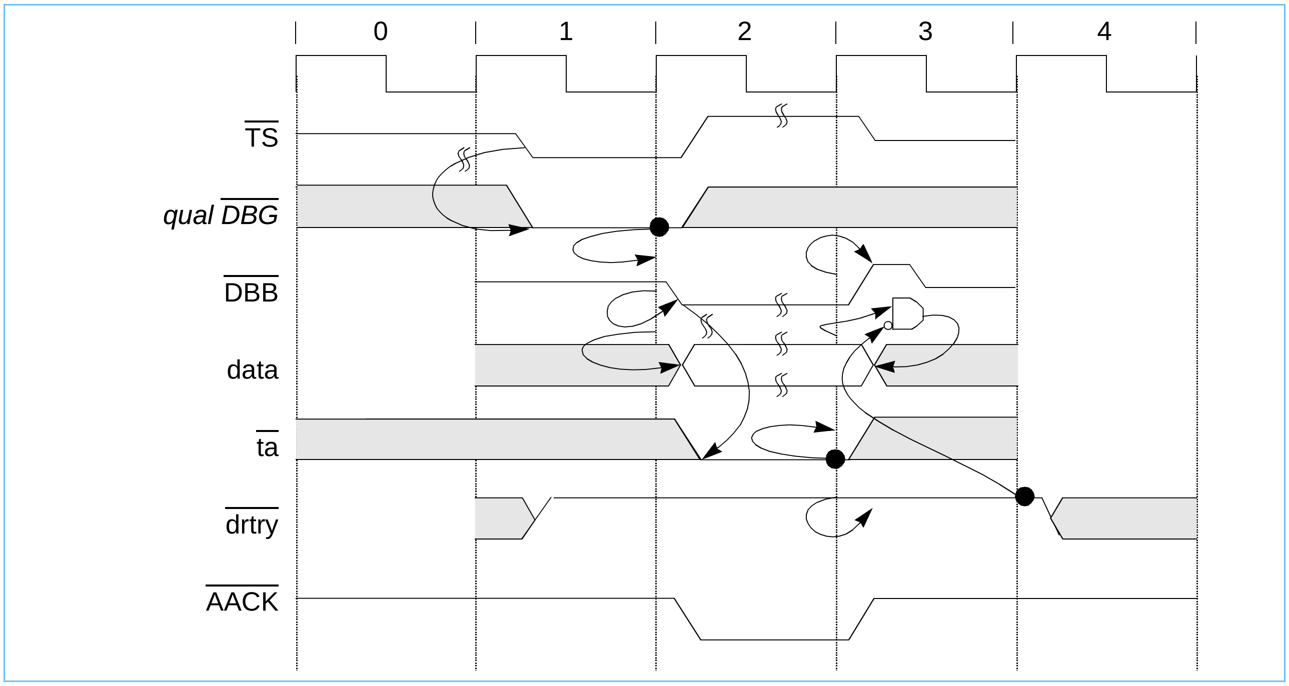
The TEA signal is used to signal a nonrecoverable error during the data transaction. It can be asserted on any cycle during DBB, or on the cycle after a qualified TA during a read operation, except when no-DRTRY mode is selected (where no-DRTRY mode cancels checking the cycle after TA). The assertion of TEA terminates the data tenure immediately even if in the middle of a burst. However, it does not prevent incorrect data that has just been acknowledged with a TA from being written into the 750GX cache or to GPRs. The assertion of TEA initiates either a machine-check exception or a checkstop condition based on the setting of the MSR[ME] bit.

An assertion of ARTRY causes the data tenure to be terminated immediately if the ARTRY is for the address tenure associated with the data tenure in operation. If ARTRY is connected for the 750GX, the earliest allowable assertion of TA to the 750GX is directly dependent on the earliest possible assertion of ARTRY to the 750GX; see *Section 8.3.3, Address Transfer Termination,* on page 300.

### 8.4.4.1 Normal Single-Beat Termination

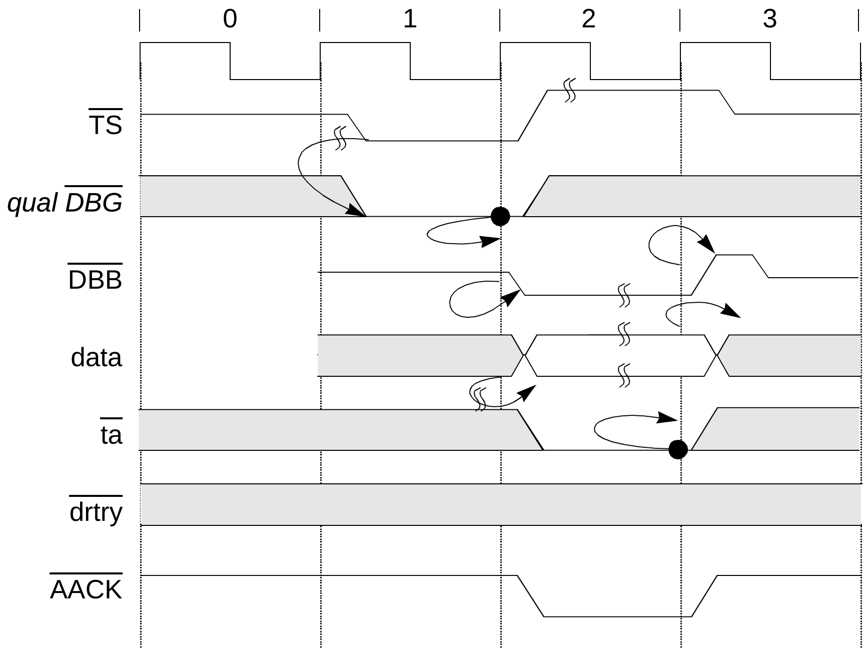
Normal termination of a single-beat data read operation occurs when TA is asserted by a responding slave. The TEA and DRTRY signals must remain negated during the transfer (see *Figure 8-11*).

*Figure 8-11. Normal Single-Beat Read Termination*



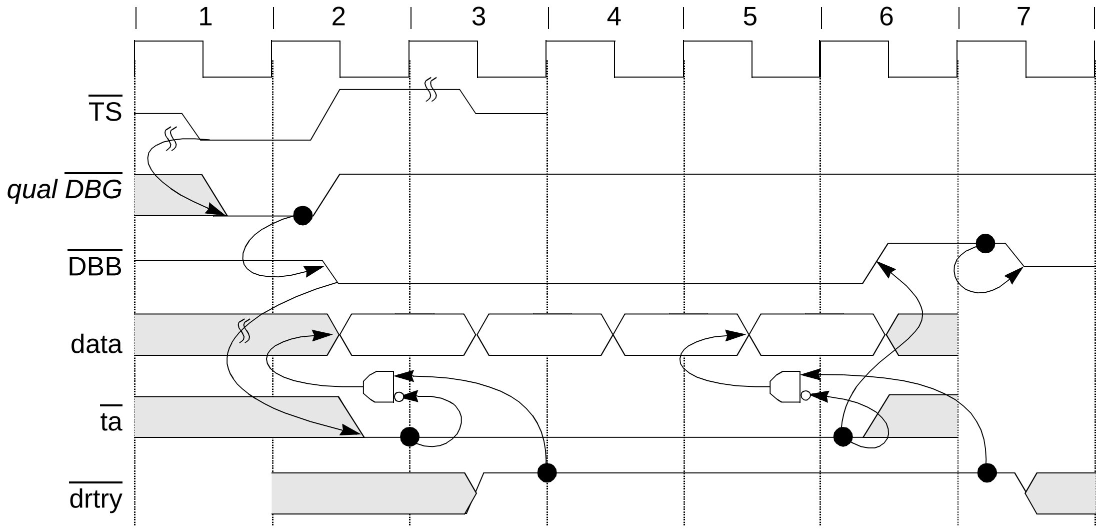
The DRTRY signal is not sampled during data writes, as shown in *Figure 8-12*.

*Figure 8-12. Normal Single-Beat Write Termination*



Normal termination of a burst transfer occurs when TA is asserted for four bus clock cycles, as shown in *Figure 8-13*. The bus clock cycles in which TA is asserted need not be consecutive, thus allowing pacing of the data-transfer beats. For read bursts to terminate successfully, TEA and DRTRY must remain negated during the transfer. For write bursts, TEA must remain negated for a successful transfer. DRTRY is ignored during data writes.

*Figure 8-13. Normal Burst Transaction*



For read bursts, DRTRY can be asserted one bus clock cycle after TA is asserted to signal that the data presented with TA is invalid and that the processor must wait for the negation of DRTRY before forwarding data to the processor (see *Figure 8-14*). Thus, a data beat can be terminated by a predicted branch with TA, and then one bus clock cycle later confirmed with the negation of DRTRY. The DRTRY signal is valid only for read transactions. TA must be asserted on the bus clock cycle before the first bus clock cycle of the assertion of DRTRY; otherwise, the results are undefined.

The DRTRY signal extends data-bus mastership such that other processors cannot use the data bus until DRTRY is negated. Therefore, in the example in *Figure 8-14*, data-bus tenure for the next transaction cannot begin until bus clock cycle 6. This is true for both read and write operations even though DRTRY does not extend bus mastership for write operations.

*Figure 8-14. Termination with DRTRY*

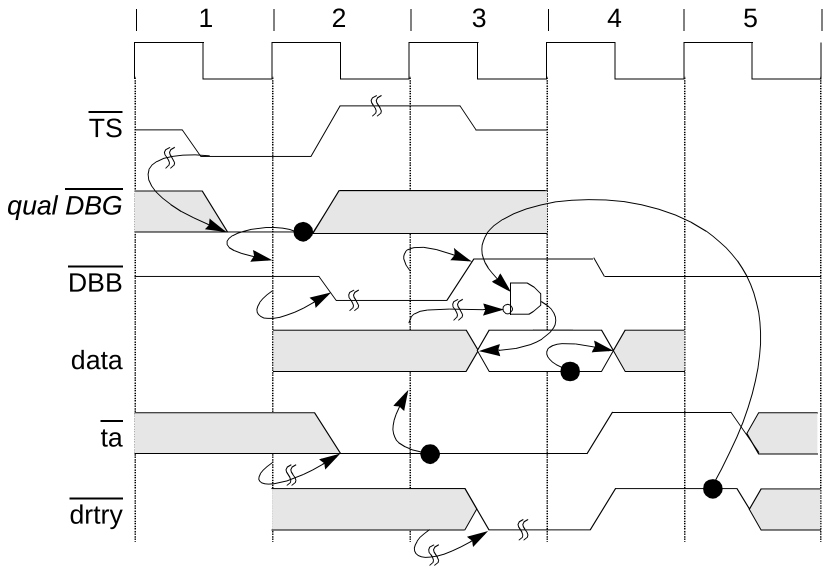
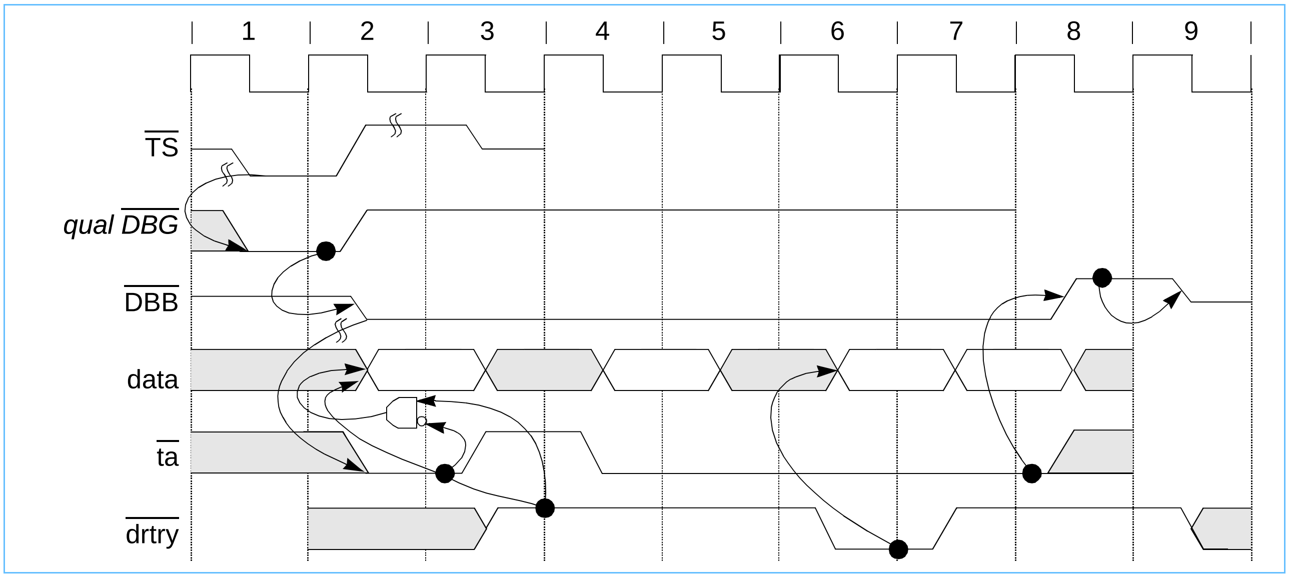


*Figure 8-15* shows the effect of using DRTRY during a burst read. It also shows the effect of using TA to pace the data-transfer rate. Notice that in bus clock cycle 3 of *Figure 8-15*, TA is negated for the second data beat. The 750GX data pipeline does not proceed until bus clock cycle 4 when the TA is reasserted.

*Figure 8-15. Read Burst with TA Wait States and DRTRY*



**Note:** DRTRY is useful for systems that implement predicted forwarding of data such as those with direct-mapped, third-level caches where hit or miss is determined on the following bus clock cycle, or for parity-checked or ECC-checked memory systems. Also note that DRTRY might not be implemented on other PowerPC processors.

### *8.4.4.2 Data-Transfer Termination Due to a Bus Error*

The TEA signal indicates that a bus error occurred. It might be asserted during data-bus tenure. Asserting TEA to the 750GX terminates the transaction. That is, further assertions of TA are ignored and the data-bus tenure is terminated.

Assertion of the TEA signal causes a machine-check exception (and possibly a checkstop condition within the 750GX).

**Note:** The 750GX does not implement a synchronous error capability for memory accesses. This means that the exception instruction pointer saved into Machine Status Save/Restore Register 0 (SRR0) does not point to the memory operation that caused the assertion of TEA, but to the instruction about to be executed (perhaps several instructions later). However, assertion of TEA does not invalidate data entering the GPR or the cache. Additionally, the address corresponding to the access that caused TEA to be asserted is not latched by the 750GX. To recover, the exception handler must determine and remedy the cause of the TEA, or the 750GX must be reset. Therefore, this function should only be used to indicate fatal system conditions to the processor.

After the 750GX has committed to run a transaction, that transaction must eventually complete. Address retry causes the transaction to be restarted. TA wait states and DRTRY assertion for reads delay termination of individual data beats. Eventually, however, the system must either terminate the transaction or assert the TEA signal. For this reason, care must be taken to check for the end of physical memory and the location of certain system facilities to avoid memory accesses that result in the assertion of TEA.

**Note:** $\overline{\text{TEA}}$ generates a machine-check exception depending on MSR[ME]. Clearing the machine-check-exception enable control bits leads to a true checkstop condition (instruction execution halted and processor clock stopped).

### 8.4.5 Memory Coherency—MEI Protocol

The 750GX provides dedicated hardware to provide memory coherency by snooping bus transactions. The address retry capability enforces the 3-state, MEI cache-coherency protocol (see *Figure 8-16* on page 309).

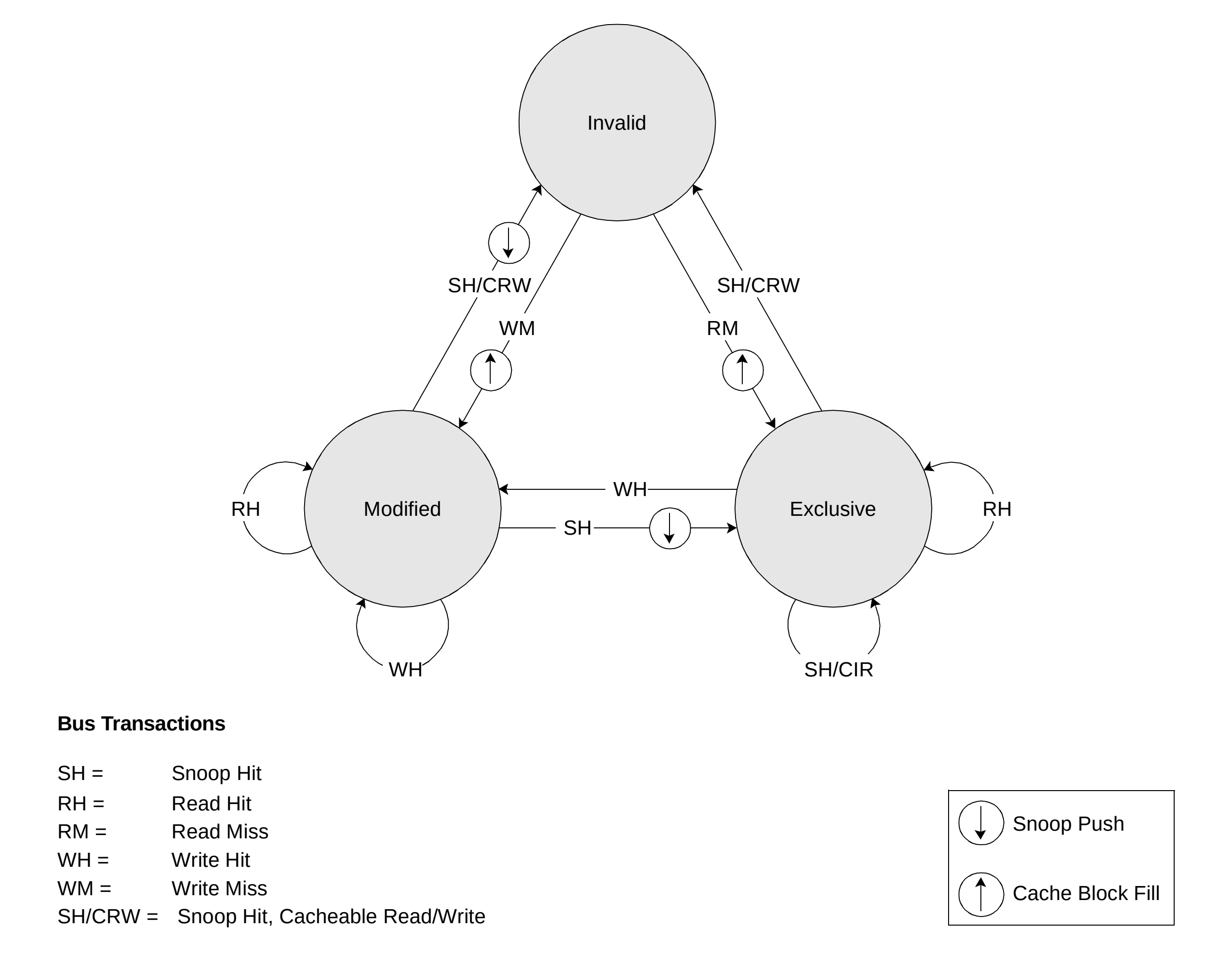
The global ($\overline{\text{GBL}}$) output signal indicates whether the current transaction must be snooped by other snooping devices on the bus. Address-bus masters assert $\overline{\text{GBL}}$ to indicate that the current transaction is a global access (that is, an access to memory shared by more than one device). If $\overline{\text{GBL}}$ is not asserted for the transaction, that transaction is not snooped. When other devices detect the $\overline{\text{GBL}}$ input asserted, they must respond by snooping the broadcast address.

Normally, $\overline{\text{GBL}}$ reflects the M bit value specified for the memory reference in the corresponding translation descriptors. Note that care must be taken to minimize the number of pages marked as global, because the retry protocol discussed in the previous section is used to enforce coherency and can require significant bus bandwidth.

When the 750GX is not the address-bus master, $\overline{\text{GBL}}$ is an input. The 750GX snoops a transaction if $\overline{\text{TS}}$ and $\overline{\text{GBL}}$ are asserted together in the same bus clock cycle (this is a qualified snooping condition). No snoop update to the 750GX cache occurs if the snooped transaction is not marked global. This includes invalidation cycles.

When the 750GX detects a qualified snoop condition, the address associated with the $\overline{\text{TS}}$ is compared against the data-cache tags. Snooping completes if no hit is detected. If, however, the address hits in the cache, the 750GX reacts according to the MEI protocol shown in *Figure 8-16*, assuming the WIM bits are set to write-back, caching-enabled, and coherency-enforced modes (WIM = 001).

*Figure 8-16. MEI Cache-Coherency Protocol—State Diagram (WIM = 001)*



## 8.5 Timing Examples

This section shows timing diagrams for various scenarios. *Figure 8-17* on page 310 illustrates the fastest single-beat reads possible for the 750GX. This figure shows both minimal latency and maximum single-beat throughput. By delaying the data-bus tenure, the latency increases, but, because of split-transaction pipelining, the overall throughput is not affected unless the data-bus latency causes the third address tenure to be delayed.

Note that all bidirectional signals are tristated between bus tenures.

*Figure 8-17. Fastest Single-Beat Reads*

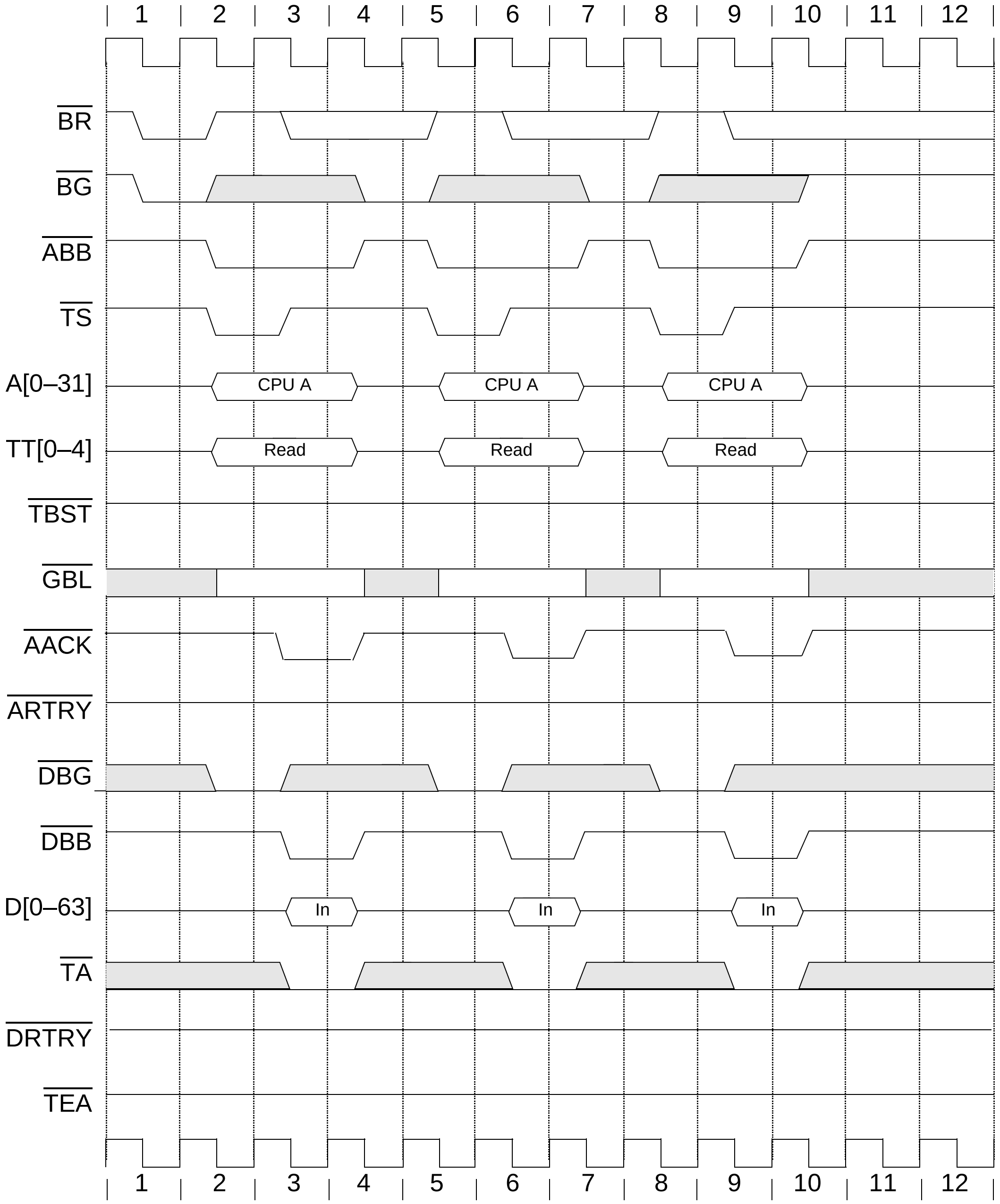


*Figure 8-18* illustrates the fastest single-beat writes supported by the 750GX. All bidirectional signals are tristated between bus tenures.

*Figure 8-18. Fastest Single-Beat Writes*

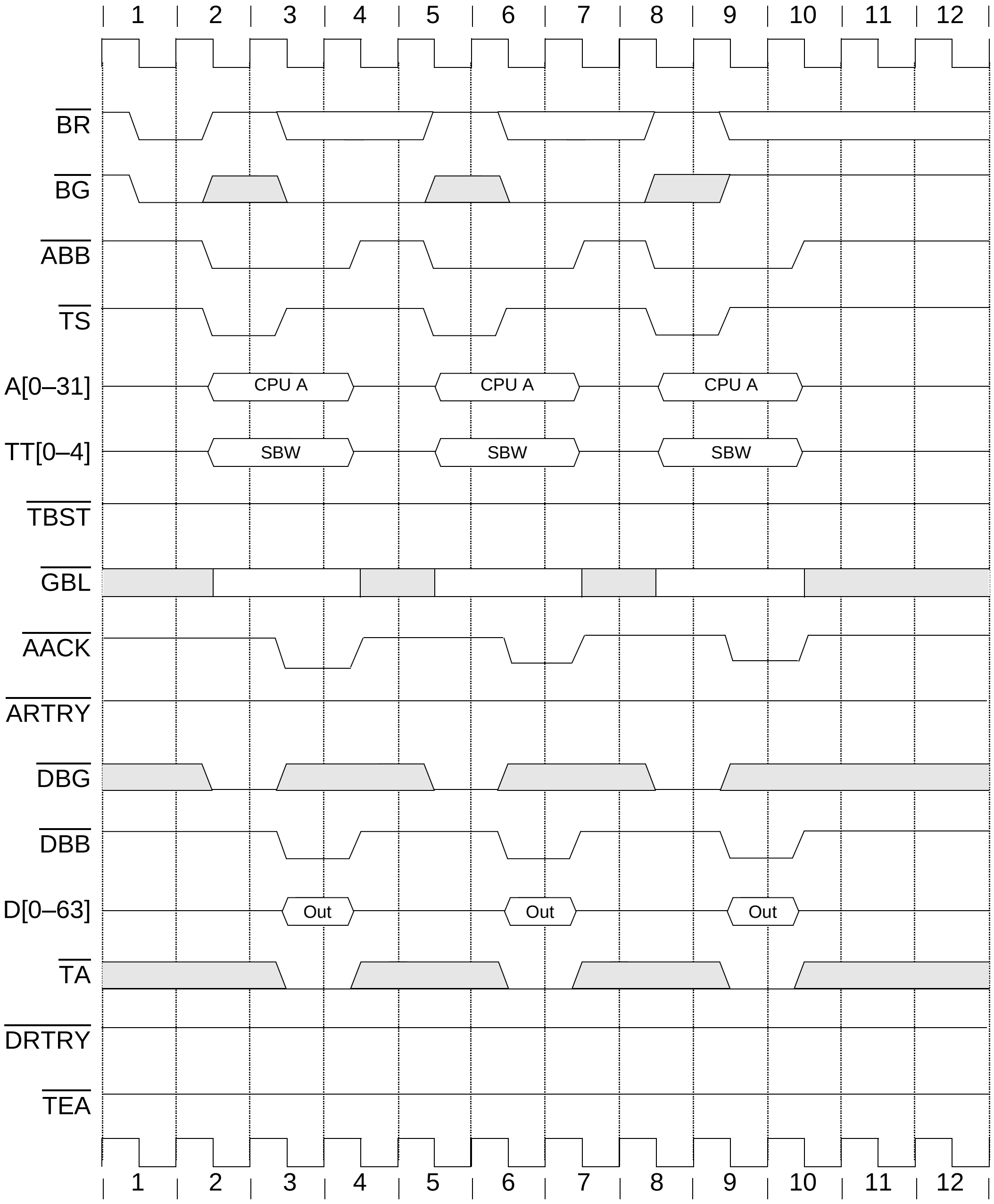


*Figure 8-19* shows three ways to delay single-beat reads using data-delay controls:

- The $\overline{\text{TA}}$ signal can remain negated to insert wait states in clock cycles 3 and 4.
- For the second access, $\overline{\text{DBG}}$ could have been asserted in clock cycle 6.
- In the third access, $\overline{\text{DRTRY}}$ is asserted in clock cycle 11 to flush the previous data.

**Note:** All bidirectional signals are tristated between bus tenures. The pipelining shown in *Figure 8-19* can occur if the second access is not another load (for example, an instruction fetch).

*Figure 8-19. Single-Beat Reads Showing Data-Delay Controls*

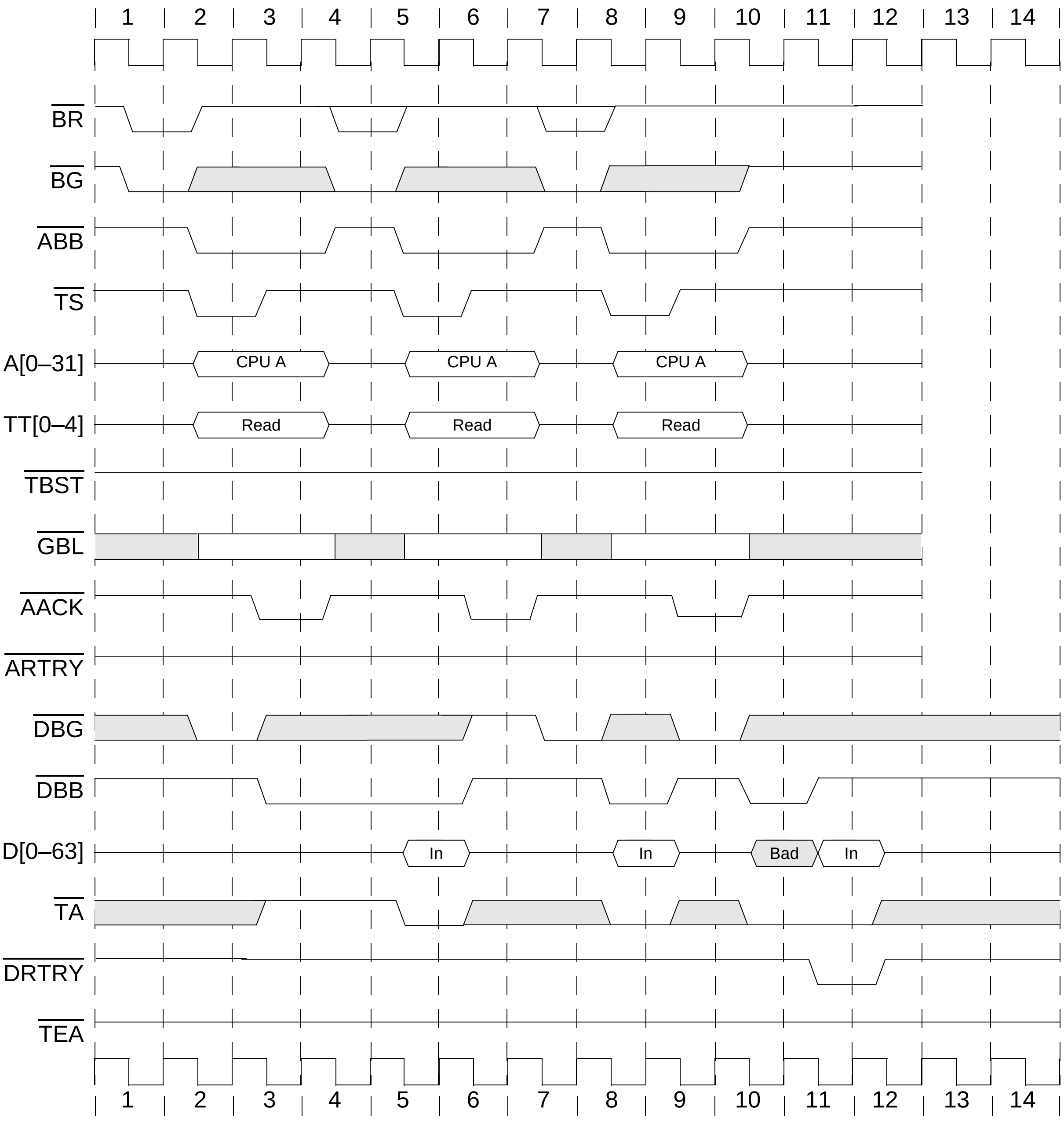


*Figure 8-20* shows data-delay controls in a single-beat write operation. Note that all bidirectional signals are tristated between bus tenures. Data transfers are delayed in the following ways:

- The $\overline{\text{TA}}$ signal is held negated to insert wait states in clocks 3 and 4.
- In clock 6, $\overline{\text{DBG}}$ is held negated, delaying the start of the data tenure.

The last access is not delayed ($\overline{\text{DRTRY}}$ is valid only for read operations).

*Figure 8-20. Single-Beat Writes Showing Data-Delay Controls*

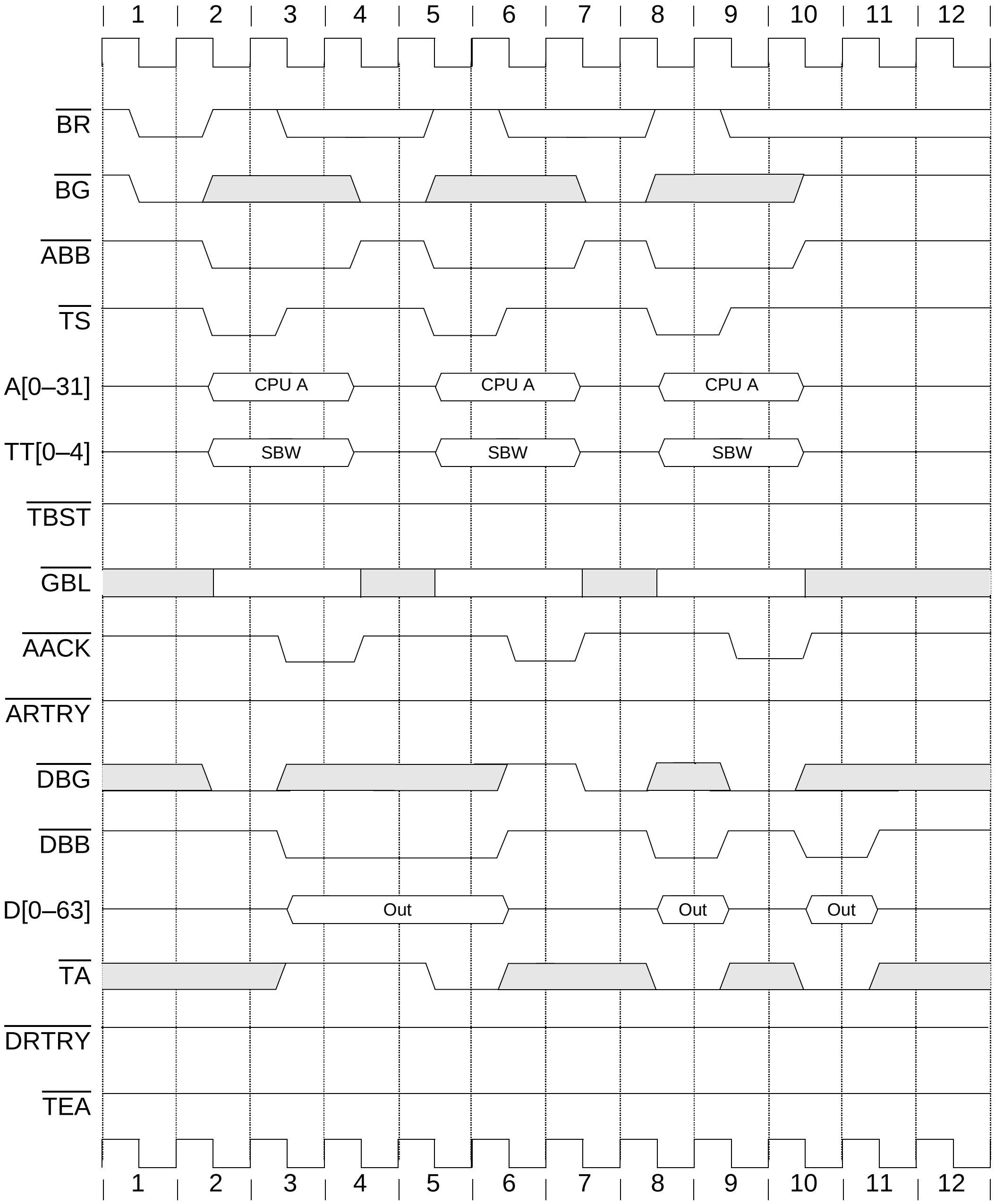


*Figure 8-21* shows the use of data-delay controls with burst transfers. Note that all bidirectional signals are tristated between bus tenures. Also note:

- The first data beat of burst read data (clock 0) is the critical quadword.
- The write burst shows the use of $\overline{\text{TA}}$ signal negation to delay the third data beat.
- The final read burst shows the use of $\overline{\text{DRTRY}}$ on the third data beat.
- The address for the third transfer is delayed until the first transfer completes.

*Figure 8-21. Burst Transfers with Data-Delay Controls*

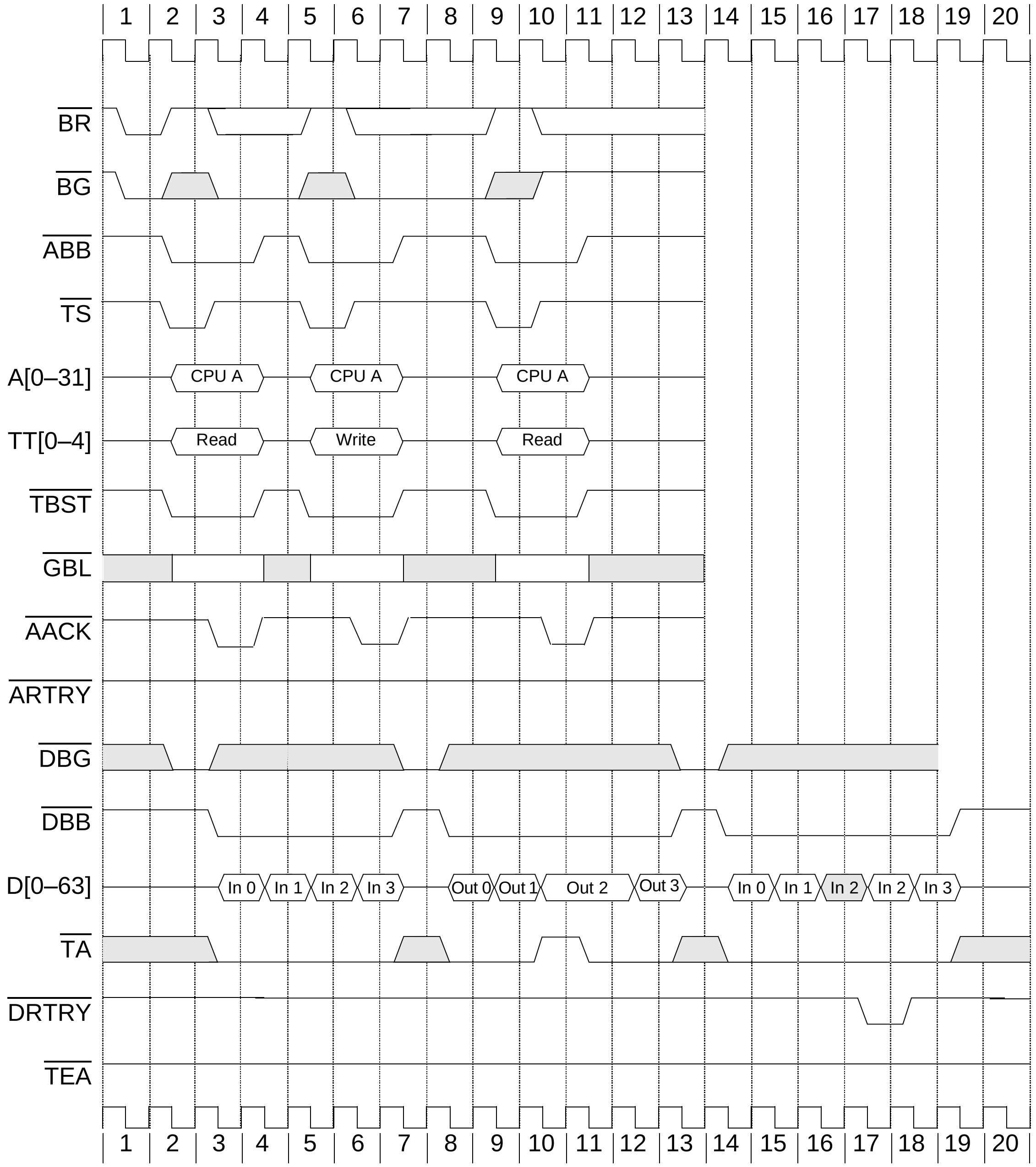
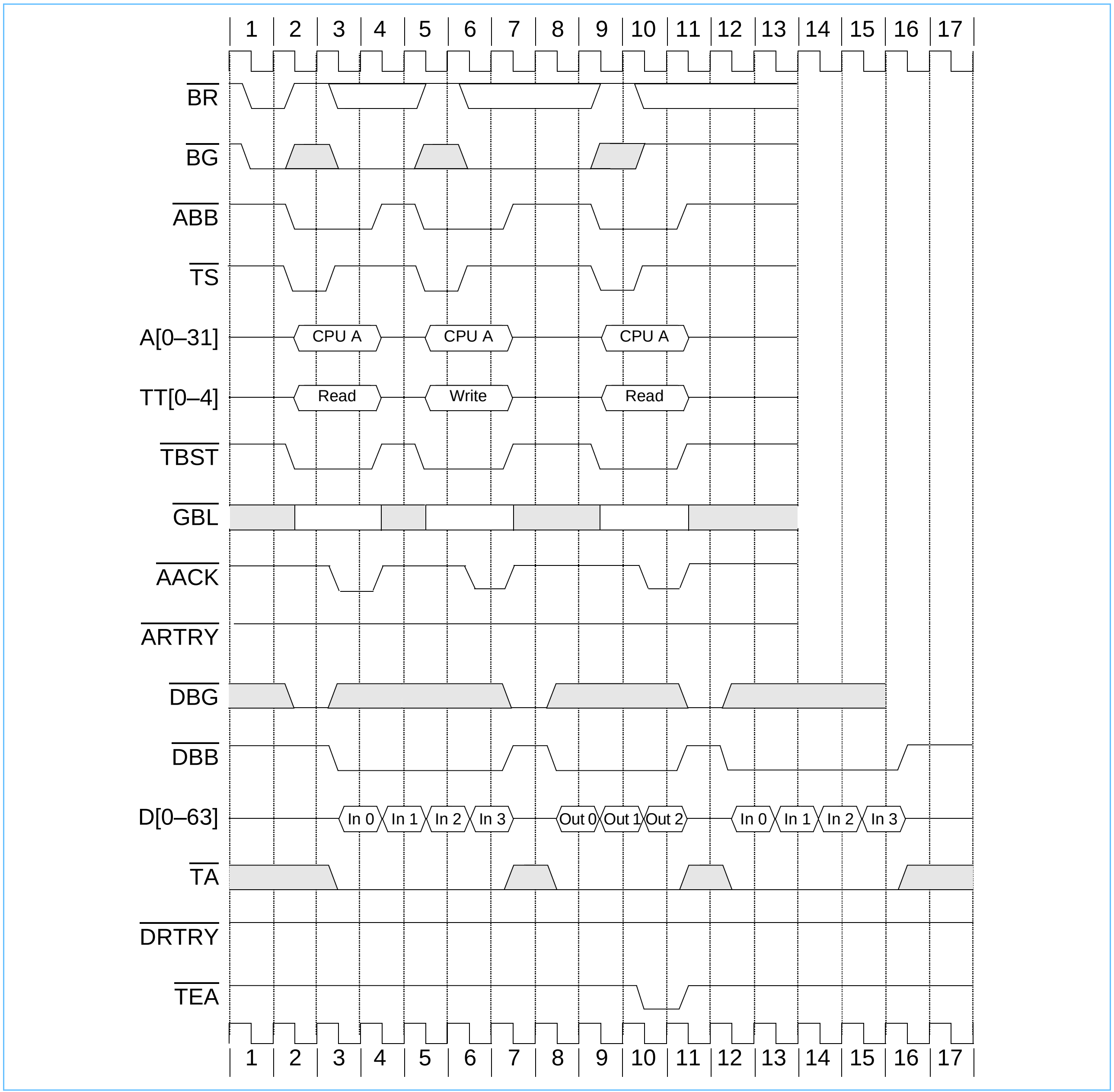


*Figure 8-22* shows the use of the $\overline{\text{TEA}}$ signal. Note that all bidirectional signals are tristated between bus tenures. Also note:

- The first data beat of the read burst (in clock 0) is the critical quadword.
- The $\overline{\text{TEA}}$ signal truncates the burst write transfer on the third data beat.
- The 750GX eventually causes an exception to be taken on the $\overline{\text{TEA}}$ event.

*Figure 8-22. Use of Transfer Error Acknowledge ($\overline{\text{TEA}}$)*



## 8.6 Optional Bus Configuration

The 750GX supports optional bus configurations that are selected during the negation of the HRESET signal. The operation and selection of the optional bus configuration are described in the following sections.

### 8.6.1 32-Bit Data Bus Mode

The 750GX supports an optional 32-bit data bus mode. 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. When in 32-bit data bus mode, only byte lanes 0 through 3 are used corresponding to DH0–DH31 and DP0–DP3. Byte lanes 4 through 7 corresponding to DL0–DL31 and DP4–DP7 are never used in this mode. The unused data bus signals are not sampled by the 750GX during read operations, and they are driven low during write operations.

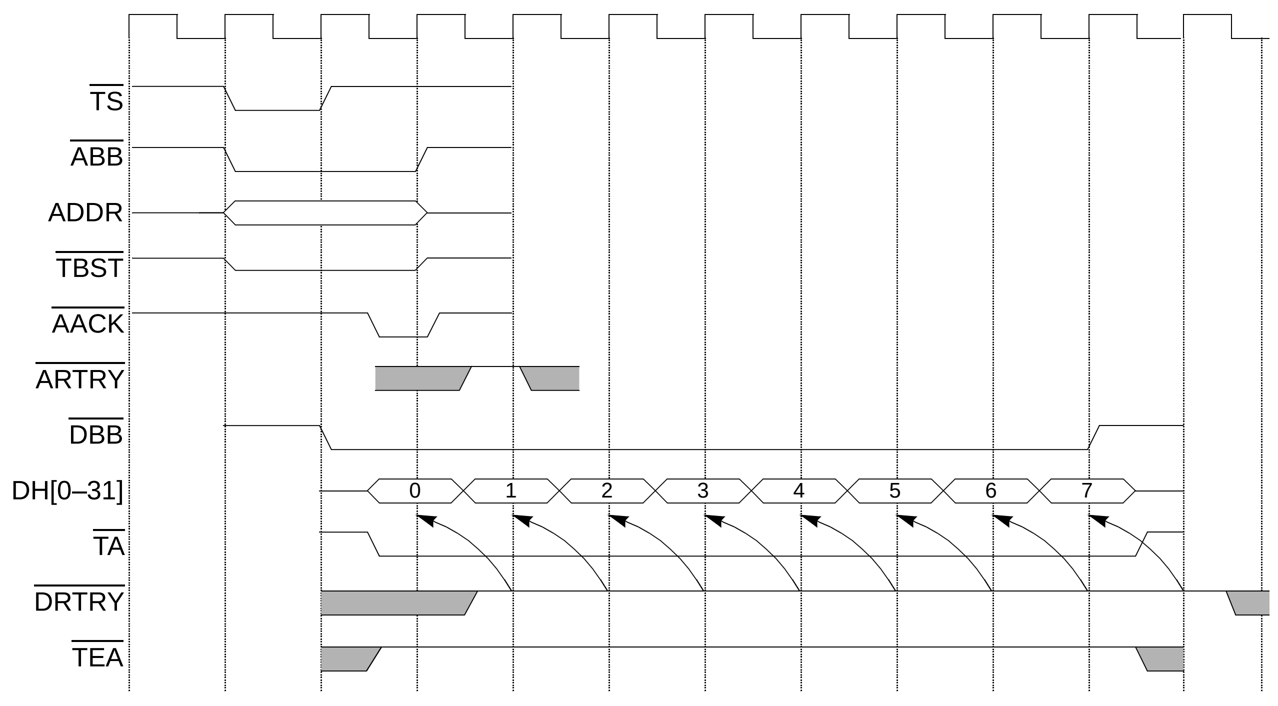
The number of data beats required for a data tenure in the 32-bit data bus mode is one, two, or eight beats depending on the size of the program transaction and the cache mode for the address. Data transactions of one or two data beats are performed for caching-inhibited load/store or write-through store operations. These transactions do not assert the TBST signal even though a two-beat burst may be performed (having the same TBST and TSIZ[0–2] encodings as the 64-bit data bus mode). Single-beat data transactions are performed for bus operations of 4 bytes or less, and double-beat data transactions are performed for 8-byte operations only. The 750GX only generates an 8-byte operation for a double-word-aligned load or store double operation to or from the Floating Point Registers. All cache-inhibited instruction fetches are performed as word (single-beat) operations.

Data transactions of eight data beats are performed for burst operations that load into or store from the 750GX's internal caches. These transactions transfer 32 bytes in the same way as in 64-bit data bus mode, asserting the TBST signal, and signaling a transfer size of 2 (TSIZ(0–2) = 0b010).

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

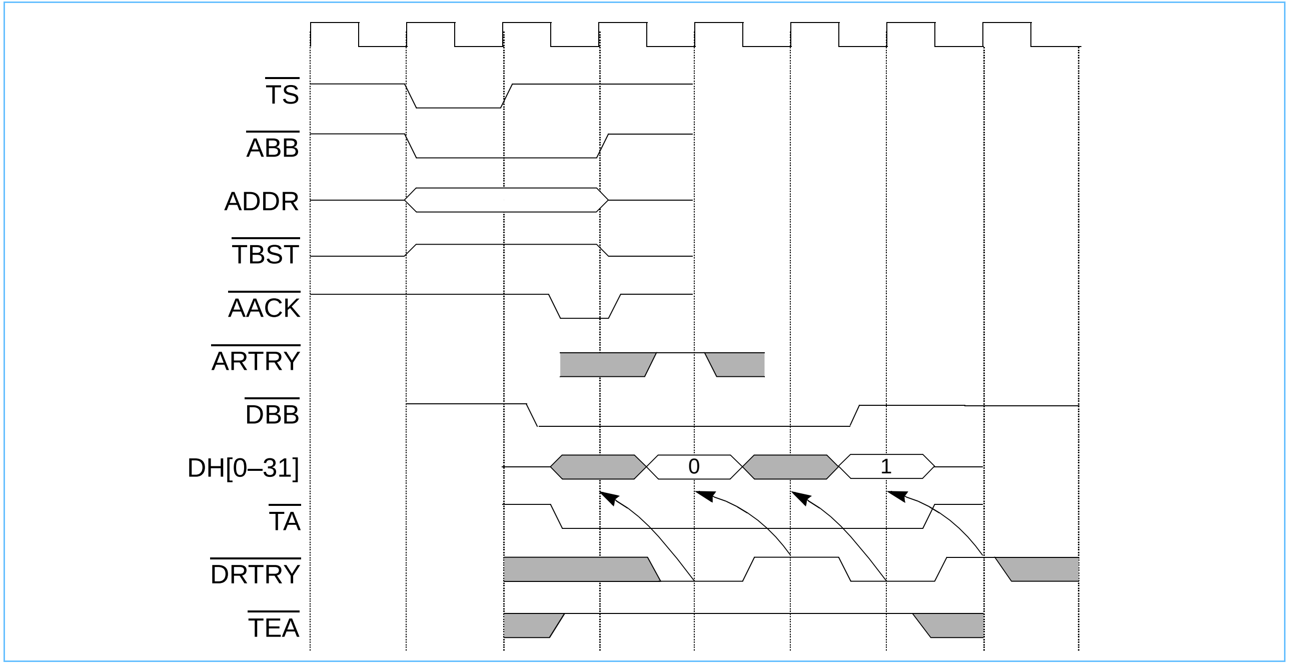
An example of an eight-beat data transfer while the 750GX is in 32-bit data bus mode is shown in *Figure 8-23* on page 317.

*Figure 8-23. 32-Bit Data-Bus Transfer (8-Beat Burst)*



An example of a two-beat data transfer (with DRTRY asserted during each data tenure) is shown in *Figure 8-24*.

*Figure 8-24. 32-Bit Data-Bus Transfer (2-Beat Burst with DRTRY)*



The 750GX selects 64-bit or 32-bit data bus mode at startup by sampling the state of the TLBISYNC signal at the negation of HRESET. If the TLBISYNC signal is negated at the negation of HRESET, the 750GX enters 64-bit data mode. If TLBISYNC is asserted at the negation of HRESET, the 750GX enters 32-bit data mode.

*Table 8-3* on page 296 describes the burst ordering when the 750GX is in 32-bit mode.

The aligned data-transfer cases for 32-bit data bus mode are shown in *Table 8-6*. All of the transfers require a single data beat (if caching-inhibited or write-through) except for double-word cases which require two data beats. The double-word case is only generated by the 750GX for load or store double operations to/from the Floating Point Registers. All caching-inhibited instruction fetches are performed as word operations.

Misaligned data transfers in the 32-bit bus mode is the same as in the 64-bit bus mode with the exception that only DH[0-31] data lines are used. *Table 8-7* shows examples of 4-byte misaligned transfers starting at each possible byte address within a double word.

### 8.6.2 No-DRTRY Mode

The 750GX supports an optional mode to disable the use of the data retry function provided through the DRTRY signal. The no-DRTRY mode allows the forwarding of data during load operations to the internal CPU one bus cycle sooner than in the normal bus protocol.

The 60x bus protocol specifies that, during load operations, the memory system can, normally, cancel data that was read by the master on the bus cycle after TA was asserted. In the 750GX implementation, this late cancellation protocol requires the 750GX to hold any loaded data at the bus interface for one additional bus clock to verify that the data is valid before forwarding it to the internal CPU. For systems that do not implement the DRTRY function, the 750GX provides an optional no-DRTRY mode that eliminates this 1-cycle stall during all load operations, and allows for the forwarding of data to the internal CPU immediately when TA is recognized.

When the 750GX is in the no-DRTRY mode, data can no longer be cancelled the cycle after it is acknowledged by an assertion of TA. Data is immediately forwarded to the CPU internally, and any attempt at late cancellation by the system might cause improper operation by the 750GX.

When the 750GX is following normal bus protocol, data might be cancelled the bus cycle after TA by either of two means—late cancellation by DRTRY, or late cancellation by ARTRY. When no-DRTRY mode is selected, both cancellation cases must be disallowed in the system design for the bus protocol.

When no-DRTRY mode is selected for the 750GX, the system must ensure that DRTRY is not asserted to the 750GX. If it is asserted, it can cause improper operation of the bus interface. The system must also ensure that an assertion of ARTRY by a snooping device occurs before or coincident with the first assertion of TA to the 750GX, but not on the cycle after the first assertion of TA.

Other than the inability to cancel data that was read by the master on the bus cycle after TA was asserted, the bus protocol for the 750GX is identical to that for the basic transfer bus protocols described in this section, including 32-bit data-bus mode.

The 750GX selects the desired DRTRY mode at startup by sampling the state of the DRTRY signal itself at the negation of the HRESET signal. If the DRTRY signal is negated at the negation of HRESET, normal operation is selected. If the DRTRY signal is asserted at the negation of HRESET, no-DRTRY mode is selected.

## 8.7 Processor State Signals

This section describes the 750GX's support for atomic update and memory through the use of the **lwarx** and **stwcx.** opcode pair, and includes a description of the TLB Invalidate Synchronize (TLBISYNC) input.

### 8.7.1 Support for the lwarx and stwcx. Instruction Pair

The Load Word and Reserve Indexed (**lwarx**) and the Store Word Conditional Indexed (**stwcx.**) instructions provide a means for atomic memory updating. Memory can be updated atomically by setting a reservation on the load and checking that the reservation is still valid before the store is performed. In the 750GX, the reservations are made on behalf of aligned, 32-byte sections of the memory address space.

The reservation (RSRV) output signal is driven synchronously with the bus clock and reflects the status of the reservation coherency bit in the Reservation Address Register; see *Chapter 3, Instruction-Cache and Data-Cache Operation,* on page 121 for more information. For information about timing, see *Section 7.2.11.3, Reservation (RSRV)—Output,* on page 273.

### 8.7.2 TLBISYNC Input

The TLBISYNC input allows for the hardware synchronization of changes to MMU tables when the 750GX and another direct memory access (DMA) master share the same MMU translation tables in system memory. It is asserted by a DMA master when it is using shared addresses that could be changed in the MMU tables by the 750GX during the DMA master's tenure.

The TLBISYNC input, when asserted to the 750GX, prevents the 750GX from completing any instructions past a TLB Synchronize (**tlbsync)** instruction. Generally, during the execution of an **eciwx** or **ecowx** instruction by the 750GX, the selected DMA device should assert the 750GX's TLBISYNC signal and maintain it asserted during its DMA tenure if it is using a shared translation address. Subsequent instructions by the 750GX should include a **sync** and **tlbsync** instruction before any MMU table changes are performed. This will prevent the 750GX from making table changes disruptive to the other master during the DMA period.

## 8.8 IEEE 1149.1a-1993 Compliant Interface

The 750GX boundary-scan interface is a fully-compliant implementation of the IEEE 1149.1a-1993 standard. This section describes the 750GX's IEEE 1149.1a-1993 (JTAG) interface.

### 8.8.1 JTAG/COP Interface

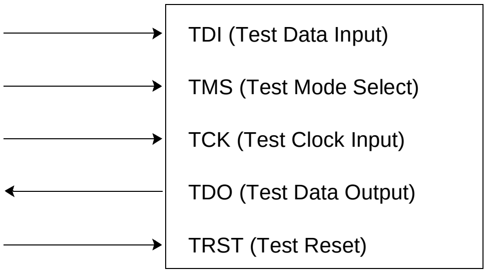
The 750GX has extensive on-chip test capability including the following:

- Debug control/observation (COP)
- Boundary scan (standard IEEE 1149.1a-1993 [JTAG] compliant interface)
- Support for manufacturing test

The COP and boundary scan logic are not used under typical operating conditions. Detailed discussion of the 750GX test functions is beyond the scope of this document. However, sufficient information has been provided to allow the system designer to disable the test functions that would impede normal operation.

The JTAG/COP interface is shown in *Figure 8-25*. For more information, see *IEEE Standard Test Access Port and Boundary Scan Architecture IEEE STD 1149.1a-1993*.

*Figure 8-25. IEEE 1149.1a-1993 Compliant Boundary-Scan Interface*

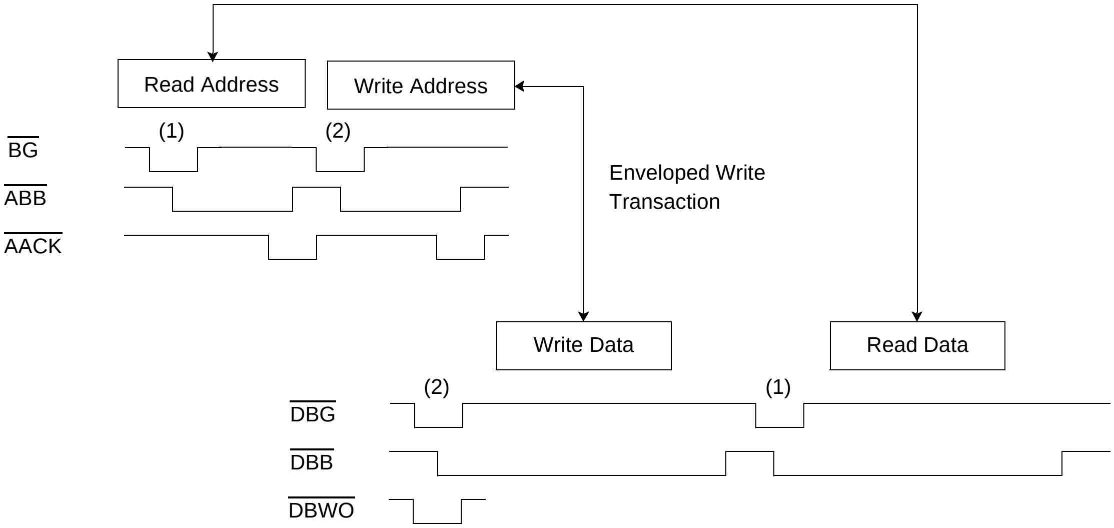


## 8.9 Using Data-Bus Write-Only

The 750GX supports split-transaction pipelined transactions. It supports a limited out-of-order capability for its own pipelined transactions through the data-bus write-only (DBWO) signal. When recognized on the clock of a qualified DBG, the assertion of DBWO directs the 750GX to perform the next pending data write tenure (if any), even if a pending read tenure would have normally been performed because of address pipelining. The DBWO signal does not change the order of write tenures with respect to other write tenures from the same 750GX. It only allows a write tenure to be performed ahead of a pending read tenure from the same 750GX.

In general, an address tenure on the bus is followed strictly in order by its associated data tenure. Transactions pipelined by the 750GX complete strictly in order. However, the 750GX can run bus transactions out of order only when the external system allows the 750GX to perform a cache-line-snoop-push-out operation (or other write transaction, if pending in the 750GX write queues) between the address and data tenures of a read operation through the use of DBWO. This effectively envelopes the write operation within the read operation. *Figure 8-26* shows how the DBWO signal is used to perform an enveloped write transaction.

*Figure 8-26. Data-Bus Write-Only Transaction*



Note that although the 750GX can pipeline any write transaction behind the read transaction, special care should be used when using the enveloped write feature. It is envisioned that most system implementations will not need this capability; for these applications, $\overline{\text{DBWO}}$ should remain negated. In systems where this capability is needed, $\overline{\text{DBWO}}$ should be asserted under the following scenario:

1. The 750GX initiates a read transaction (either single-beat or burst) by completing the read address tenure with no address retry.
2. Then, the 750GX initiates a write transaction by completing the write address tenure, with no address retry.
3. At this point, if $\overline{\text{DBWO}}$ is asserted with a qualified data-bus grant to the 750GX, the 750GX asserts $\overline{\text{DBB}}$ and drives the write data onto the data bus, out of order with respect to the address pipeline. The write transaction concludes with the 750GX negating $\overline{\text{DBB}}$.
4. The next qualified data-bus grant signals the 750GX to complete the outstanding read transaction by latching the data on the bus. This assertion of $\overline{\text{DBG}}$ should not be accompanied by an asserted $\overline{\text{DBWO}}$.

Any number of bus transactions by other bus masters can be attempted between any of these steps.

Note the following regarding $\overline{\text{DBWO}}$:

- $\overline{\text{DBWO}}$ can be asserted if no data-bus read is pending, but it has no effect on write ordering.
- The ordering and presence of data-bus writes is determined by the writes in the write queues at the time $\overline{\text{BG}}$ is asserted for the write address (not $\overline{\text{DBG}}$). If a particular write is desired (for example, a cache-line-snoop-push-out operation), then $\overline{\text{BG}}$ must be asserted after that particular write is in the queue, and it must be the highest priority write in the queue at that time. A cache-line-snoop-push-out operation might be the highest priority write, but more than one might be queued.
- Because more than one write might be in the write queue when $\overline{\text{DBG}}$ is asserted for the write address, more than one data-bus write can be enveloped by a pending data-bus read.

The arbiter must monitor bus operations and coordinate the various masters and slaves with respect to the use of the data bus when $\overline{\text{DBWO}}$ is used. Individual $\overline{\text{DBG}}$ signals associated with each bus device should allow the arbiter to synchronize both pipelined and split-transaction bus organizations. Individual $\overline{\text{DBG}}$ and $\overline{\text{DBWO}}$ signals provide a primitive form of source-level tagging for the granting of the data bus.

Note that use of the $\overline{\text{DBWO}}$ signal allows some operation-level tagging with respect to the 750GX and the use of the data bus.

# 9. L2 Cache

This chapter describes the 750GX microprocessor‘s implementation of the 1-MB L2 cache.

**Note:** The L2 cache is initially disabled following a power-on or hard reset. Before enabling the L2 cache, configuration parameters must be set in the L2 Cache Control Register (L2CR), and the L2 tags must be globally invalidated. The L2 cache should be initialized during system start-up (see *Section 9.4* on page 329).

## 9.1 L2 Cache Overview

The 750GX microprocessor's L2 cache is implemented with an internal 4-way set-associative tag memory with 4096 tags per way, and an internal 1-MB SRAM for data storage. The tags are sectored to support two cache blocks per tag entry (two 32-byte sectors totalling 64 bytes). Each sector (32-byte cache block) in the L2 cache has its own valid and modified bits. Each set of four cache lines maintains three least recently used (LRU) bits to implement a pseudo-LRU replacement mechanism. In addition, the SRAM includes an 8-bit error correction code (ECC) for every double word. The ECC logic corrects most single-bit errors and detects the remaining single-bit errors and all double-bit errors as data is read from the SRAM. The L2 cache maintains cache coherency through snooping, and is normally configured to operate in copy-back mode.

The L2 Cache Control Register (L2CR) allows control of the following:

- L2-cache configuration
- Double-bit error machine check
- Global invalidation of L2 contents
- Write-through operation
- L2 test support
- L2 locking by way
- Data-only and instruction-only modes

## 9.2 L2 Cache Operation

The L2 cache for the 750GX microprocessor is a combined instruction and data cache that receives memory requests from both L1 instruction and L1 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 cache 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 castout from the L2 cache to the 60x bus.

At any given time, the L1 instruction cache might have one instruction fetch request, and the L1 data cache might have four loads 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 multiple read requests from the L1 caches are pending, the L2 cache can perform hit-under-miss operations, supplying the available instruction or data while a bus transaction for previous L2 cache misses is being performed. The L2 cache also supports miss-under-miss operation. Up to four outstanding misses are supported: one miss from the instruction cache and three from the data cache, or four data-cache misses. Requests that hit will be serviced even while misses are in progress on the bus.

All requests to the L2 cache that are marked cacheable (even if the respective L1 cache is disabled or locked) cause tag lookup and will be serviced if the instructions or data are in the L2 cache. Burst and single-beat read requests from the L1 caches that hit in the L2 cache are forwarded to the L1 caches as 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.

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 read requests from the L1 caches 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 copy-back that hit in the L2 cache 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 cache are forwarded to the 60x bus. Single-beat store requests marked write-through (through address translation or through the configuration of L2CR[WT]) 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. All requests to the L2 cache that are marked cache-inhibited by address translation, through either the memory management unit (MMU) or by the default write/cache inhibit/memory coherence/guarded storage (WIMG) configuration, bypass the L2 cache and do not cause any L2-cache tag state change.

The 750GX microprocessor 4-way set-associative L2 cache uses a 3-bit per set, pseudo-LRU replacement algorithm. Bit 0 is the global LRU bit, which indicates that the LRU way is in the lower (0 or 1) or upper (2 or 3) ways. Bit 1 is the lower LRU bit, which indicates that the LRU way is either way 0 or way 1. Bit 2 is the upper LRU bit, which indicates that the LRU way is either way 2 or way 3. Together, the three bits represent a partial ordering of the four ways, such that the LRU and most recently used (MRU) ways are identified, but the other two ways are not ordered with respect to each other. *Table 9-1* shows the interpretation of the three LRU bits in the absence of any cache locking.

*Table 9-1. Interpretation of LRU Bits*

| LRU Bits | LRU Way | MRU Way |
|---|---|---|
| 000 | 0 | 3 |
| 001 | 0 | 2 |
| 010 | 1 | 3 |
| 011 | 1 | 2 |
| 100 | 2 | 1 |
| 110 | 2 | 0 |
| 101 | 3 | 1 |
| 111 | 3 | 0 |

Whenever a way in the set is referenced, the LRU bits are updated. The new value of the LRU bits depends on the old value, which way is currently being accessed, and whether the operation is an invalidation or a load/store. *Table 9-2* shows the new value of the LRU bits for the various combinations of these variables. An ‘x’ indicates don’t care, while a ‘-’ indicates no change from previous value.

*Table 9-2. Modification of LRU Bits*

| Old LRU | Hit Way | Invalidate | New LRU |
|---|---|---|---|
| 00x | none | x | 11- |
| 01x | none | x | 10- |
| 1x0 | none | x | 0-1 |
| 1x1 | none | x | 0-0 |
| xxx | 0 | 0 | 11- |
| xxx | 1 | 0 | 10- |
| xxx | 2 | 0 | 0-1 |
| xxx | 3 | 0 | 0-0 |
| xxx | 0 | 1 | 00- |
| xxx | 1 | 1 | 01- |
| xxx | 2 | 1 | 1-0 |
| xxx | 3 | 1 | 1-1 |

The 4-way set-associative L2 cache can be locked by way as described below. The determination of the new LRU value does not depend on the locked status of the ways. However, the interpretation of the LRU bits shown in *Table 9-2* does change when one or more ways of the cache are locked.

Any combination of ways can be locked. The effect of locking on the replacement algorithm is that the least recently used of the unlocked ways is chosen for replacement. *Table 9-3* shows the interpretation of the LRU bits in the presence of one or two locked ways. If three ways are locked, the unlocked way is always replaced, and if all four ways are locked, no replacement takes place. In *Table 9-3*, bit zero of the lock bits controls whether way 0 is locked, bit one controls whether way 1 is locked, and so forth.

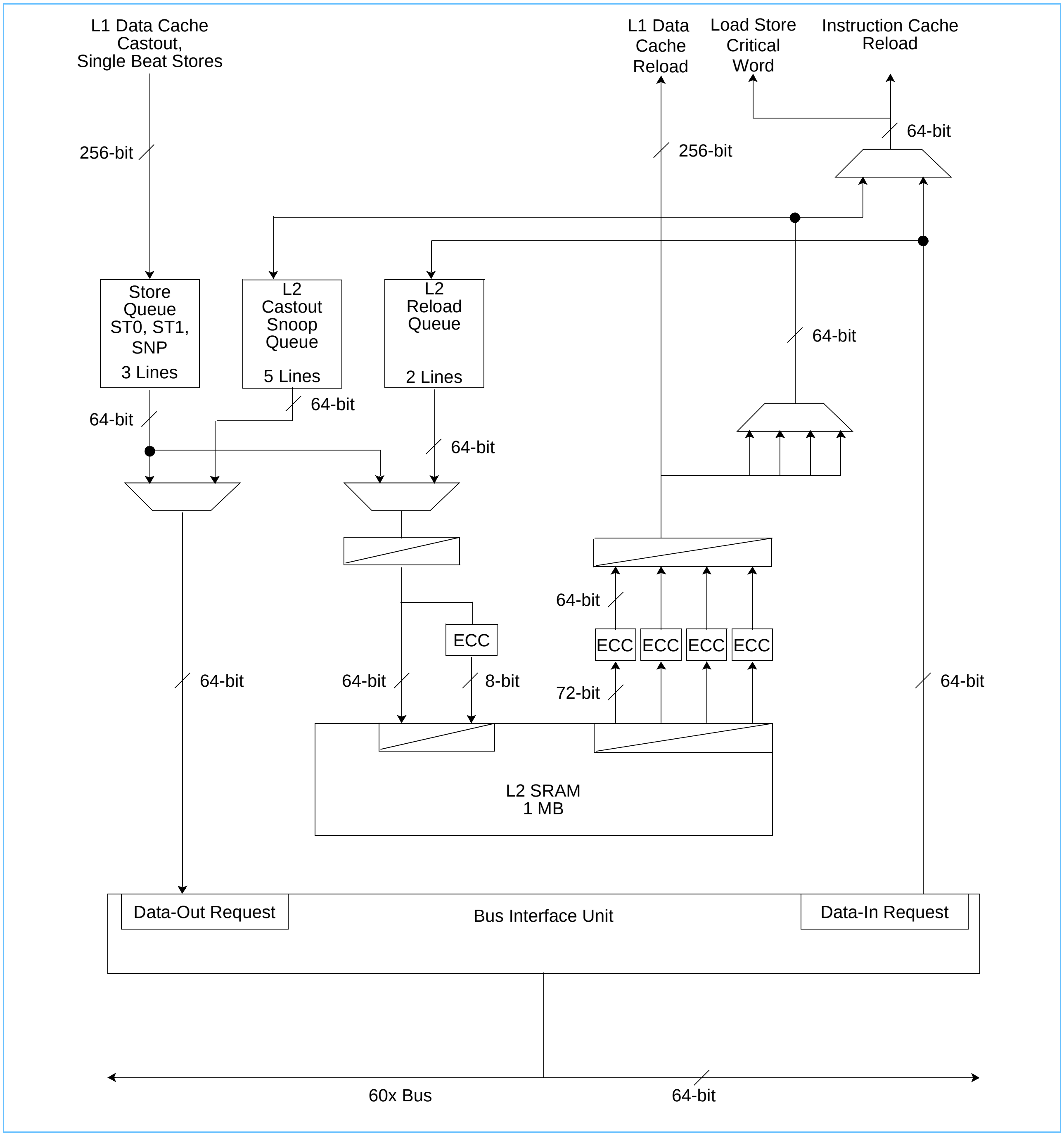
*Table 9-3. Effect of Locked Ways on LRU Interpretation* (Page 1 of 2)

| LRU Bits | Lock Bits | LRU Way |
|---|---|---|
| 00x | 0xxx | 0 |
| 00x | 10xx | 1 |
| 000 | 110x | 2 |
| 001 | 11x0 | 3 |
| 01x | x0xx | 1 |
| 01x | 01xx | 0 |
| 010 | 110x | 2 |
| 011 | 11x0 | 3 |
| 1x0 | xx0x | 2 |
| 1x0 | xx10 | 3 |
| 100 | 0x11 | 0 |

*Table 9-3. Effect of Locked Ways on LRU Interpretation* (Page 2 of 2)

| LRU Bits | Lock Bits | LRU Way |
|---|---|---|
| 110 | x011 | 1 |
| 1x1 | xxx0 | 3 |
| 1x1 | xx01 | 2 |
| 101 | 0x11 | 0 |
| 111 | x011 | 1 |

*Figure 9-1. L2 Cache*



The execution of the Store Word Conditional Indexed (**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.** instruction 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 cache and the reservation completes.
- If the **stwcx.** hits an unmodified sector in the L2 cache, and either the L1 or L2 cache is in write-through mode, the **stwcx.** is forwarded to the 60x bus interface and the sector hit in the L2 cache is invalidated.

L1 cache-block-push operations generated by the execution of Data Cache Block Flush (**dcbf)** and Data Cache Block Store (**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 a 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.

The L2 flush mechanism is similar to the L1 data-cache flush mechanism. The L2 flush requires that the entire L1 data cache be flushed prior to flushing the L2 cache. Also, interrupts must be disabled during the L2 flush so that the LRU algorithm does not get disturbed. The L2 can be flushed by executing uniquely addressed load instructions to each of the 32-byte blocks of the L2 cache. This requires a load to each of the two sectors in each of the four ways in each of the 4096 sets of the L2 cache. The loads must not hit in the L1 cache in order to effect a flush of the L2 cache.

The Data Cache Block Invalidate (**dcbi)** instruction is always forwarded to the L2 cache and causes a sector invalidation if a hit occurs. The instruction is also forwarded to the 60x bus interface for broadcast if HID0[ABE] is set to 1. The instruction-cache-block invalidate (**icbi**) instruction invalidates only L1-cache blocks and is never forwarded to the L2 cache.

Any Data Cache Block Set To Zero (**dcbz**) instructions that are marked global do not affect the L2 cache state. If an instruction hits in the L1 and L2 caches, the L1 data-cache block is cleared and the instruction completes. If an 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 without reference to the state of the L2 cache.

The Synchronize (**sync)** and Enforce In-Order Execution of I/O (**eieio)** instructions bypass the L2 cache and are forwarded to the 60x bus.

## 9.3 L2 Cache Control Register (L2CR)

The L2 Cache Control Register is used to configure and enable the L2 cache. The L2CR is a supervisor-level read/write, implementation-specific register that is accessed as Special Purpose Register (SPR) 1017. The contents of the L2CR are cleared during power-on reset. For a full description of L2CR and its bits, see *Section 2.1.5, L2 Cache Control Register (L2CR),* on page 81.

## 9.4 L2 Cache Initialization

The L2 cache is initially disabled following a power-on or hard reset. 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.

1. Power-on reset (automatically performed by the assertion of the $\overline{\text{HRESET}}$ signal).
2. Disable interrupts and dynamic power management (DPM).
3. Disable L2 cache by clearing L2CR[L2E].
4. Perform an L2 global invalidate as described in *Section 9.5*.
5. After the 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.

## 9.5 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[GI], and it must be performed only while the L2 cache is disabled.

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

1. Flush the L2 to save any modified data.
2. 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.
3. Initiate the global invalidation operation by setting the L2CR[GI] bit to 1.
4. Monitor the L2CR[IP] bit to determine when the global invalidation operation is complete (indicated by the clearing of L2CR[IP]). The global invalidation requires approximately 32 K core clock cycles to complete.
5. After detecting the clearing of L2CR[IP], clear L2CR[GI] and re-enable the L2 cache for normal operation by setting L2CR[L2E].

Never perform a global invalidation of the L2 cache while in dynamic power-management enable mode. Be sure the HID0[DPM] bit is zero. Also ensure that the processor is in a tight, uninterruptable software loop monitoring the end of the global invalidate, so that an L1 data-cache miss cannot occur that would initiate a reload from system memory during the global invalidate operation.

## 9.6 L2 Cache Used as On-Chip Memory

The L2 cache can be configured to be unlocked, partially locked, or completely locked. When configured to be unlocked, the L2 cache is 4-way set-associative, with 32 bytes per sector, two sectors per block. When configured to be completely locked, the L2 cache is a 1-MB on-chip memory (OCM) that is explicitly managed by software. With one, two, or three ways locked, the locked partition is a software managed OCM, while the unlocked partition is a 3-, 2-, or 1-way (direct mapped) cache that is managed by hardware. The locked cache is considered to be local memory, and so is not kept coherent with main memory.

### 9.6.1 Locking the L2 Cache

Locking of the L2 cache is controlled by the L2CR[LOCK] bits (bits 24:27) as follows:

- 0000 No cache locking
- 1xxx Lock way 0
- x1xx Lock way 1
- xx1x Lock way 2
- xxx1 Lock way 3

**Note:** L2CR[LOCKLO] and L2CR[LOCKHI] can also be used to lock ways 0 and 1, and ways 2 and 3, respectively. These bits are defined in this way to provide a form of backward compatibility with the 750FX design. However, new software should use the L2CR[LOCK] bits to control L2-cache locking.

Any cache line in a locked part of the L2-cache array can be read or written by the processor, but cannot be deallocated for line replacement. The locked L2 cache is intended to be a local memory for the processor, and so should not contain addresses that are accessed outside the processor. However, the L2 controller does snoop the locked ways, and a snoop hit can cause a deallocation. In addition, for the specific case of an **stwcx.** marked write through that hits in a locked line, the line will be invalidated in the L2 cache. In this case, the store will be forwarded to the bus, as is the case for the unlocked L2. Finally, invalid lines in the locked part cannot be allocated by any mechanism.

To lock instructions or data in way 0 of the L2 cache requires the following sequence:

1. Execute a **sync** instruction to allow all load/store activity to complete.
2. Set the data-only bit (L2CR[DO] = 1) to prevent the current instruction stream from being cached in the L2.
3. Flush the L2 to save any modified data.
4. Disable the L2 as usual for invalidation.
5. Invalidate the L2 to prevent collisions with lines to be locked.
6. Lock ways 1 through 3 (L2CR[LOCK] = 0111) so all allocations are in way 0.
7. Enable the L2.
8. Load the contents to be locked (see *Section 9.6.1.1*).
9. Execute a **sync** instruction
10. Lock way 0 and unlock ways 1 through 3 (L2CR[LOCK] = 1000).
11. Reset the data-only bit (L2CR[DO] = 0).

At this point, all data and instructions to be locked are in way 0 of the L2 cache. To lock multiple ways, first *unlock* the ways that are to be locked, and lock all others as described in step 6. Then, *lock* the selected ways and unlock all others as described in step 10.

#### *9.6.1.1 Loading the Locked L2 Cache*

Contents are loaded into the L2 cache simply by executing load instructions to cacheable addresses that miss in the L1. Note that instructions to be locked in the L2 cache are loaded as data. Only one access to each 32-byte cache block is needed to allocate the entire block in the cache.

**Note:** While lines are being allocated in way 0 using this procedure, the cache behaves as a direct-mapped cache. Therefore, the various blocks of code and data must be located in physical memory such that each line is in a unique equivalence class with respect to the organization of the L2 cache. Each way contains 256 KB, consisting of 4096 lines of two 32-byte sectors.

In addition to the constraint imposed by the cache associativity, it is important while loading the contents to be locked that no spurious lines are allocated. The use of the 'data only' mode prevents the current instruction stream from being allocated. To prevent unwanted data from being allocated, the load sequence must either avoid using other data (for example, by using immediate fields in the instructions to specify load addresses) or must prevent that data from colliding with other data or instructions to be allocated (for example, by allocating it explicitly as part of the data to be locked).

The contents of the locked cache cannot be deallocated implicitly under normal conditions (see the exceptions for snoops and **stwcx.** marked write through described above), but can be deallocated by the processor using a **dcbi** instruction, as described below.

#### *9.6.1.2 Locked Cache Operation*

When one or more ways of the L2 cache are locked, the locked ways behave like the normal (unlocked) cache except in the following situations:

> Replacement never occurs in the locked cache. If one way is locked, an L2 miss causes replacement in one of the unlocked ways of the cache by the new block of data or instructions received from the bus (or from the L1 cache in the case of a castout). If all ways are locked, an L2 miss causes the new block of data or instructions from the bus to be forwarded to the L1 cache without updating the L2. In the case of a castout that misses in a completely locked L2, the data is forwarded to the bus from the L1 without updating the L2. Although the locked cache is not kept coherent with main memory in general, a store access marked write-through that hits in a locked way updates the L2 as usual and is also forwarded to the bus.

An **stwcx.** marked cache-inhibited that hits in a locked way invalidates the corresponding cache line, and also gets forwarded to the 60x bus. The locked cache is snooped and responds identically to an unlocked cache, which might also result in the invalidation of locked cache lines. The L2CR[SHEE] bit can be set to enable such an event to raise a machine check. The L2CR[SHERR] bit is a sticky bit that records the occurrence of an invalidation in a locked line. This bit can be cleared by a Move-to Special-Purpose Register (**mtspr**) instruction to the L2CR. Note that Load Word and Reserve Indexed (**lwarx**) and **stwcx.** instructions should not be used with locked cache addresses when trying to synchronize outside the processor, since the locked memory is not shared externally. More generally, coherency is not maintained for locked addresses. Any updates to locked memory that must be reflected outside the processor must be made through software, by cache-inhibited operations or copies to other addresses.

A **dcbst** instruction is used to update external memory with the more recent contents of the internal cache. A **dcbf** instruction also does this, while invalidating the internal copy. A **dcbf** or **dcbst** that hits in the locked cache modifies the locked cache with the castout block if it hit in the L1 cache, but does nothing if it missed in the L1 cache (the corresponding behavior for a hit in the normal L2 cache is to invalidate the block if the access hit in the L1, and to flush the block if it missed in the L1).

The **dcbz** instruction has no effect on the L2-cache state, whether the state is locked or not. The **dcbi** instruction causes invalidation of the block in the case of an L2 hit, for both normal and locked caches.

## 9.7 Data-Only and Instruction-Only Modes

The 750GX microprocessor supports a data-only mode of L2 operation that can be used for test (as described in *Section 9.8.2* on page 333) or for specific applications that might perform better when the L2 is used to store only data. This mode is selected by setting the L2CR[DO] bit to 1. In L2 data-only mode, all requests from the L1 instruction cache are treated as cache-inhibited, and so bypass the L2 and are forwarded to the external bus. Once the L2CR[DO] bit is set, instructions currently in the L2 cache are not accessible. Over time, cache lines containing instructions will be replaced with those containing data.

Similarly, the 750GX microprocessor supports an instruction-only mode of L2 operation, selected by setting the L2CR[IO] bit to 1. In this mode, all requests from the L1 data cache are treated as cache inhibited by the L2 cache. When L2CR[IO] is set, any L2-cache lines containing data will be replaced over time with those containing instructions, until, in steady state, the L2 cache contains only instructions.

## 9.8 L2 Cache Test Features and Methods

In the course of system power-up, testing might be required to verify the proper operation of the L2 tag memory, SRAM, and overall L2-cache system. The following sections describe the 750GX’s features and methods for testing the L2 cache. The L2-cache address space should be marked as guarded (G = 1) so spurious load operations are not forwarded to the 60x bus interface before branch resolution during L2-cache testing.

### 9.8.1 L2CR Support for L2 Cache Testing

L2CR[DO] and L2CR[TS] support the testing of the L2 cache. L2CR[DO] prevents instructions from being cached in the L2. This allows the L1 instruction cache to remain enabled during the testing process without having L1 instruction misses affect the contents of the L2 cache. It also allows all L2-cache activity to be controlled by program-specified load-and-store operations.

L2CR[TS] is used with the **dcbf** and **dcbst** instructions to push data into the L2 cache. When L2CR[TS] is set, and the L1 data cache is enabled, an instruction loop containing a **dcbf** instruction can be used to store any address or data pattern to the L2 cache. Additionally, 60x bus broadcasting is inhibited when a **dcbz** instruction is executed. This allows the use of a **dcbz** instruction to clear an L1-cache block, followed by a **dcbf** instruction to push the cache block into the L2 cache and invalidate the L1-cache block.

When the L2 cache is enabled, cacheable single-beat read operations are allowed to hit in the L2 cache, and cacheable write operations are allowed to modify the contents of the L2 cache when a hit occurs. Cacheable single-beat reads and writes occur when address translation is disabled, which invokes the use of the default WIMG bits (0011). They also occur when address translation is enabled and accesses are marked as cacheable through the page table entries or the Block Address Translation (BAT) Registers, and the L1 data cache is disabled or locked. When the L2 cache has been initialized and the L1 cache has been disabled or locked, load or store instructions then bypass the L1 cache and hit in the L2 cache directly. When L2CR[TS] is set, cacheable single-beat writes are inhibited from accessing the 60x bus interface after an L2-cache miss.

During L2-cache testing, the performance monitor can be used to count L2-cache hits and misses, thereby providing a numerical signature for test routines and a way to verify proper L2-cache operation.

### 9.8.2 L2 Cache Testing

A typical test for verifying the proper operation of the 750GX microprocessor's L2-cache memory follows this sequence:

1. Initialize the L2 test sequence by disabling address translation to invoke the default WIMG setting (0011). Set L2CR[DO] and L2CR[TS], 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.
2. Test the L2-cache SRAM by enabling the L1 data cache and executing a sequence of **dcbz**, store word (**stw**), and **dcbf** instructions to initialize the L2 cache with a desired range of consecutive addresses and with cache data consisting of zeros. Once the L2 cache holds a sequential range of addresses, disable the L1 data cache and execute a series of single-beat load-and-store operations employing a variety of bit patterns to test for stuck bits and pattern sensitivities in the L2-cache SRAM. The performance monitor can be used to verify whether the number of L2-cache hits or misses corresponds to the tests performed.
3. Test the L2-cache tag memory by enabling the L1 data cache and executing a sequence of **dcbz**, **stw**, and **dcbf** instructions to initialize the L2 cache with a wide range of addresses and cache data. Once the L2 cache is populated with a known range of addresses and data, disable the L1 data cache and execute a series of store operations to addresses not previously in the L2 cache. These store operations should miss in every case. Note that setting L2CR[TS] inhibits L2-cache misses from being forwarded to the 60x bus interface, thereby avoiding the potential for bus errors due to addressing hardware or nonexistent memory. The L2 cache then can be further verified by reading the previously loaded addresses and observing whether all the tags hit, and that the associated data compares correctly. The performance monitor can also be used to verify whether the proper number of L2-cache hits and misses correspond to the test operations performed.
4. The entire L2 cache can be tested by clearing L2CR[DO] and L2CR[TS], 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.

## 9.9 L2 Cache Timing

Loading the L2-cache SRAM can occur from the store data queue (which includes single beat stores and L1 castouts), or from the 2-entry, L2 reload data queue. When data is available in either queue, arbitration for the L2 cache takes place. The requests for the L2 cache, in prioritized order, include a snoop request, an L2 castout, the store data queue, a lookup request, or an L2 reload. Loads always take four beats, starting in the cycle in which a request is granted. A double word of data with the ECC correction bits is written with each beat, filling a 32-byte cache line. The arbitration phase and data phase are pipelined, allowing new arbitration during a previous data phase.

L1 misses that hit in the L2 cache will incur a 5-cycle latency for the critical word returned to the L1. This latency includes one cycle for ECC correction. The L2-cache read data path is 256 bits, which loads the L1 data-cache reload buffer in one cycle, at the same time forwarding the critical word to the load/store unit. Instruction-cache misses, however, will be serviced in four beats from the L2 with the critical word first.

# 10. Power and Thermal Management

The 750GX microprocessor is specifically designed for low-power operation. It provides both automatic and program-controlled power reduction modes for progressive reduction of power consumption. It also provides a thermal assist unit (TAU) to allow on-chip thermal measurement, allowing sophisticated thermal management for high-performance portable systems. This chapter describes the hardware support provided by the 750GX for power and thermal management.

## 10.1 Dynamic Power Management

Dynamic power management (DPM) automatically powers up and down the individual execution units of the 750GX, based upon the contents of the instruction stream. For example, if no floating-point instructions are being executed, the floating-point unit is automatically powered down. Power is not actually removed from the execution unit; instead, each execution unit has an independent clock input, which is automatically controlled on a clock-by-clock basis. Since complementary metal-oxide semiconductor (CMOS) circuits consume negligible power when they are not switching, stopping the clock to an execution unit effectively eliminates its power consumption. The operation of DPM is completely transparent to software or any external hardware. Dynamic power management is enabled by setting the DPM bit in Hardware-Implementation-Dependent Register 0 (HID0[DPM] = 1).

## 10.2 Programmable Power Modes

The 750GX provides four programmable power modes—full on, doze, nap, and sleep. Software selects these modes by setting one (and only one) of the three power saving mode bits in the HID0 Register.

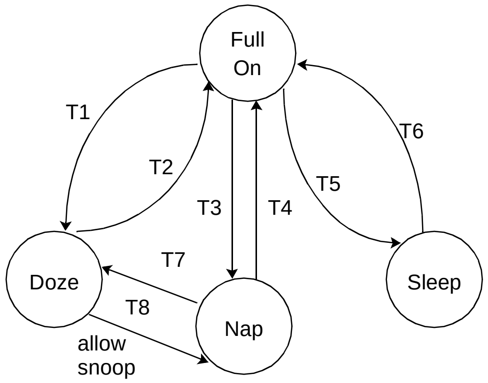
Hardware can enable a power management state through external asynchronous interrupts. Such a hardware interrupt causes the transfer of program flow to interrupt handler code, which then invokes the appropriate power saving mode. The 750GX also contains a decrementer, which allows it to enter the nap or doze mode for a predetermined amount of time and then return to full power operation through a decrementer interrupt.

**Note:** The 750GX cannot switch from one power management mode to another without first returning to full-on mode.

The sleep mode disables bus snooping. Therefore, a hardware handshake is provided to ensure coherency before the 750GX enters this power management mode.

These power states and power saving modes are shown *Figure 10-1, 750GX Power States* and *Table 10-1* on page 336 summarizes the four power modes.

*Figure 10-1. 750GX Power States*



T1: HID0(Doze) = 1 and MSR(POW) 0 → 1
T2: HRESET, SRESET, INT, SMI, MCP, DEC, PFM, machine-check interrupts, thermal-management interrupt
T3: HID0(Nap) = 1 and MSR(POW) 0 → 1
T4:HRESET, SRESET, INT, SMI, MCP, DEC
T5: HID0(Sleep) = 1 and MSR(POW) 0 → 1
T6: HRESET, SRESET, INT, SMI, MCP
T7: QACK 0 → 1
T8: QACK 1 → 0

*Table 10-1. 750GX Microprocessor Programmable Power Modes*

| Power Management Mode | Functioning Units | Activation Method | Full-Power Wake Up Method |
|---|---|---|---|
| Full on | All units active | — | — |
| Doze | • Bus snooping<br>• Data cache as needed<br>• Decrementer timer | Controlled by software | External asynchronous exceptions[1]<br>Decrementer interrupt<br>Performance-monitor interrupt<br>Thermal-management interrupt<br>Hard or soft reset |
| Nap | • Bus snooping (enabled by deassertion of QACK)<br>• Decrementer timer | Controlled by hardware and software | External asynchronous exceptions[1]<br>Decrementer interrupt<br>Hard or soft reset |
| Sleep | None | Controlled by hardware and software | External asynchronous exceptions*<br>Hard or soft reset |

1. Exceptions are referred to as interrupts in the architecture specification.

### 10.2.1 Power Management Modes

The following sections describe the characteristics of the 750GX’s power management modes, the requirements for entering and exiting the various modes, and the system capabilities provided by the 750GX while the power management modes are active.

A power saving mode is activated by setting the power management enable bit in the Machine State Register (MSR[POW]) and one of the three HID0 power saving mode bits which are listed in *Table 10-2*.

*Table 10-2. HID0 Power Saving Mode Bit Settings*

| HID0 Bits | Power Saving Mode |
|---|---|
| 8 | Low-power doze |
| 9 | Low-power nap |
| 10 | Low-power sleep |

#### *10.2.1.1 Full On Mode*

Full on mode is selected when the POW bit in MSR is cleared. This is the default state following initialization of a hard reset (HRESET). The 750GX is fully powered, and all functional units are operating at full processor speed at all times.

#### *10.2.1.2 Doze Mode*

Doze mode disables most functional units but maintains cache coherency by enabling the bus interface unit and snooping. A snoop hit causes the 750GX to enable the data cache, copy the data back to memory, disable the cache, and fully return to the doze state. Doze mode can be summarized as follows:

- Most functional units are disabled.
- Data cache, L2 cache, bus snooping logic, and the time base/decrementer are still enabled.
- Doze mode is enabled with the following sequence:
  1. Set the doze bit (HID0[8] = 1); clear the nap and sleep bits (HID0[9] and HID0[10] = 0).
  2. 750GX enters doze mode after several processor clocks.
- Several methods of returning to full-on mode:
  – Assert INT, MCP, SMI, decrementer, performance-monitor, or thermal-management interrupts
  – Assert hard reset or soft reset.
- Transition to full-power state takes no more than a few processor cycles.
- Phase-locked loop (PLL) is required to be running and locked to the system clock (SYSCLK).

#### *10.2.1.3 Nap Mode*

The nap mode disables the 750GX but still maintains the PLL and the time base/decrementer. The time base can be used to restore the 750GX to full-power state after a programmed amount of time. To maintain data coherency, bus snooping is disabled for nap and sleep modes through a hardware handshake sequence using the quiesce request (QREQ) and quiesce acknowledge (QACK) signals. The 750GX asserts the QREQ signal to indicate that it is ready to disable bus snooping. When the system has ensured that snooping is no longer necessary, it will assert QACK and the 750GX will enter the nap mode. If the system determines that a bus snoop cycle is required, QACK is deasserted to the 750GX for at least eight bus clock cycles, and the

750GX will then be able respond to a snoop cycle. Assertion of QACK following the snoop cycle will again disable the 750GX's snoop capability. The 750GX's power dissipation while in nap mode with QACK deasserted is the same as the power dissipation while in doze mode.

The 750GX also allows dynamic switching between nap and doze modes to allow the use of nap mode without sacrificing hardware snoop coherency. For this operation, negating QACK at any time for at least eight bus cycles guarantees that the 750GX has transitioned from nap mode to doze mode in order to snoop. Reasserting QACK then allows the 750GX to return to nap mode. This sequencing could be used by the system at any time with knowledge of what power management mode the 750GX is in currently, if any. Nap mode can be summarized as follows:

- Time base/decrementer still enabled.
- Thermal-management unit enabled.
- Most functional units disabled.
- All nonessential input receivers disabled.
- Nap mode is enabled with the following sequence:
  1. Set nap bit (HID0[9] = 1); clear doze and sleep bits (HID0[8] and HID0[10] = 0).
  2. 750GX asserts quiesce request (QREQ) signal.
  3. System asserts quiesce acknowledge (QACK) signal.
  4. 750GX enters nap mode after several processor clocks.
- Nap mode bus snoop sequence:
  1. System deasserts QACK signal for eight or more bus clock cycles.
  2. 750GX snoops address tenures on the bus.
  3. System asserts QACK signal to restore full nap mode.
- Several methods of returning to full-power mode:
  – Assert INT, MCP, SMI, or decrementer interrupts.
  – Assert hard reset or soft reset.
- Transition to full-power takes no more than a few processor cycles.
- PLL running and locked to SYSCLK.

### *10.2.1.4 Sleep Mode*

Sleep mode consumes the least amount of power of the four modes since all functional units are disabled. To conserve the maximum amount of power, the PLL can be disabled by placing the PLL_CFG signals in the PLL bypass mode, and disabling SYSCLK.

**Note:** Forcing the SYSCLK signal into a static state does not disable the 750GX’s PLL, which will continue to operate internally at an undefined frequency unless placed in PLL bypass mode.

Due to the fully static design of the 750GX, internal processor state is preserved when no internal clock is present. Because the time base and decrementer are disabled while the 750GX is in sleep mode, the 750GX’s time-base contents will have to be updated from an external time base after exiting sleep mode if maintaining an accurate time-of-day is required. Before entering the sleep mode, the 750GX asserts the $\overline{\text{QREQ}}$ signal to indicate that it is ready to disable bus snooping.

When the system has ensured that snooping is no longer necessary, it asserts $\overline{\text{QACK}}$ and the 750GX will enter sleep mode. Sleep mode can be summarized as follows:

- All functional units disabled (including bus snooping and time base/decrementer).
- All nonessential input receivers disabled.
  - Internal clock regenerators disabled.
  - PLL still running (see below).
- Sleep mode is enabled with the following sequence:
  1. Set sleep bit (HID0[10] = 1); clear doze and nap bits (HID0[8] and HID0[9]).
  2. 750GX asserts quiesce request ($\overline{\text{QREQ}}$).
  3. System asserts quiesce acknowledge ($\overline{\text{QACK}}$).
  4. 750GX enters sleep mode after several processor clocks.
- Several methods of returning to full-on mode:
  - Assert $\overline{\text{INT}}$, $\overline{\text{SMI}}$, or $\overline{\text{MCP}}$ interrupts.
  - Assert hard reset or soft reset.
- PLL can be disabled and SYSCLK can be removed while in sleep mode.
- Return to full-on mode after PLL and SYSCLK are disabled in sleep mode:
  - Enable SYSCLK.
  - Reconfigure PLL into desired processor clock mode.
  - System logic waits for PLL start-up and relock time (100 µs).
  - System logic asserts one of the sleep recovery signals (for example, $\overline{\text{INT}}$).

### *10.2.1.5 Dynamic Power Reduction*

The 750GX functional units will go into a low power mode automatically if the unit is idle. This mode will not affect operational performance and is entered when the DPM bit is enabled in HID0 (HID0 bit 11). This operation is transparent to software or any external hardware.

### 10.2.2 Power Management Software Considerations

Since the 750GX is a dual-issue processor with out-of-order execution capability, care must be taken in how the power management mode is entered. Furthermore, nap and sleep modes require all outstanding bus operations to be completed before these power management modes are entered. Normally, during system configuration time, one of the power management modes would be selected by setting the appropriate HID0 mode bit. Later on, the power management mode is invoked by setting the MSR[POW] bit. To ensure a clean transition into and out of a power management mode, the Move-to Machine State Register (**mtmsr)** (POW) instruction should be preceded by a Synchronize (**sync**) instruction and followed by an Instruction Synchronize (**isync**) as shown below.

```
...
loop:
sync
mtmsr (POW)
isync
br loop
...
```

## 10.3 750GX Dual PLL Feature

### 10.3.1 Overview

Due to the relationship of power to frequency and voltage (power is proportional to frequency and a square of voltage), running the processor at a lower frequency and associated lower voltage can result in significant power savings. The 750GX design includes two PLLs (PLL0 and PLL1), which allows the processor clock frequency to be dynamically changed to one of the PLL frequencies via software control. The HID1 Register (described in *Section 2.1.2.3* on page 70) contains fields that specify the frequency range of each PLL, the clock multiplier for each PLL, external or internal control of PLL0, and a bit to choose which PLL is selected (that is, which is the source of the processor clock at any given time). In addition, the supplied processor voltage ($V_{DD}$) can be varied to support the selected frequency: lower voltage, lower frequency, and lower power for normal processing tasks or higher voltage, higher frequency for situations requiring high performance. PLL voltages ($AV_{DD}$) should remain constant at all times.

At power-on reset, the HID1 Register contains zeros for all the non-read-only bits (bits 7 to 31). This configuration corresponds to the selection of PLL0 as the source of the processor clocks, and selects the external configuration and range pins to control PLL0 (see *Chapter 8, Bus Interface Operation,* on page 279). The external configuration and range pin values are accessible to software via HID1 read-only bits 0-6. PLL1 is always controlled by its internal configuration and range bits. The HID1 setting associated with a hard reset corresponds to a PLL1 configuration of clock off, and the selection of the medium frequency range.

As stated in the *PowerPC 750GX RISC Microprocessor Datasheet*, $\overline{\text{HRESET}}$ must be asserted during power up long enough for the PLLs to lock and for the internal hardware to be reset. Once this timing is satisfied, $\overline{\text{HRESET}}$ can be negated. The processor will now proceed to execute instructions, clocked by PLL0 as configured via the external pins. The processor clock frequency can be modified from this initial setting in one of two ways. First, as with earlier designs, $\overline{\text{HRESET}}$ can be asserted, and the external configuration pins can be set to a new value. The machine state is lost in this process, and, as always, $\overline{\text{HRESET}}$ must be held asserted while the PLL relocks, and the internal state is reset. Second, the introduction of another PLL provides an alternative means of changing the processor clock frequency, which does not involve the loss of machine state, nor a delay for PLL relock.

**Note:** If the PLL software configuration is used, sufficient time must be allowed for the chosen PLL to lock. See the *PowerPC 750GX RISC Microprocessor Datasheet* for more information.

The following sequence can be used to change processor clock frequency. Assume PLL0 is currently the source for the processor clock. The first step is to configure PLL1 to produce the desired clock frequency, by setting HID1[PR1] and HID1[PC1] to the appropriate values. Next, wait for PLL1 to lock. The lock time is the same for both PLLs and is provided in the *PowerPC 750GX RISC Microprocessor Datasheet*. Finally, set HID1[PS] to a 1 to initiate the transition from PLL0 to PLL1 as the source of the processor clocks. From the time the HID1 Register is updated to select the new PLL, the transition to the new clock frequency will complete within three bus cycles. After the transition, the HID1(PSTAT1) bit indicates which PLL is in use.

Once both PLLs are running and locked, the processor frequency can be toggled with very low latency. For instance, when it is time to change back to the PLL0 frequency, there is no need to wait for PLL lock. HID1[PS] can be reset to 0, causing the processor clock source to transition from PLL1 back to PLL0. If PLL0 will not be needed for some time, it can be configured to be off while not in use. This is done by resetting the HID1[PC0] field to 0, and setting HID1[PI0] to 1. Turning the nonselected PLL off results in a modest power savings, but introduces added latency when changing frequency. If PLL0 is configured to be off, the procedure for switching to PLL0 as the selected PLL involves changing the configuration and range bits, waiting for lock, and then selecting PLL0, as described in the previous paragraph.

The following are hazards that must be avoided in reconfiguring the PLLs:

- The configuration and range bits in HID1 should only be modified for the nonselected PLL, since it will require time to lock before it can be used as the source for the processor clock.
- The HID1[PI0] bit should only be modified when PLL0 is not selected.
- Whenever one of the PLLs is reconfigured, it must not be selected as the active PLL until enough time has elapsed for the PLL to lock.
- At all times, the frequency of the processor clock, as determined by the various configuration settings, must be within the specification range for the current operating conditions. In particular, in systems where $V_{DD}$ can be varied to achieve additional power efficiency, a transition from low frequency to high frequency requires that $V_{DD}$ is at a sufficiently high level to support the higher frequency.
- Never select a PLL that is in the “off” configuration.

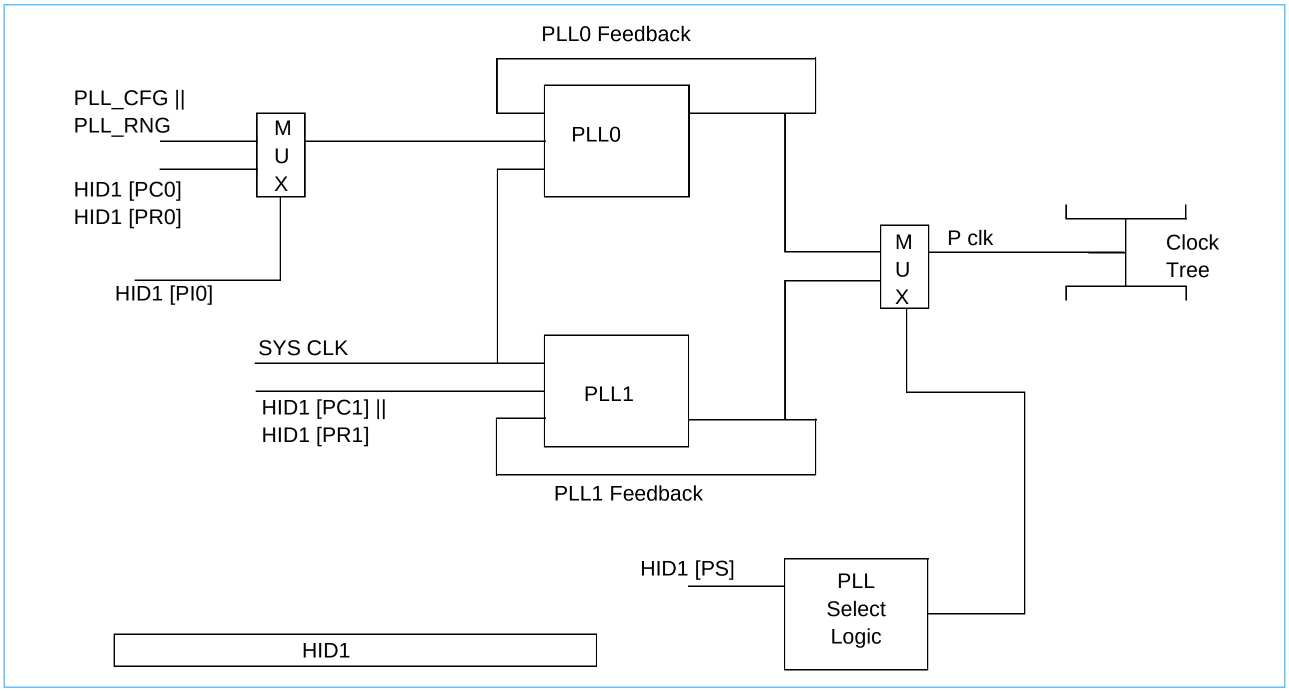
### 10.3.2 Configuration Restriction on Frequency Transitions

It is considered a programming error to switch from one PLL to the other when both are configured in a half-cycle multiplier mode. For example, with PLL0 configured in 9:2 mode (PLL_CFG[0:4] = '01001') and PLL1 configured in 13:2 mode (PLL_CFG[0:4] = '01101'), changing the select bit (HID1[PS]) is not allowed. In cases where such a pairing of configurations is desired, an intermediate full-cycle configuration must be used between the two half-cycle modes. For example, with PLL0 at 9:2, PLL1 configured at 6:1 is selected. Then PLL0 is reconfigured at 13:2, locked, and selected. For more information about hardware-implementation-dependent bit functions for HID1, see *Section 2.1.2.3* on page 70.

### 10.3.3 Dual PLL Implementation

Switching between the two PLLs on the 750GX is intended to be a seamless, 3-cycle operation. As shown in *Figure 10-2*, the two PLL outputs will feed a multiplexer (MUX), controlled by a signal from the PLL select logic.

*Figure 10-2. Dual PLL Block Diagram*



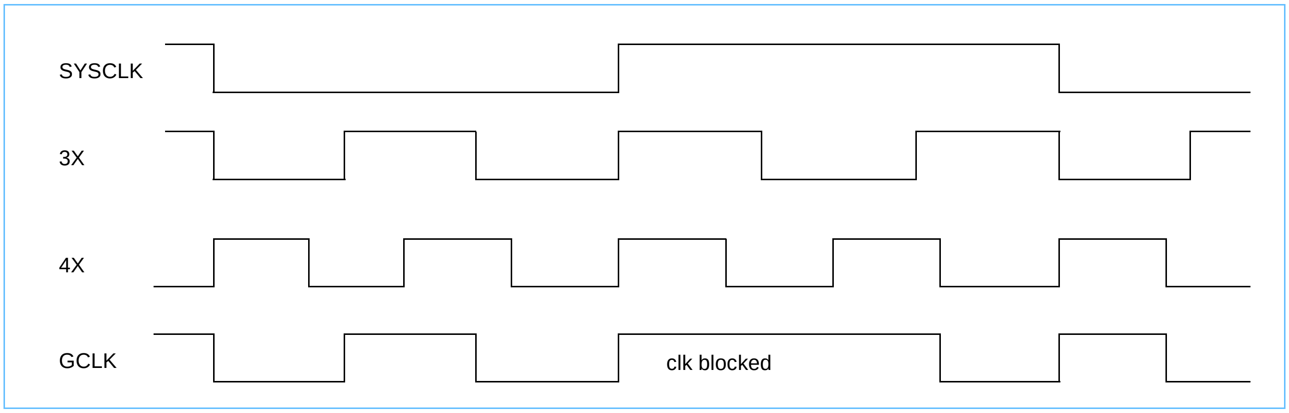
Each PLL will use as a feedback path, a clock regeneration path that is a copy of a typical path in the actual clock tree. Since both PLLs will be generating outputs that are integral or half integral multiples of the SYSCLK frequency, all three clocks (SYSCLK, PLL0, and PLL1 out) will have a rising edge on (at least) every other rising edge of SYSCLK. When the two PLLs are both configured for half integral multiples of SYSCLK, they cannot have a common rising edge. This leads to the restriction that switching between half cycle settings is not allowed.

In the case where at least one PLL is configured for an integral multiple of SYSCLK, all three clocks will have a common rising edge. In the absence of skew between the two PLL outputs, the MUX control signal could be changed just before, or just after, that common rising edge to achieve seamless switching. The PLL select logic in *Figure 10-2* represents the logic needed to generate the MUX control signal.

When HID1 is written to switch from one to the other PLL, the control logic waits for the rising edges of both PLLs to line up with the rising edge of SYSCLK. When both the PLL0 and PLL1 clocks are high, the MUX control signal is switched. If the bus/core ratio of the PLL being switched to is greater than 2.5x, one clock pulse will be blocked. This provides seamless functionality regardless of any skew between the PLLs, including snoop requests that could come in during a PLL switch operation. Timing of the switch signal is critical to ensure that there are no glitches or short clocks distributed to the logic.

There is also fence logic between the HID1 Register and the PLL and associated control logic to allow reset functionality and to prevent the PLLs from becoming corrupted by a scan operation. This requirement allows an operation such as a RISCWatch Long Shift Register Latch (LSRL) scan to occur without corrupting the clocks. See *Figure 10-3* on page 343.

*Figure 10-3. Dual PLL Switching Example, 3X to 4X*



## 10.4 Thermal Assist Unit

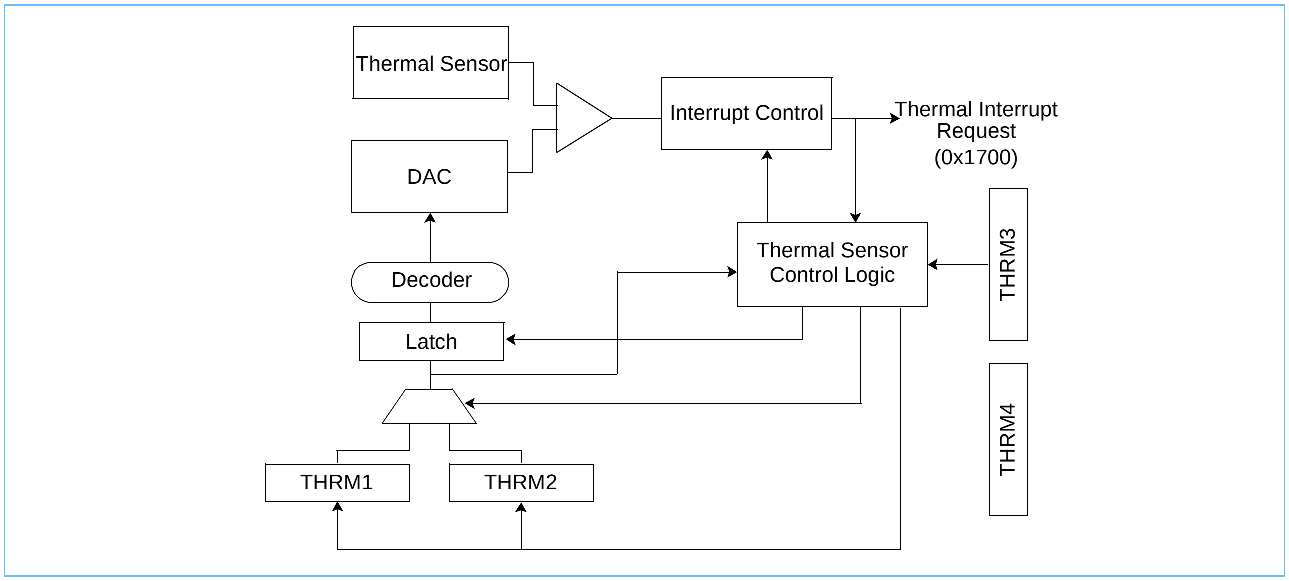
With the increasing power dissipation of high-performance processors and operating conditions that span a wider range of temperatures than desktop systems, thermal management becomes an essential part of system design to ensure reliable operation of portable systems. One key aspect of thermal management is ensuring that the junction temperature of the microprocessor does not exceed the operating specification. While the case temperature can be measured with an external thermal sensor, the thermal constant from the junction to the case can be large, and accuracy can be a problem. This might lead to lower overall system performance due to the necessary compensation to alleviate measurement deficiencies.

The 750GX provides the system designer an efficient means of monitoring junction temperature through the incorporation of an on-chip thermal sensor and programmable control logic to enable a thermal-management implementation tightly coupled to the processor for improved performance and reliability.

### 10.4.1 Thermal Assist Unit Overview

The on-chip thermal assist unit (TAU) is composed of a thermal sensor, a digital-to-analog converter (DAC), a comparator, control logic, and four dedicated Special Purpose Registers (SPRs). See *Figure 10-4* on page 344 for a block diagram of the TAU.

*Figure 10-4. Thermal Assist Unit Block Diagram*



The TAU provides thermal control by periodically comparing the 750GX’s junction temperature against user-programmed thresholds, and generating a thermal-management interrupt if the threshold values are crossed. The TAU also enables the user to determine the junction temperature through a software successive approximation routine.

The TAU is controlled through four supervisor-level SPRs, accessed through the Move-to Special Purpose Register (**mtspr**) and Move-from Special Purpose Register (**mfspr**) instructions. Two of the SPRs (THRM1 and THRM2) provide temperature threshold values that can be compared to the junction temperature value, and control bits that enable comparison and thermal interrupt generation. The third SPR (THRM3) provides a TAU enable bit and a sample interval timer. To enhance accuracy, THRM4 provides the temperature offset measured and burned into the THRM4 Register at the factory.

Note that all the bits in THRM1, THRM2, and THRM3 are cleared to 0 during a hard reset. THRM4 always contains the fused offset value determined at the factory. The TAU remains idle and in a low-power state until configured and enabled.

The bit fields of THRM1 and THRM2 are described in *Section 2.1.4.1, Thermal-Management Registers 1–2 (THRM1–THRM2),* on page 78. The bit fields of THRM3 are described *Section 2.1.4.2, Thermal-Management Register 3 (THRM3),* on page 79. The bit fields of THRM4 are described on *Section 2.1.4.3, Thermal-Management Register 4 (THRM4),* on page 80.

### 10.4.2 Thermal Assist Unit Operation

The TAU can be programmed to operate in single-threshold or dual-threshold modes, which results in the TAU generating a thermal-management interrupt when one or both threshold values are crossed. In addition, with the appropriate software routine, the TAU can also directly determine the junction temperature. The following sections describe the configuration of the TAU to support these modes of operation.

#### *10.4.2.1 TAU Single-Threshold Mode*

When the TAU is configured for single-threshold mode, either THRM1 or THRM2 can be used to contain the threshold value, and a thermal-management interrupt is generated when the threshold value is crossed. To configure the TAU for single-threshold operation, set the desired temperature threshold, the thermal-management interrupt direction (TID), thermal-management interrupt enable (TIE), and SPR valid (V) bits for either THRM1 or THRM2. The unused THRM*n* threshold SPR should be disabled by clearing the V bit to 0. In this discussion, THRM*n* refers to the THRM threshold SPR (THRM1 or THRM2) selected to contain the active threshold value.

After setting the desired operational parameters, the TAU is enabled by setting the THRM3[E] bit to 1, and placing a value that allows a sample interval of 20 microseconds or greater in the THRM3[SITV] field. The THRM3[SITV] setting determines the number of processor clock cycles between input to the DAC and sampling of the comparator output. Accordingly, the use of a value smaller than recommended in the THRM3[SITV] field can cause inaccuracies in the sensed temperature.

If the junction temperature does not cross the programmed threshold, the thermal-management interrupt bit (THRM*n*[TIN]) is cleared to 0 to indicate that no interrupt is required, and the thermal-management interrupt valid bit (THRM*n*[TIV]) is set to 1 to indicate that the TIN bit state is valid. If the threshold value has been crossed, the THRM*n*[TIN] and THRM*n*[TIV] bits are set to 1, and a thermal-management interrupt is generated if both the THRM*n*[TIE] and MSR[EE] bits are set to 1.

A thermal-management interrupt is held asserted internally until recognized by the 750GX's interrupt unit. Once a thermal-management interrupt is recognized, further temperature sampling is suspended, and the THRM*n*[TIN] and THRM*n*[TIV] values are held until an **mtspr** instruction is executed to THRM*n*.

The execution of an **mtspr** instruction to THRM*n* anytime during TAU operation will clear the THRM*n*[TIV] bit to 0 and restart the temperature comparison. Executing an **mtspr** instruction to THRM3 will clear both THRM1[TIV] and THRM2[TIV] bits to 0, and restart temperature comparison in THRM*n* if the THRM3[E] bit is set to 1.

Examples of valid THRM1 and THRM2 bit settings are shown in *Table 10-3*.

*Table 10-3. Valid THRM1 and THRM2 Bit Settings* (Page 1 of 2)

| TIN[1] | TIV[1] | TID | TIE | V | Description |
|---|---|---|---|---|---|
| x | x | x | x | 0 | The threshold in the SPR will not be used for comparison. |
| x | x | x | 0 | 1 | Threshold is used for comparison; thermal-management interrupt assertion is disabled. |
| x | x | 0 | 0 | 1 | Set TIN, and do not assert thermal-management interrupt if the junction temperature exceeds the threshold. |
| x | x | 0 | 1 | 1 | Set TIN, and assert thermal-management interrupt if the junction temperature exceeds the threshold. |
| x | x | 1 | 0 | 1 | Set TIN, and do not assert thermal-management interrupt if the junction temperature is less than the threshold. |
| x | x | 1 | 1 | 1 | Set TIN, and assert thermal-management interrupt if the junction temperature is less than the threshold. |
| x | 0 | x | x | 1 | The state of the TIN bit is not valid. |

**Note:**

1. The TIN and TIV bits are read-only status bits.

*Table 10-3. Valid THRM1 and THRM2 Bit Settings* (Page 2 of 2)

| TIN[1] | TIV[1] | TID | TIE | V | Description |
|---|---|---|---|---|---|
| 0 | 1 | 0 | x | 1 | The junction temperature is less than the threshold, and, as a result, the thermal-management interrupt is not generated for TIE = 1. |
| 1 | 1 | 0 | x | 1 | The junction temperature is greater than the threshold, and, as a result, the thermal-management interrupt is generated if TIE = 1. |
| 0 | 1 | 1 | x | 1 | The junction temperature is greater than the threshold, and, as a result, the thermal-management interrupt is not generated for TIE = 1. |
| 1 | 1 | 1 | x | 1 | The junction temperature is less than the threshold, and, as a result, the thermal-management interrupt is generated if TIE = 1. |

**Note:**

1. The TIN and TIV bits are read-only status bits.

### *10.4.2.2 TAU Dual-Threshold Mode*

The configuration and operation of the TAU's dual-threshold mode is similar to single-threshold mode, except both THRM1 and THRM2 are configured with the desired threshold and TID values, and the TIE and V bits are set to 1. When the THRM3[E] bit is set to 1 to enable temperature measurement and comparison, the first comparison is made with THRM1. If no thermal-management interrupt results from the comparison, the number of processor cycles specified in THRM3[SITV] elapses, and the next comparison is made with THRM2. If no thermal-management interrupt results from the THRM2 comparison, the time specified by THRM3[SITV] again elapses, and the comparison returns to THRM1.

This sequence of comparisons continues until a thermal-management interrupt occurs, or the TAU is disabled. When a comparison results in an interrupt, the comparison with the threshold SPR causing the interrupt is halted, but comparisons continue with the other threshold SPR. Following a thermal-management interrupt, the interrupt service routine must read both THRM1 and THRM2 to determine which threshold was crossed. Note that it is possible for both threshold values to have been crossed, in which case the TAU ceases making temperature comparisons until an **mtspr** instruction is executed to one or both of the threshold SPRs.

### *10.4.2.3 750GX Junction Temperature Determination*

While the 750GX's TAU does not implement an analog-to-digital converter to enable the direct determination of the junction temperature, system software can execute a simple successive approximation routine to find the junction temperature.

The TAU configuration used to approximate the junction temperature is the same required for single-threshold mode, except that the threshold SPR selected has its TIE bit cleared to 0 to disable thermal-management interrupt generation. Once the TAU is enabled, the successive approximation routine loads a threshold value into the active threshold SPR, and then continuously polls the threshold SPRs TIV bit until it is set to 1, indicating a valid TIN bit. The successive approximation routine can then evaluate the TIN bit value, and then increment or decrement the threshold value for another comparison. This process is continued until the junction temperature is determined.

#### *10.4.2.4 Power Saving Modes and TAU Operation*

The static power saving modes provided by the 750GX (the nap, doze, and sleep modes) allow the temperature of the processor to be lowered quickly, and can be invoked through the use of the TAU and associated thermal-management interrupt. The TAU remains operational in the nap and doze modes, and in sleep mode as long as the SYSCLK signal input remains active. If the SYSCLK signal is made static when sleep mode is invoked, the TAU is rendered inactive. If the 750GX is entering sleep mode with SYSCLK disabled, the TAU should be configured to disable thermal-management interrupts, to avoid an unwanted thermal-management interrupt when the SYSCLK input signal is restored.

TAU Calibration Offset

Due to process and thermal sensor variations, a temperature offset is provided and can be read via an **mfspr** instruction to THRM4. The TOFFSET field is an 8-bit signed integer that represents the temperature offset measured, and it is burned into the THRM4 Register at test to allow for enhanced accuracy. When in TAU single- or dual-threshold mode, TOFFSET should be subtracted from the desired temperature before setting the THRM*n*(THRESHOLD) field. In junction temperature determination mode, TOFFSET must be added to the final threshold number to determine the temperature.

The temperature, in °C, equals:

THRM*n*[THRESHOLD] + sign-extended [TOFFSET]

## 10.5 Instruction-Cache Throttling

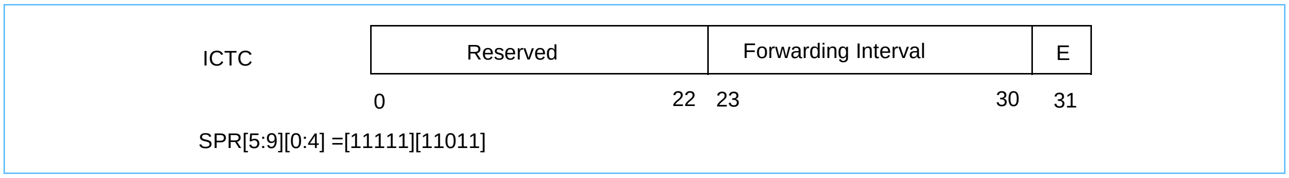
The 750GX provides an instruction-cache throttling mechanism to effectively reduce the instruction execution rate without the complexity and overhead of dynamic clock control. Instruction-cache throttling, when used in conjunction with the TAU and the dynamic power management capability, provides the system designer with a flexible means of controlling device temperature while allowing the processor to continue operating.

The instruction-cache throttling mechanism simply throttles the instruction forwarding from the instruction cache to the instruction buffer. Normally, the instruction cache forwards four instructions to the instruction buffer every clock cycle if all the instructions hit in the cache. For thermal management, the 750GX provides a supervisor-level Instruction Cache Throttling Control (ICTC) Special Purpose Register (SPR). The instruction forwarding rate is reduced by writing a nonzero value into the ICTC[FI] field, and enabling instruction-cache throttling by setting the ICTC[E] bit to 1. An overall junction temperature reduction can result in processors that implement dynamic power management by reducing the power to the execution units while waiting for instructions to be forwarded from the instruction cache. Thus, instruction-cache throttling does not provide thermal reduction unless HID0[DPM] is set to 1.

A description of the ICTC Register and its bit fields can be found at *Section 2.1.3* on page 77.

**Note:** During instruction-cache throttling, the configuration of the PLL remains unchanged.

*Figure 10-5. Instruction Cache Throttling Control SPR Diagram*



The bit field settings of the ICTC SPR are shown in *Table 10-4* on page 348.

*Table 10-4. ICTC Bit Field Settings*

| Bits | Name | Description |
|---|---|---|
| 0-22 | Reserved | Bits reserved for future use. The system software should always write zeros to these bits when writing to the THRM SPRs. |
| 23–30 | FI | Instruction forwarding interval expressed in processor clocks.<br>0x00 0 clock cycle<br>0x01 1 clock cycle<br>.<br>.<br>0xFF 255 clock cycles |
| 31 | E | Cache throttling enable<br>0 Disable instruction-cache throttling.<br>1 Enable instruction-cache throttling. |

# 11. Performance Monitor and System Related Features

The performance-monitor facility provides the ability to monitor and count predefined events such as processor clocks, misses in the instruction cache, data cache, or L2 cache, types of instructions dispatched, mispredicted branches, and other occurrences. The count of such events (which might be an approximation) can be used to trigger the performance-monitor exception. The performance-monitor facility is not defined by the PowerPC Architecture.

The performance monitor can be used for the following:

- To increase system performance with efficient software, especially in a multiprocessing system. Memory hierarchy behavior can be monitored and studied in order to develop algorithms that schedule tasks (and perhaps partition them) and that structure and distribute data optimally.
- To improve processor architecture, the detailed behavior of the PowerPC 750GX’s structure must be known and understood in many software environments. Some environments might not be easily characterized by a benchmark or trace.
- To help system developers bring up, debug, and tune their systems.

The performance monitor uses the following 750GX-specific Special-Purpose Registers (SPRs).

- The Performance-Monitor Counter Registers (PMC1–PMC4) are used to record the number of times a certain event has occurred. UPMC1–UPMC4 provide user-level read access to these registers.
- The Monitor Mode Control Registers (MMCR0–MMCR1) are used to enable various performance-monitor interrupt functions and select events to count. UMMCR0–UMMCR1 provide user-level read access to these registers.
- The Sampled Instruction Address Register (SIA) contains the effective address of an instruction executing at or around the time that the processor signals the performance-monitor interrupt condition. USIA provides user-level read access to the SIA.

Four 32-bit counters in the 750GX count occurrences of software-selectable events. Two control registers, MMCR0 and MMCR1, are used to control performance-monitor operation. The counters and the Control Registers are supervisor-level SPRs. However, in the 750GX, the contents of these registers can be read by user-level software using separate SPRs (UMMCR0 and UMMCR1). Control fields in the MMCR0 and MMCR1 select the events to be counted, can enable a counter overflow to initiate a performance-monitor exception, and specify the conditions under which counting is enabled.

As with other PowerPC exceptions, the performance-monitor interrupt follows the normal PowerPC exception model with a defined exception vector offset (0x00F00). Its priority is below the external interrupt and above the decrementer interrupt.

## 11.1 Performance-Monitor Interrupt

The performance monitor enables the generation of a performance-monitor interrupt triggered by a counter overflow condition in one of the Performance-Monitor Counter Registers (PMC1–PMC4). A counter is considered to have overflowed when its most-significant bit is set. A performance-monitor interrupt can also be caused by flipping certain bits from 0 to 1 in the Time Base Register, which provides a way to generate a time reference-based interrupt.

Although the interrupt signal condition can occur with the external interrupt enable bit in the Machine State Register (MSR[EE]) off, the actual exception cannot be taken until the MSR[EE] bit is on.

As a result of a performance-monitor exception being taken, the action taken depends on the programmable events. To help track which part of the code was being executed when an exception was signaled, the address of the last completed instruction during that cycle is saved in the Sampled Instruction Address (SIA) register. The SIA is not updated if no instruction completed the cycle in which the exception was taken.

Exception handling for the Performance-Monitor Interrupt Exception is described in *Section 4.5.13, Performance-Monitor Interrupt (0x00F00),* on page 172.

## 11.2 Special-Purpose Registers Used by Performance Monitor

The performance monitor incorporates the SPRs listed in *Table 11-1*. All of these supervisor-level registers are accessed through Move-to Special Purpose Register (**mtspr**) and Move-from Special Purpose Register (**mfspr**) instructions.

*Table 11-1. Performance Monitor SPRs*

| SPR Number | SPR[5-9] \|\| SPR[0-4] | Register Name | Access Level |
|---|---|---|---|
| 952 | 11101 11000 | MMCR0 | Supervisor |
| 953 | 11101 11001 | PMC1 | Supervisor |
| 954 | 11101 11010 | PMC2 | Supervisor |
| 955 | 11101 11011 | SIA | Supervisor |
| 956 | 11101 11100 | MMCR1 | Supervisor |
| 957 | 11101 11101 | PMC3 | Supervisor |
| 958 | 11101 11110 | PMC4 | Supervisor |
| 936 | 11101 01000 | UMMCR0 | User (read only) |
| 937 | 11101 01001 | UPMC1 | User (read only) |
| 938 | 11101 01010 | UPMC2 | User (read only) |
| 939 | 11101 01011 | USIA | User (read only) |
| 940 | 11101 01100 | UMMCR1 | User (read only) |
| 941 | 11101 01101 | UPMC3 | User (read only) |
| 942 | 11101 01110 | UPMC4 | User (read only) |

**Notes:**

- The user registers (UMMCR0, UMMCR1, UPMC1, and so on) contain the same values as the nonuser registers but provide read-only access to the Performance-Monitor Registers while in user mode. An attempt to write to a user register in either supervisor or user mode results in a program interrupt.
- Reading and writing these registers does not synchronize the machine. An explicit synchronization instruction should be placed before and after a Move-from Special Purpose Register (**mfspr)** or Move-to Special Purpose Register (**mtspr)** instruction to one of these registers to ensure an accurate count.

### 11.2.1 Performance-Monitor Registers

This section describes the registers used by the performance monitor.

#### *11.2.1.1 Monitor Mode Control Register 0 (MMCR0)*

The Monitor Mode Control Register 0 (MMCR0) is a 32-bit SPR provided to specify events to be counted and recorded. MMCR0 can be written to only in supervisor mode. User-level software can read the contents of MMCR0 by issuing an **mfspr** instruction to UMMCR0, described in *Section 11.2.1.2* on page 351.

This register must be cleared at power up. Reading this register does not change its contents. MMCR0 can be accessed with the **mtspr** and **mfspr** instructions using SPR 952.

For a diagram of this register and a description of its fields, see *Monitor Mode Control Register 0 (MMCR0)* on page 72.

#### *11.2.1.2 User Monitor Mode Control Register 0 (UMMCR0)*

The contents of MMCR0 are reflected to UMMCR0, which can be read by user-level software. UMMCR0 can be accessed with the **mfspr** instructions using SPR 936.

#### *11.2.1.3 Monitor Mode Control Register 1 (MMCR1)*

The Monitor Mode Control Register 1 (MMCR1) functions as an event selector for Performance-Monitor Counter Registers 3 and 4 (PMC3 and PMC4). Corresponding events to the MMCR1 bits are described in *Section 11.2.1.5, Performance-Monitor Counter Registers (PMCn),* on page 351.

MMCR1 can be accessed with the **mtspr** and **mfspr** instructions using SPR 956. User-level software can read the contents of MMCR1 by issuing an **mfspr** instruction to UMMCR1, described in *Section 11.2.1.4*.

For a diagram of this register and a description of its fields, see *Monitor Mode Control Register 1 (MMCR1)* on page 74.

#### *11.2.1.4 User Monitor Mode Control Register 1 (UMMCR1)*

The contents of MMCR1 are reflected to UMMCR1, which can be read by user-level software. UMMCR1 can be accessed with the **mfspr** instructions using SPR 940.

#### *11.2.1.5 Performance-Monitor Counter Registers (PMCn)*

PMC1–PMC4 are 32-bit counters that can be programmed to generate interrupt signals when they overflow. For a diagram of these registers and a description of the fields, see *Performance-Monitor Counter Registers (PMCn)* on page 74.

Counters overflow when the high-order bit (the sign bit) becomes set; that is, they reach the value 2147483648 (0x8000_0000). However, an interrupt is not signaled unless both MMCR0[ENINT] and either PMC1INTCONTROL or PMCINTCONTROL in the MMCR0 register are also set appropriately.

**Note:** The interrupts can be masked by clearing MSR[EE]. The interrupt signal condition might occur with MSR[EE] cleared, but the exception is not taken until MSR[EE] is set. Setting MMCR0[DISCOUNT] forces counters to stop counting when a counter interrupt occurs.

Software is expected to use the **mtspr** instruction to explicitly set PMC to nonoverflowed values. Setting an overflowed value might cause an erroneous exception. For example, if both MMCR0[ENINT] and either PMC1INTCONTROL or PMCINTCONTROL are set and the **mtspr** instruction loads an overflow value, an interrupt signal might be generated without event counting having taken place.

The event to be monitored can be chosen by setting MMCR0[19:31]. The selected events are counted beginning when MMCR0 is set until either MMCR0 is reset or a performance-monitor interrupt is generated. *Table 11-2* lists the selectable events and their encodings.

*Table 11-2. PMC1 Events—MMCR0[19:25] Select Encodings*

| Encoding | Description |
|---|---|
| 000 0000 | Register holds current value. |
| 000 0001 | Number of processor cycles. |
| 000 0010 | Number of instructions that have completed. Does not include folded branches. |
| 0000011 | Number of transitions from 0 to 1 of specified bits in the Time Base Lower (TBL) register. Bits are specified through RTCSELECT, MMCR0[7–8].<br>00 31<br>01 23<br>10 19<br>11 15 |
| 0000100 | Number of instructions dispatched—0, 1, or 2 instructions per cycle. |
| 0000101 | Number of Enforce In-Order Execution of I/O (**eieio**) instructions completed. |
| 0000110 | Number of cycles spent performing table-search operations for the instruction translation lookaside buffer (ITLB). |
| 0000111 | Number of accesses that hit the L2. This event includes cache operations (such as data-cache-block set-to-zero [**dcbz**]). |
| 0001000 | Number of valid instruction effective addresses (EAs) delivered to the memory subsystem. |
| 0001001 | Number of times the address of an instruction being completed matches the address in the Instruction Address Breakpoint Register (IABR). |
| 0001010 | Number of loads that miss the L1 with latencies that exceeded the threshold value. |
| 0001011 | Number of branches that are unresolved when processed. |
| 0001100 | Number of cycles the dispatcher stalls due to a second unresolved branch in the instruction stream. |
| All others | Reserved. Might be used in a later revision. |

Bits MMCR0[26:31] specify events associated with PMC2, as shown in *Table 11-3*.

*Table 11-3. PMC2 Events—MMCR0[26:31] Select Encodings* (Page 1 of 2)

| Encoding | Description |
|---|---|
| 00 0000 | Register holds current value. |
| 00 0001 | Counts processor cycle. |
| 00 0010 | Counts completed instructions. Does not include folded branches. |
| 00 0011 | Counts transitions from 0 to 1 of TBL bits specified through MMRC0[RTCSELECT].<br>00 47<br>01 51<br>10 55<br>11 63 |
| 00 0100 | Counts instructions dispatched: 0, 1, or 2 instructions per cycle. |

*Table 11-3. PMC2 Events—MMCR0[26:31] Select Encodings* (Page 2 of 2)

| Encoding | Description |
|---|---|
| 00 0101 | Counts L1 instruction-cache misses. |
| 00 0110 | Counts ITLB misses. |
| 00 0111 | Counts L2 instruction misses. |
| 00 1000 | Counts branches predicted or resolved not taken. |
| 00 1001 | Reserved |
| 00 1010 | Counts times a reserved load operations completes. |
| 00 1011 | Counts completed load-and-store instructions. |
| 00 1100 | Counts snoops to the L1 and the L2. |
| 001101 | Counts the L1 castout to the L2. |
| 001110 | Counts completed system unit instructions. |
| 001111 | Counts instruction fetch misses in the L1. |
| 010000 | Counts branches allowing out-of-order execution that resolved correctly. |
| All others | Reserved. |

Bits MMCR1[0:4] specify events associated with PMC3, as shown in *Table 11-4*.

*Table 11-4. PMC3 Events—MMCR1[0:4] Select Encodings* (Page 1 of 2)

| Encoding | Description |
|---|---|
| 0 0000 | Register holds current value. |
| 0 0001 | Number of processor cycles. |
| 0 0010 | Number of completed instructions, not including folded branches. |
| 0 0011 | Number of transitions from 0 to 1 of specified bits in the Time Base Lower (TBL) register. Bits are specified through RTCSELECT (MMCR0[7–8]).<br>00 47<br>01 51<br>10 55<br>11 63 |
| 0 0100 | Number of instructions dispatched. 0, 1, or 2 per cycle. |
| 0 0101 | Number of L1 data-cache misses. Does not include cache operations. |
| 0 0110 | Number of data TLB (DTLB) misses. |
| 0 0111 | Number of L2 data misses. |
| 0 1000 | Number of predicted branches that were taken. |
| 0 1001 | Reserved. |
| 0 1010 | Number of store conditional instructions completed. |
| 0 1011 | Number of instructions completed from the floating-point unit (FPU). |
| 0 1100 | Number of L2 castouts caused by snoops to modified lines. |
| 0 1101 | Number of cache operations that hit in the L2 cache. |
| 0 1110 | Reserved. |
| 0 1111 | Number of cycles generated by L1 load misses. |

*Table 11-4. PMC3 Events—MMCR1[0:4] Select Encodings* (Page 2 of 2)

| Encoding | Description |
|---|---|
| 1 0000 | Number of branches in the second speculative stream that resolve correctly. |
| 1 0001 | Number of cycles the BPU stalls due to LR or CR unresolved dependencies. |
| All others | Reserved. Might be used in a later revision. |

Bits MMCR1[5:9] specify events associated with PMC4, as shown in *Table 11-5*.

*Table 11-5. PMC4 Events—MMCR1[5:9] Select Encodings*

| Encoding | Comments |
|---|---|
| 00000 | Register holds current value |
| 00001 | Number of processor cycles |
| 00010 | Number of completed instructions, not including folded branches |
| 00011 | Number of transitions from 0 to 1 of specified bits in the Time Base Lower (TBL) register. Bits are specified through RTCSELECT, MMCR0[7–8]. 00 = 31, 01 = 23, 10 = 19, 11 = 15 |
| 00100 | Number of instructions dispatched. 0, 1, or 2 per cycle |
| 00101 | Number of L2 castouts |
| 00110 | Number of cycles spent performing table searches for DTLB accesses. |
| 00111 | Reserved. Might be used in a later revision. |
| 01000 | Number of mispredicted branches. Reserved for future use. |
| 01001 | Reserved. Might be used in a later revision. |
| 01010 | Number of store conditional instructions completed with reservation intact |
| 01011 | Number of completed **sync** instructions |
| 01100 | Number of snoop request retries |
| 01101 | Number of completed integer operations |
| 01110 | Number of cycles the branch processing unit (BPU) cannot process new branches due to having two unresolved branches |
| All others | Reserved. Might be used in a later revision. |

The PMC registers can be accessed with the **mtspr** and **mfspr** instructions using the following SPR numbers:

- PMC1 is SPR 953.
- PMC2 is SPR 954.
- PMC3 is SPR 957.
- PMC4 is SPR 958.

### *11.2.1.6 User Performance-Monitor Counter Registers (UPMC1–UPMC4)*

The contents of the PMC1–PMC4 are reflected to UPMC1–UPMC4, which can be read by user-level software. The UPMC registers can be read with the **mfspr** instructions using the following SPR numbers:

- UPMC1 is SPR 937.
- UPMC2 is SPR 938.
- UPMC3 is SPR 941.
- UPMC4 is SPR 942.

### *11.2.1.7 Sampled Instruction Address Register (SIA)*

The Sampled Instruction Address Register (SIA) is a supervisor-level register that contains the effective address of an instruction executing at or around the time that the processor signals the performance-monitor interrupt condition. The SIA is shown in *Sampled Instruction Address Register (SIA)* on page 75.

If the performance-monitor interrupt is triggered by a threshold event, the SIA contains the address of the exact instruction (called the sampled instruction) that caused the counter to overflow.

If the performance-monitor interrupt was caused by something besides a threshold event, the SIA contains the address of the last instruction completed during that cycle. SIA can be accessed with the **mtspr** and **mfspr** instructions using SPR 955.

### *11.2.1.8 User Sampled Instruction Address Register (USIA)*

The contents of SIA are reflected to USIA, which can be read by user-level software. USIA can be accessed with the **mfspr** instructions using SPR 939.

## 11.3 Event Counting

Counting can be enabled if conditions in the processor state match a software-specified condition. Because a software task scheduler can switch a processor’s execution among multiple processes and because statistics on only a particular process might be of interest, a facility is provided to mark a process. The performance-monitor (PM) bit, MSR[29], is used for this purpose. System software can set this bit when a marked process is running. This enables statistics to be gathered only during the execution of the marked process. The states of MSR[PR] and MSR[PM] together define a state that the processor (supervisor or program) and the process (marked or unmarked) can be in at any time. If this state matches a state specified by the MMCR0, the state for which monitoring is enabled, counting is enabled.

The following states can be monitored:

- Supervisor only
- User only
- Marked and user only
- Not marked and user only
- Marked and supervisor only
- Not marked and supervisor only
- Marked only
- Not marked only

In addition, one of two unconditional counting modes can be specified:

- Counting is unconditionally enabled regardless of the states of MSR[PM] and MSR[PR]. This can be accomplished by clearing MMCR0[0–4].
- Counting is unconditionally disabled regardless of the states of MSR[PM] and MSR[PR]. This is done by setting the disable bit (DIS) to 1 (MMCR0[0] = 1).

The performance-monitor counters count specified events and are used to generate performance-monitor exceptions when an overflow (most-significant bit is a 1) situation occurs. The 750GX performance monitor has four, 32-bit registers that can count up to 0x7FFFFFFF (2,147,483,648 in decimal) before overflowing. Bit 0 of the registers is used to determine when an interrupt condition exists.

## 11.4 Event Selection

Event selection is handled through MMCR0 and MMCR1.

- The four event-select fields in MMCR0 and MMCR1 are:
  - MMCR0[19:25] PMC1SELECT PMC1 input selector. 128 events selectable; 25 defined. See *Table 11-2* on page 352.
  - MMCR0[26:31] PMC2SELECT PMC2 input selector. 64 events selectable; 21 defined. See *Table 11-3* on page 352.
  - MMCR0[0:4] PMC3SELECT PMC3 input selector. 32 events selectable and defined. See *Table 11-4* on page 353.
  - MMCR0[5:9] PMC4SELECT PMC4 input selector. 32 events selectable. See *Table 11-5* on page 354.
- In the tables, a correlation is established between each counter, events to be traced, and the pattern required for the desired selection.
- The first five events are common to all four counters and are considered to be reference events. These are as follows.
  - 00000 Register holds current value
  - 00001 Number of processor cycles
  - 00010 Number of completed instructions, not including folded branches
  - 00011 Number of transitions from 0 to 1 of specified bits in the Time Base Lower (TBL) register. Bits are specified through RTCSELECT, MMCR0[7–8].

    00 = 31
    01 = 23
    10 = 19
    11 = 15
  - 00100 Number of instructions dispatched. 0, 1, or 2 per cycle
- Some events can have multiple occurrences per cycle, and therefore need two or three bits to represent them.

## 11.5 Notes

The following warnings should be noted:

- Only those load and stores in queue position 0 of their respective load/store queues are monitored when a threshold event is selected in PMC1.
- The 750GX cannot accurately track threshold events with respect to the following types of loads and stores:
  - Unaligned load-and-store operations that cross a word boundary
  - Load-and-store multiple operations
  - Load-and-store string operations

## 11.6 Debug Support

### 11.6.1 Overview

The 750GX provides the following debug support features:

- Branch trace
- Single step instruction trace
- Instruction-address breakpoint
- Data-address breakpoint
- Externally triggered soft stop

The trace mode allows either a single step trace if MSR[SE] = 1 or a branch trace if MSR[BE] = 1. The instruction-address breakpoint and data-address breakpoint modes are invoked by setting the appropriate bits in the Instruction Address Breakpoint Register (IABR) and Data Address Breakpoint Register (DABR). Each of these debug features except for data-address breakpoint is a common feature of PowerPC devices. The variances are noted in the following paragraphs.

### 11.6.2 Data-Address Breakpoint

The data-address breakpoint feature is controlled by the DABR Special Purpose Register which is described in *Section 4.5.17, Data Address Breakpoint Exception,* on page 175. The data-address breakpoint action can be one of the following:

- Data-storage interrupt (DSI)
- Soft stop
- Hard stop

A DSI on a data access does not complete the interrupting instruction.

## 11.7 JTAG/COP Functions

### 11.7.1 Introduction

The 750GX implements the Joint Test Action Group (JTAG) and common on-chip processor (COP) functions for facilitating board testing and chip debug. The JTAG boundary scan features are used for board testing, while the COP features are used mainly for system debug using a RISCWatch. The JTAG features and the interface are fully compliant with the IEEE 1149.1a-1993 standard. The COP functions are compliant with the JTAG standard whenever possible, and the COP external interface adheres to the IEEE 1149.1a-1993 serial protocol. In this document, IEEE 1149.1a-1993 and JTAG are used interchangeably. The 750GX does support the optional test reset pin.

### 11.7.2 Processor Resources Available through JTAG/COP Serial Interface

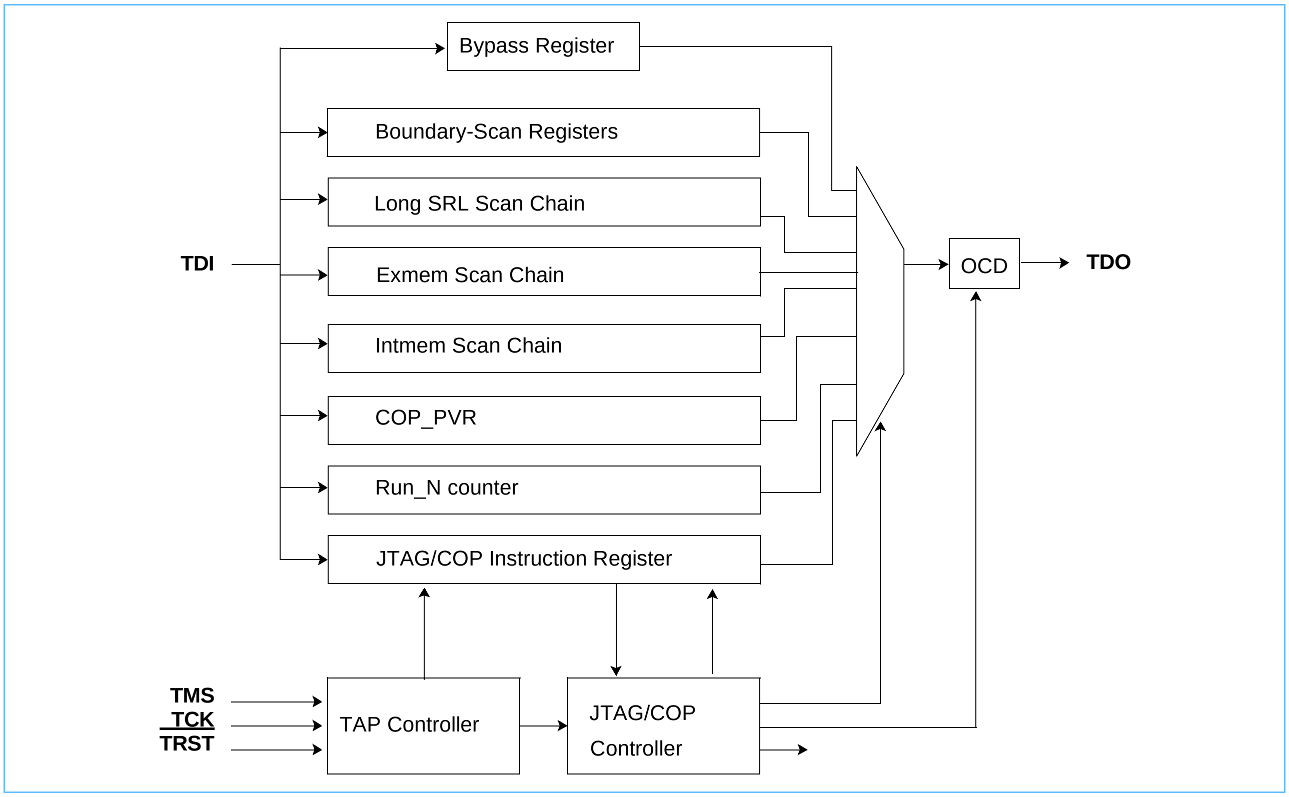
The shift register latches (SRLs) on the 750GX are linked so that data can be shifted serially through them to either control or observe resources (such as caches and register files) within the processor. Various chain configurations are selected by the COP and placed between the JTAG TDI and TDO pins as shown in *Figure 11-1, 750GX IEEE 1149.1a-1993/COP Organization,* on page 358. RISCWatch configures and controls the appropriate SRL chains to read and write various processor resources for system debug, including:

- Internal registers (such as the general-purpose, floating-point, and processor version registers)
- Data cache
- Instruction cache
- L2 cache
- L2 tag
- Data tag
- Instruction tag
- Data translation lookaside buffer (TLB)
- Data Segment Registers
- Instruction TLB
- Instruction Segment Registers
- Instruction Block-Address-Translation (BAT) Registers
- External memory

INTMEM will allow reading and writing the above arrays while accessing a chain shorter than the LSRL. INTMEM is a proper subset of the LSRL (Long Shift Register Latch).

The scan chains for the 750GX are shown *Figure 11-1*. The 750GX does support the optional test reset (TRST) pin.

*Figure 11-1. 750GX IEEE 1149.1a-1993/COP Organization*



## 11.8 Resets

The 750GX supports two types of resets: a hard and a soft reset.

### 11.8.1 Hard Reset

The hard reset is triggered by the assertion of the hard reset pin, $\overline{\text{HRESET}}$. The $\overline{\text{HRESET}}$ pin is asserted by several sources:

- System power-on reset
- System reset from a panel switch
- RISCWatch

The duration of $\overline{\text{HRESET}}$ assertion depends on two factors: phase-locked loop (PLL) lock time (as specified in the 750GX datasheet) and internal processor state initialization time. During a hard reset, the internal latches are scanned with zeros for initialization which requires a minimum of 255 CPU clocks. A power-on reset (POR) requires that both $\overline{\text{HRESET}}$ and $\overline{\text{TRST}}$ be active. $\overline{\text{HRESET}}$ must be active for the duration that includes the PLL lock time plus the 255 CPU clocks for initialization. POR also requires that the test access port (TAP) controller enter the test-logic-reset state by applying $\overline{\text{TRST}}$.

For a hard reset to recover from a hardware problem, like a checkstop, only 255 bus clock cycles are necessary to initialize the state of the processor provided the PLL remains locked.

During hard reset, all off-chip drivers will be tristated. After removal of $\overline{\text{HRESET}}$, the processor will vector to the system reset interrupt routine at 0xFFF00100 with MSR[IP] set high.

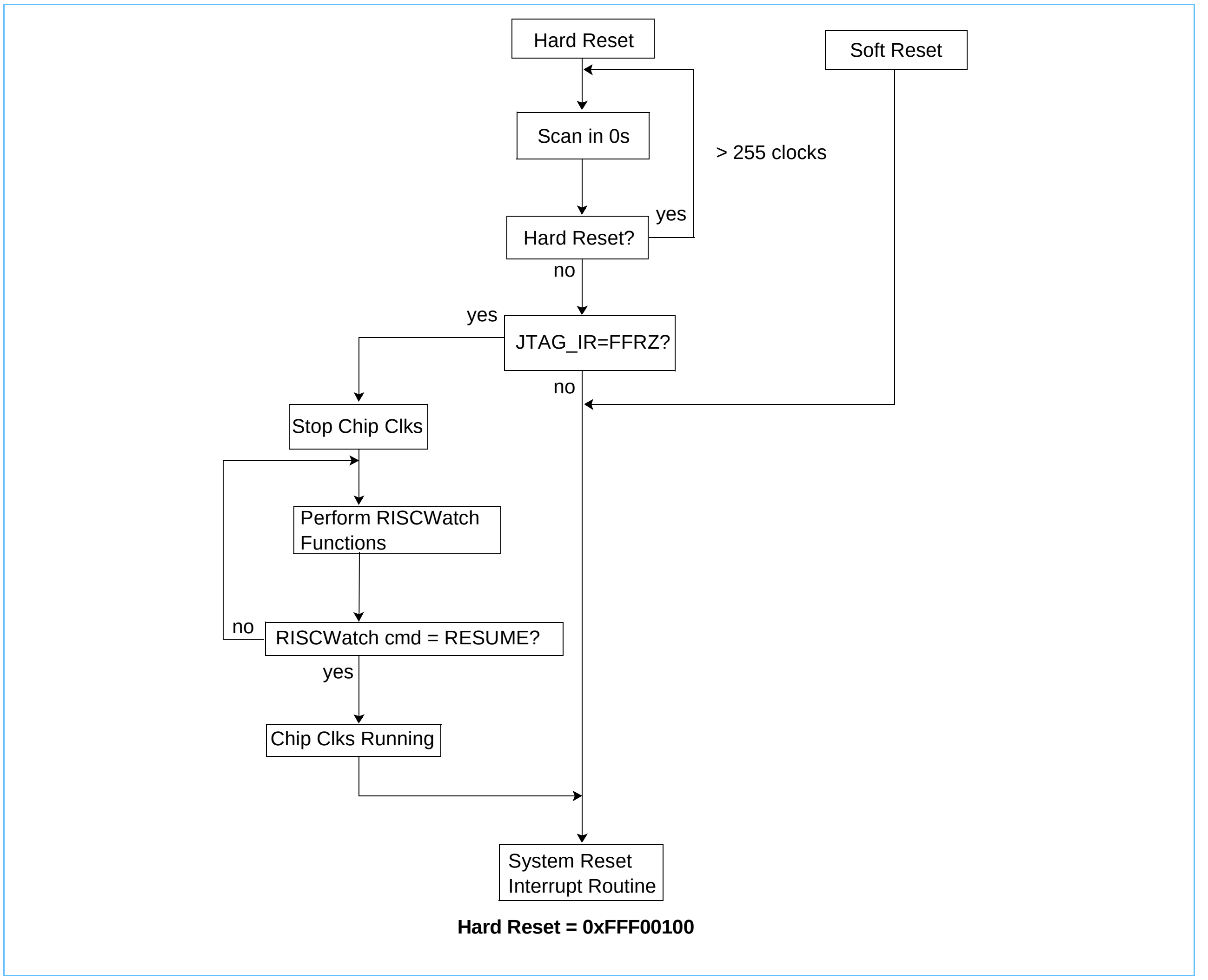
During $\overline{\text{HRESET}}$, the latches dedicated to JTAG functions are not initialized. The JTAG reset signal, $\overline{\text{TRST}}$, resets the dedicated JTAG logic. This is in compliance with the IEEE 1149.1a-1993 standard, which prohibits the chip reset from resetting the JTAG logic. The RISCWatch can stop the processor shortly after the SRL0 scan sequence during a hard reset, by issuing a COP force freeze command. This allows complete control of the processor by the RISCWatch from a hard reset.

### 11.8.2 Soft Reset

The processor executes a system reset interrupt if the $\overline{\text{SRESET}}$ signal is asserted. Unlike a hard reset, the latches are not initialized and the output of the MSR[IP] bit is not modified. Therefore, the system reset interrupt vector address depends on the MSR[IP] bit setting prior to the assertion of $\overline{\text{SRESET}}$. The $\overline{\text{SRESET}}$ signal must be asserted for a minimum of two bus clocks.

### 11.8.3 Reset Sequence

*Figure 11-2. Reset Sequence*



## 11.9 Checkstops

A checkstop causes the processor to halt and assert the checkstop output pin, CKSTP_OUT. Once the 750GX enters a checkstop state, only a hard reset can clear the processor.

### 11.9.1 Checkstop Sources

Following is the list of checkstop sources:

- Machine Check with MSR[ME] = 0.
  If MSR[ME] = 0 when a machine-check interrupt occurs, then the checkstop state is entered. The machine-check sources for the 750GX are as follows:
  - TEA assertion on the 60x bus
  - Address-parity error on the 60x bus
  - Data-parity error on the 60x bus
  - Machine-check input pin (MCP)
  - Locked L2 Snoop, with the snoop hit in locked line error enable bit in the L2 Cache Control Register (L2CR[SHEE]) set
  - A parity error in either the instruction tag, data tag, instruction cache, data cache, or L2 tag (if enabled)
- Checkstop input pin (CKSTP_IN)

### 11.9.2 Checkstop Control Bits

Some of the checkstop sources can be controlled via Hardware-Implementation-Dependent Register 0 (HID0) and the L2CR register bits.

*Table 11-6. HID0 Checkstop Control Bits*

| Bits | Field Name | Hard Reset State | Description |
|---|---|---|---|
| 0 | EMCP | 0 | MCP pin mask bit<br>1 Enables MCP to cause a checkstop if MSR(ME) = 0, or a machine-check interrupt if MSR(ME) = 1.<br>0 Masks out the MCP pin. Therefore, the MCP pin cannot generate a machine-check interrupt or a checkstop. The main purpose of this bit is to mask out further machine-check interrupts from MCP, similar to the MSR(EE) bit for external interrupts. |
| 2 | EBA | 0 | Bus address-parity checking enable<br>1 Enables an address-parity error to cause a checkstop if MSR(ME) = 0, or a machine-check interrupt if MSR(ME) = 1.<br>0 Prevents address-parity checking. |
| 3 | EBD | 0 | Bus data-parity checking enable<br>1 Enables a data-parity error to cause a checkstop if MSR(ME) = 0, or a machine-check interrupt if MSR(ME) = 1.<br>0 Prevents data-parity checking. |
| **Note:** The EBA and EBD bits allow the processor to operate with memory subsystems that do not generate parity. A checkstop latch is provided in the COP to indicate the checkstop source. | | | |

*Table 11-7* shows the control bits for HID2.

*Table 11-7. HID2 Checkstop Control Bits*

| Bits | Field Name | Hard Reset State | Description |
|---|---|---|---|
| 29 | ICPE | | Enable L1 instruction-cache or instruction-tag parity checking. |
| 30 | DCPE | | Enable L1 data-cache or data-tag parity checking. |
| 31 | L2PE | | Enable L2 Tag parity checking. |

*Table 11-8. L2CR Checkstop Control Bits*

| Bits | Field Name | Hard Reset State | Description |
|---|---|---|---|
| 22 | SHEE | 0 | 1 Enables a checkstop when snoop encounters a locked L2 line.<br>0 Prevents checkstop.<br>Enables a snoop hit in a locked line to raise a machine check. See *Section 9.6.1.2, Locked Cache Operation,* on page 331 for more information. |

The checkstop input pin (CKSTP_IN) always causes a checkstop regardless of the state of the MSR[ME] bit.

**Note:** All checkstops are disabled by a hard reset. To enable the individual checkstops, the user has to set the appropriate checking enable bits in HID0 and L2CR register.

### 11.9.3 Open-Collector-Driver States during Checkstop

All the nontest Open Collector Driver (OCD) states except for the checkstop output pin, CKSTP_OUT, are disabled during a checkstop. This forces the checkstopped processor off the bus, and prevents potential OCD damage due to multiple drivers being enabled on the same bus during a checkstop.

### 11.9.4 Vacancy Slot Application

The checkstop input (CKSTP_IN) to the 750GX can be used to implement a vacancy slot mechanism since a checkstop halts the processor and tristates the OCDs as mentioned above. Several points need to be considered for the vacancy slot implementation:

- The internal checkstop logic requires its latches to be initialized properly upon a hard reset, and SYSCLK to be running. Therefore, the processor that is being replaced needs to go through the same hard reset sequence as the replacement processor. With SYSCLK running, the checkstop power consumption of the 750GX should be similar to the power consumption of the part in nap mode.
- Since a hard reset clears all checkstop conditions, the CKSTP_IN pin needs to be kept asserted after the negation of a hard reset for the part to enter checkstop.
- The checkstop output pin, CKSTP_OUT, which is asserted, needs to be isolated.
- The IEEE 1149.1a-1993 requires the boundary scan output pin, TDO, to be controllable only by the JTAG logic. Therefore, if the system POR sequence leaves the TDO pin tristated, then no further isolation is required. However, if boundary scan is to be done with the replacement processor, then the JTAG logic of the processor being replaced must be disabled through TRST = 0.

## 11.10 750GX Parity

Parity is implemented for the following arrays: instruction cache, instruction tag, data cache, data tag, and L2 tag. All parity errors, when parity is enabled, result in either a machine-check or checkstop interrupt that is not recoverable.

For all of the following arrays, parity for a given set of data is a one if there is an odd number of ones in the data (even parity).

Parity is computed each time data is written to the arrays, independent of the parity checking enable/disable control bits.

The Force Bad Parity control bits (5 bits) are provided as user visible bits in the HID2 control register and control the parity bits written when the arrays are written. This is again independent of the parity enable/disable control bits.

The parity checking enable/disable control bits are provided to select when parity is to be checked for reads by array group (L1 instruction cache and tag, L1 data cache and tag, and L2 tag). If parity checking is enabled and bad parity is found on a read for the enabled array, then the MSR(ME) bit controls the action taken by the processor for the parity error. Parity checking can be enabled or disabled at any time in the code stream without changing the array enable/disable state, and the arrays do not require invalidation.

The MSR(ME) bit enables the processor to take a machine-check interrupt allowing the operating system to determine the failing array. If this bit is not asserted, then the processor will take a checkstop. See *Section 4.5.2.1, Machine-Check Exception Enabled (MSR[ME] = 1),* on page 168 for further details.

The parity status bits are set at the time of the detection of bad parity only when the array parity enable is set. These bits are by array group (L1 instruction cache and tag, L2 data cache and tag, and L1 tag) and are helpful for determining the problem within the machine-check interrupt handler.

The HID2 Register updates are not serialized in the processor. Therefore, it is strongly recommended to include an Instruction Synchronization (**isync)** instruction after any write to the HID2 Register to ensure the changes are complete before proceeding in the code stream.

### 11.10.1 Parity Control and Status

Parity is enabled with the Hardware-Implementation-Dependent Register 2 (HID2).

For a diagram of this register and a description of its fields, see *Hardware-Implementation-Dependent Register 2 (HID2)* on page 71.

HID2 SPR number is 1016 decimal, (spr[5-9] = 11111, spr[0-4] = 11000).

The status bits (25:27) are set when a parity error is detected and cleared when the HID2 Register is written.

### 11.10.2 Enabling Parity Error Detection

Parity error detection can be enabled at any time by setting the parity enable bits for the desired arrays in the HID2 Register (ICPE, DCPE, and L2PE for the ICache/ITag, DCache/DTag, and L2Tag respectively). Parity errors are reported with the parity status bits in the HID2 Register (ICPS, DCPS, and L2PS for ICache/ITag, DCache/DTag, and L2Tag respectively). The parity status bits are read only and are automatically cleared each time the HID2 register is written.

### 11.10.3 Parity Errors

All parity errors will cause a machine-check interrupt. Since this is an imprecise interrupt, recovery is not possible. To determine the cause of the machine check, the software must have set the machine-check enable [ME] bit of the MSR and have an interrupt handler located at 0x00200. This handler can read the parity status bits in HID2 for display or checking. If the cause of the machine check is found to be a parity error then after the handler has completed an $\overline{\text{HRESET}}$ must be initiated.

# Acronyms and Abbreviations

| | |
|---|---|
| BAT | block-address translation |
| BHT | branch history table |
| BIST | built-in self test |
| BIU | bus interface unit |
| BPU | branch processing unit |
| BSDL | Boundary-Scan Description Language |
| BTIC | branch target instruction cache |
| BUID | bus unit ID |
| CMOS | complementary metal-oxide semiconductor |
| COP | common on-chip processor |
| CQ | completion queue |
| CR | Condition Register |
| CTR | Count Register |
| DABR | Data Address Breakpoint Register |
| DAR | Data Address Register |
| DBAT | data BAT |
| DCMP | data TLB compare |
| DEC | Decrementer Register |
| DLL | delay-locked loop |
| DMISS | data TLB miss address |
| DMMU | data MMU |
| DPM | dynamic power management |
| DSISR | Register used for determining the source of a DSI exception. |
| DTLB | data translation lookaside buffer |
| EA | effective address |
| EAR | External Access Register |
| ECC | error checking and correction |
| FIFO | first-in-first-out |

| | |
|---|---|
| FPR | Floating Point Register |
| FPSCR | Floating-Point Status and Control Register |
| FPU | floating-point unit |
| GPR | General Purpose Register |
| HIDn | Hardware-Implementation-Dependent Register |
| IABR | Instruction Address Breakpoint Register |
| IBAT | instruction BAT |
| ICTC | Instruction Cache Throttling Control Register |
| IEEE | Institute for Electrical and Electronics Engineers |
| IMMU | instruction MMU |
| IQ | instruction queue |
| ITLB | instruction translation lookaside buffer |
| IU | integer unit |
| JTAG | Joint Test Action Group |
| L2 | secondary cache (level 2 cache) |
| L2CR | L2 Cache Control Register |
| LIFO | last-in-first-out |
| LR | lInk Register |
| LRU | least recently used |
| LSb | least-significant bit |
| LSB | least-significant byte |
| LSU | load/store unit |
| MEI | modified/exclusive/invalid |
| MESI | modified/exclusive/shared/invalid—cache-coherency protocol |
| MMCRn | Monitor Mode Control Registers |
| MMU | memory management unit |
| MSb | most-significant bit |
| MSB | most-significant byte |
| MSR | Machine State Register |

| | |
|---|---|
| NaN | not a number |
| no-op | no operation |
| OEA | operating environment architecture |
| PID | processor identification tag |
| PLL | phase-locked loop |
| PLRU | pseudo least recently used |
| PMCn | Performance-Monitor Counter Registers |
| POR | power-on reset |
| POWER | Performance Optimized with Enhanced RISC architecture |
| PTE | page table entry |
| PTEG | page-table-entry group |
| PVR | Processor Version Register |
| RAW | read-after-write |
| RISC | reduced instruction set computing |
| RTL | register transfer language |
| RWITM | read with intent to modify |
| RWNITM | read with no intent to modify |
| SDA | sampled data address register |
| SDR1 | Register that specifies the page table base address for virtual-to-physical address translation. |
| SIA | Sampled Instruction Address Register |
| SPR | Special-Purpose Register |
| SRn | Segment Register |
| SRR0 | Machine Status Save/Restore Register 0 |
| SRR1 | Machine Status Save/Restore Register 1 |
| SRU | system register unit |
| TAU | thermal-management assist unit |
| TB | time-base facility |
| TBL | Time Base Lower Register |
| TBU | Time Base Upper Register |

| | |
|---|---|
| THRM n | Thermal-Management Registers |
| TLB | translation lookaside buffer |
| TTL | transistor-to-transistor logic |
| UIMM | unsigned immediate value |
| UISA | user instruction set architecture |
| UMMCRn | User Monitor Mode Control Registers |
| UPMCn | User Performance-Monitor Counter Registers |
| USIA | User Sampled Instruction Address Register |
| VEA | virtual environment architecture |
| WAR | write-after-read |
| WAW | write-after-write |
| WIMG | write-through/caching-inhibited/memory-coherency enforced/guarded bits |
| XATC | extended address transfer code |
| XER | Register used for indicating conditions such as carries and overflows for integer operations. |

# Index

# Revision Log

| Revision Date | Contents of Modification |
|---|---|
| February 27, 2004 | Initial release (version 1.0) |
| September 30, 2004 | (version 1.1)<br>on page 26, added the following to the list under "2-stage load/store unit (LSU)."<br>"4-entry load queue."<br>On page 32, added a third paragraph under Section 1.2.2.3 Load/Store Unit.<br>On page 279, changed the number "Two" to "Four" in the second subbullet in the bullet list.<br>On page 279, rewrote the first paragraph after the bullet list.<br>On page 286, rewrote the first paragraph in Section 8.2.2 Miss-Under-Miss.<br>On page 286, added a note after the first paragraph.<br>On page 289, added "in LSU " to #8.<br>On page 289, added " in BIU" "(same as the reload-request queue)" to #13. |
| March 27, 2006 | (version 1.2)<br>Removed “Preliminary “ from title and to top of every page.<br>Added “and 750GL” to UM title and to top of every page.<br>On page 19, added “and 750GL” to the fitst paragraph of “About This Manual.”<br>On page 19, added two notes following first paragraph of “About This Manual.” |