G522-0291-00

MPCBUSIF/AD
3/97
REV. 0

PowerPC™ Microprocessor Family:

The Bus Interface for 32-Bit Microprocessors

IBM PowerPC MOTOROLA

CONTENTS

Paragraph Number	Title	Page Number

About This Document

Chapter 1
Overview

Chapter 2
Signal Descriptions

CONTENTS

CONTENTS

Paragraph Number	Title	Page Number

Chapter 3
Memory Access Protocol

CONTENTS

CONTENTS

Paragraph Number	Title	Page Number

Chapter 5
System Status Signals

Chapter 6
Additional Bus Configurations

CONTENTS

Paragraph Number	Title	Page Number

Chapter 7
Direct-Store Interface

Chapter 8
System Considerations

CONTENTS

Paragraph Number	Title	Page Number

Appendix A
Processor Summary

Appendix B
Processor Clocking Overview

Appendix C
Processor Upgrade Suggestions

Appendix D
L2 Considerations for the PowerPC 604 Processor

CONTENTS

Paragraph Number	Title	Page Number

Appendix E
Coherency Action Tables

ILLUSTRATIONS

ILLUSTRATIONS

TABLES

TABLES

About This Document

The primary objective of this document is to provide a detailed functional description of the 60x bus interface, as implemented on the PowerPC 601™, PowerPC 603™, and PowerPC 604™ family of PowerPC™ microprocessors. This document is intended to help system and chip set developers by providing a centralized reference source to identify the bus interface presented by the 60x family of PowerPC microprocessors. This document should be used in conjunction with the individual microprocessors' user's manuals, hardware specifications, and the *PowerPC Microprocessor Family: The Programming Environments* (referred to as *The Programming Environments Manual*).

The 60x bus is the communication channel for the first generation of PowerPC microprocessors. This bus description documents the current operations and system implementation information for the following PowerPC processors:

- PowerPC 601 processor. The 601 is the first PowerPC processor and is designed for desktop, server, and workstation implementations and is designed to support implementation in multiprocessing systems.
- PowerPC 603 processors. References to the 603 include the PowerPC 603e™ processors unless otherwise specified. The 603 family of processors is optimized for implementation in low-power systems, and includes bus support for power management, but provides less support for multiprocessing than either the 601 or 604 families of processors.
- PowerPC 604 processors. References to the 604 include the PowerPC 604e™ processors unless specified otherwise. The 604 family of processors is designed for implementation in desktop, workstation, and server systems and provides extensive support both for multiprocessing and for power management.

Although this book can be used as a general guide for the PowerPC 602™ processor, and for some other 32-bit PowerPC processors, it does not include descriptions of operations unique to that processor.

All of these processors support 32-bit addressing, and provide separate address and data buses. All provide 64-bit data buses, and some allow the option of configuring the data bus to work in an optional 32-bit mode.

The 60x bus allows processors to access or otherwise communicate with other resources that may share the bus, including system memory, secondary caches, I/O devices, bus arbiters, and other devices. By and large, the 60x bus implementation is consistent among the 601, 603, and 604; however, because the PowerPC architecture supports a broad range of system implementations, each processor offers unique features.

Primary goals of this book are to provide the reader with an understanding of the operations of the basic signals that are common to and required by all 60x processors as well as a familiarity with those signals that are not common to all parts or required for basic operation that can maximize the performance of a system implementation. To aid in this understanding, this document focuses on the following bus relationships among current 60x microprocessors:

- General bus characteristics
- Common bus characteristics
- Differences between current implementations

This document specifically describes the communication signals and protocols used by the 601, 603, and 604, and does not describe the power, test, and clock signals. For that information, refer to the particular 60x microprocessor user's manual.

In this document, the terms '601', '603', '603e' '604', '604e', and '60x bus' are used as abbreviations for 'PowerPC 601 microprocessor', 'PowerPC 603 microprocessor', PowerPC 603e microprocessor', 'PowerPC 604 microprocessor', PowerPC 604e microprocessor', and 'PowerPC 60x microprocessor bus interface', respectively. The terms 'processor bus interface' and 'interface' are analogous with the 60x bus.

To locate any published errata or updates for this document, refer to the world-wide web at http://www.mot.com/powerpc/ or at http://www.chips.ibm.com/products/ppc.

Audience

This document is intended for system and processor hardware developers who are developing products that incorporate or interface with the 60x microprocessors. It can also benefit software developers who work with products that use these microprocessors.

Organization

Following is a summary and brief description of the major sections of this manual:

- Chapter 1, "Overview," is useful for readers wanting a general understanding of the features and functions of the PowerPC processor interface. It defines various operational subsets of these features and functions.
- Chapter 2, "Signal Descriptions," describes each processor input and output signal and gives timing considerations.

- Chapter 3, “Memory Access Protocol,” describes the operation of the processor interface for memory operations.
- Chapter 4, “Memory Coherency,” describes bus features and protocols for maintaining coherency in uniprocessor and multiprocessor systems.
- Chapter 5, “System Status Signals,” describes the operation of the interrupt, checkstop, and reset signals. It also includes a brief overview of the asynchronous exceptions, with particular attention given to the differences in how the 60x processors implement those exceptions.
- Chapter 6, “Additional Bus Configurations,” describes some alternate modes available for the bus.
- Chapter 7, “Direct-Store Interface, ” describes the optional direct-store interface for synchronous I/O.
- Chapter 8, “System Considerations,” gives useful information for designing systems that use the processor bus.
- Appendix A, “Processor Summary,” summarizes the processor objectives and a table comparing processor behavior.
- Appendix B, “Processor Clocking Overview,” describes the clocking for the 601, 603, and 604.
- Appendix C, “Processor Upgrade Suggestions,” describes considerations for systems designed to allow a processor upgrade.
- Appendix D, “L2 Considerations for the PowerPC 604 Processor,” gives useful information for those implementing an L2 cache on a system with a 604.
- Appendix E, “Coherency Action Tables,” provides a comprehensive table of coherency actions that are generated in response to various bus operations in different contexts such as WIM bit settings, cache state, and bus states.

Suggested Reading

This section lists additional reading that provides background for the information in this manual as well as general information about the PowerPC architecture.

General Information

The following documentation provides useful information about the PowerPC architecture and computer architecture in general:

- The following books are available from the Morgan-Kaufmann Publishers, 340 Pine Street, Sixth Floor, San Francisco, CA 94104; Tel. (800) 745-7323 (U.S.A.), (415) 392-2665 (International); internet address: mkp@mkp.com.
  — *The PowerPC Architecture: A Specification for a New Family of RISC Processors,* Second Edition, by International Business Machines, Inc.

  Updates to the architecture specification are accessible via the world-wide web at http://www.austin.ibm.com/tech/ppc-chg.html.

— *PowerPC Microprocessor Common Hardware Reference Platform: A System Architecture*, by Apple Computer, Inc., International Business Machines, Inc., and Motorola, Inc.

— *Macintosh Technology in the Common Hardware Reference Platform*, by Apple Computer, Inc.

— *Computer Architecture: A Quantitative Approach*, Second Edition, by John L. Hennessy and David A. Patterson

- *Inside Macintosh: PowerPC System Software*, Addison-Wesley Publishing Company, One Jacob Way, Reading, MA, 01867; Tel. (800) 282-2732 (U.S.A.), (800) 637-0029 (Canada), (716) 871-6555 (International)
- *PowerPC Programming for Intel Programmers*, by Kip McClanahan; IDG Books Worldwide, Inc., 919 East Hillsdale Boulevard, Suite 400, Foster City, CA, 94404; Tel. (800) 434-3422 (U.S.A.), (415) 655-3022 (International)

PowerPC Documentation

The PowerPC documentation is organized in the following types of documents:

- User's manuals—These books provide details about individual PowerPC implementations and are intended to be used in conjunction with *The Programming Environments Manual.* These include the following:

  — *PowerPC 601™ RISC Microprocessor User's Manual*: MPC601UM/AD (Motorola order #) and 52G7484/(MPR601UMU-02) (IBM order #)

  — *PowerPC 602™ RISC Microprocessor User's Manual*: MPC602UM/AD (Motorola order #) and MPR602UM-01 (IBM order #)

  — *PowerPC 603e™ RISC Microprocessor User's Manual with Supplement for PowerPC 603 Microprocessor*: MPC603EUM/AD (Motorola order #) and MPR603EUM-01 (IBM order #)

  — *PowerPC 604™ RISC Microprocessor User's Manual*: MPC604UM/AD (Motorola order #) and MPR604UMU-01 (IBM order #)

- Programming environments manuals—These books provide information about resources defined by the PowerPC architecture that are common to PowerPC processors. There are two versions, one that describes the functionality of the combined 32- and 64-bit architecture models and one that describes only the 32-bit model.

  — *PowerPC Microprocessor Family: The Programming Environments*, Rev 1: MPCFPE/AD (Motorola order #) and G522-0290-00 (IBM order #)

  — *PowerPC Microprocessor Family: The Programming Environments for 32-Bit Microprocessors*, Rev. 1, MPCFPE32B/AD (Motorola order #)

- *Implementation Variances Relative to Rev. 1 of The Programming Environments Manual* is available via the world-wide web at http://www.mot.com/powerpc/ or at http://www.chips.ibm.com/products/ppc.

- Addenda/errata to user's manuals—Because some processors have follow-on parts an addendum is provided that describes the additional features and changes to functionality of the follow-on part. These addenda are intended for use with the corresponding user's manuals. These include the following:
  - — *Addendum to PowerPC 603e RISC Microprocessor User's Manual: PowerPC 603e Microprocessor Supplement and User's Manual Errata:* MPC603EUMAD/AD (Motorola order #) and SA14-2034-00 (IBM order #)
  - — *Addendum to PowerPC 604 RISC Microprocessor User's Manual*: *PowerPC 604e™ Microprocessor Supplement and User's Manual Errata*: MPC604UMAD/AD (Motorola order #) and SA14-2056-01 (IBM order #)
- Hardware specifications—Hardware specifications provide specific data regarding bus timing, signal behavior, and AC, DC, and thermal characteristics, as well as other design considerations for each PowerPC implementation. These include the following:
  - — *PowerPC 601 RISC Microprocessor Hardware Specifications*: MPC601EC/D (Motorola order #) and MPR601HSU-03 (IBM order #)
  - — *PowerPC 602 RISC Microprocessor Hardware Specifications*: MPC602EC/D (Motorola order #) and SC229897-00 (IBM order #)
  - — *PowerPC 603 RISC Microprocessor Hardware Specifications*: MPC603EC/D (Motorola order #) and G522-0289-00 (IBM order #)
  - — *PowerPC 603e RISC Microprocessor Family: PID6-603e Hardware Specifications*: MPC603EEC/D (Motorola order #) and G522-0268-00 (IBM order #)
  - — *PowerPC 603e RISC Microprocessor Family: PID7V-603e Hardware Specifications*: MPC603E7VEC/D (Motorola order #) and G522-0267-00 (IBM order #)
  - — *PowerPC 604 RISC Microprocessor Hardware Specifications*: MPC604EC/D (Motorola order #) and MPR604HSU-02 (IBM order #)
  - — *PowerPC 604e RISC Microprocessor Family: PID9V-604e Hardware Specifications*: MPC604E9VEC/D (Motorola order #) and SA14-2054-00 (IBM order #)
- Technical Summaries—Each PowerPC implementation has a technical summary that provides an overview of its features. This document is roughly the equivalent to the overview (Chapter 1) of an implementation's user's manual. Technical summaries are available for the 601, 602, 603, 603e, 604, and 604e as well as the following:
  - — *PowerPC 620™ RISC Microprocessor Technical Summary*: MPC620/D (Motorola order #) and SA14-2069-01 (IBM order #)

- *PowerPC Microprocessor Family: The Programmer's Reference Guide* is a concise reference that includes the register summary, memory control model, exception vectors, and the PowerPC instruction set.
  MPCPRG/D (Motorola order #) and MPRPPCPRG-01 (IBM order #)
- *PowerPC Microprocessor Family: The Programmer's Pocket Reference Guide*: This foldout card provides an overview of the PowerPC registers, instructions, and exceptions for 32-bit implementations.
  MPCPRGREF/D (Motorola order #) and SA14-2093-00 (IBM order #)
- Application notes—These short documents contain useful information about specific design issues useful to programmers and engineers working with PowerPC processors.
- Documentation for support chips—These include the following:
  - *MPC105 PCI Bridge/Memory Controller User's Manual*:
    MPC105UM/AD (Motorola order #)
  - *MPC106 PCI Bridge/Memory Controller User's Manual*:
    MPC106UM/AD (Motorola order #)

Additional literature on PowerPC implementations is being released as new processors become available. For a current list of PowerPC documentation, refer to the world-wide web at http://www.mot.com/powerpc/ or at http://www.chips.ibm.com/products/ppc.

Conventions

This document uses the following notational conventions:

ACTIVE_HIGH	Names for signals that are active high are shown in uppercase text without an overbar.
$\overline{\text{ACTIVE\_LOW}}$	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 AP[0–3] (address bus parity signals) and TT[0–4] (transfer type signals) are referred to as asserted when they are high and negated when they are low.
SYS- (or $\overline{\text{SYS-}}$)	This prefix is used to distinguish signals coming from the system bus from one on the 60x processor that otherwise have the same name.
mnemonics	Instruction mnemonics are shown in lowercase bold.
OPERATIONS	Address-only bus operations that are named for the instructions that generate them are identified in uppercase letters, for example, ICBI, SYNC, TLBSYNC, and EIEIO operations.
italics	Italics indicate variable command parameters, for example, **bcctr***x*
0x0	Prefix to denote hexadecimal number
0b0	Prefix to denote binary number

rA, **r**B	Instruction syntax used to identify a source GPR
rA|0	The contents of a specified GPR or the value 0
rD	Instruction syntax used to identify a destination GPR
frA, **fr**B, **fr**C	Instruction syntax used to identify a source FPR
frD	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.

Acronyms and Abbreviations

Table i contains acronyms and abbreviations that are used in this document.

Table i. Acronyms and Abbreviated Terms

Term	Meaning
ALU	Arithmetic logic unit
ASR	Address space register
BAT	Block address translation
BIST	Built-in self test
BIU	Bus interface unit
BUID	Bus unit ID
COP	Common on-chip processor
CR	Condition register
CTR	Count register
DABR	Data address breakpoint register
DAR	Data address register
DBAT	Data BAT
DEC	Decrementer (register)
DSISR	Register used for determining the source of a DSI exception
DTLB	Data translation look-aside buffer
EA	Effective address
EAR	External access register
ECC	Error checking and correction

Table i. Acronyms and Abbreviated Terms (Continued)

Term	Meaning
ETP	Extended transfer protocol
EX	Exclusive state (includes shared, S, and exclusive unmodified, E)
FIFO	First-in, first-out
FPR	Floating-point register
FPSCR	Floating-point status and control register
FPU	Floating-point unit
GPR	General-purpose register
HID*n*	Hardware implementation-dependent register
IABR	Instruction address breakpoint register
IBAT	Instruction BAT
IEEE	Institute of Electrical and Electronics Engineers
ITLB	Instruction translation look-aside buffer
JTAG	Joint Test Action Group
L2	Secondary cache
LR	Link register
LRS	**lwarx** reservation set
LRU	Least recently used
LSB	Least-significant byte
lsb	Least-significant bit
MESI	Modified/exclusive/shared/invalid—cache coherency protocol
MMCR*n*	Monitor mode control register *n*
MMU	Memory management unit
MSB	Most-significant byte
msb	Most-significant bit
MSR	Machine state register
NaN	Not a number
No-op	No operation
OEA	Operating environment architecture
PID	Processor identification tag
PLL	Phase-locked loop
PMC*n*	Performance monitor control (register) *n*
PMI	Performance monitor interrupt

Table i. Acronyms and Abbreviated Terms (Continued)

Term	Meaning
PTE	Page table entry
PTEG	Page table entry group
PVR	Processor version register
RdA	Read atomic
RISC	Reduced instruction set computing/computer
RTL	Register transfer language
RWITM	Read with intent to modify
RWITMA	Read with intent to modify atomic
SBR	Single-beat read
SBRA	Single-beat read atomic
SBW	Single-beat write
SDR1	Register that specifies the page table base address for virtual-to-physical address translation
SLB	Segment lookaside buffer
SPR	Special-purpose register
SPRG*n*	Registers available for general purposes
SR	Segment register
SRR0	(Machine status) save/restore register 0
SRR1	(Machine status) save/restore register 1
TAP	Test access port controller
TB	Time base register
TLB	Translation lookaside buffer
UISA	User instruction set architecture
VEA	Virtual environment architecture
WWF	Write with flush
WWFA	Write with flush atomic
WWK	Write with kill
XATC	Extended address transfer code
XER	Register used for indicating conditions such as carries and overflows for integer operations

Chapter 1
Overview

This chapter gives an overview of the bus interface common to the 60x microprocessors. It describes the operation and features of this interface, lists all the microprocessor signals, shows differences between the three microprocessors in the use and number of signals, and defines various operational subsets of signals, and in particular identifies those that are required for any system.

This bus description documents the current operations and system implementation information for the following PowerPC™ processors:

- PowerPC 601™ processor. The 601 is the first PowerPC processor and is designed for desktop, server, and workstation implementations and is designed to support implementation in multiprocessing systems.
- PowerPC 603™ processors. References to the 603 include the PowerPC 603e™ processors unless otherwise specified. The 603 family for processors is optimized for implementation in low-power systems, and includes bus support for power management, but provides less support for multiprocessing than either the 601 or 604 families of processors.
- PowerPC 604™ processors. References to the 604 include the PowerPC 604e™ processors unless otherwise specified. The 604 family of processors is designed for implementation in desktop, workstation, and server systems and provides extensive support both for multiprocessing and for power management.

Although this book can be used as a general guide for the PowerPC 602™ processor, it does not include descriptions of the specific operations that are unique to that processor.

1.1 PowerPC 60x Microprocessor Interface

The 601, 603, and 604 support a range of systems, including low-power and notebook machines, low-cost desktop personal computers, high-performance workstations, and multiprocessor server systems. To meet those needs, the interface to these processors was defined with a minimum set of functions and 32-bit or 64-bit data bus modes, as well as optional performance and function enhancement signals and modes.

The 60x bus definition is based on the Motorola 88110 bus definition. This interface runs synchronous to the system clock. Inputs are sampled at and outputs are driven from the rising edge of the system clock. This processor bus provides two transfer protocols:

- The basic transfer protocol is used to access normal memory segments. This protocol supports transfer of any number of 32- or 64-bit continuous bytes within an aligned double word to any address in the 32-bit address range. It also supports the use of burst transfers and multiple-beat transfers that transfer up to 64 bits of data during each beat.
- Direct-store operations (elsewhere referred to as extended transfer protocol, or ETP) use a slightly different protocol for accessing the direct-store segments as defined in the PowerPC architecture. This protocol provides an extended address, support for split transactions, and a positive reply for each transaction. The synchronous nature of this protocol limits its performance compared to the basic protocol, but provides for enhanced error recovery. This functionality is now considered optional to the PowerPC architecture and is not supported in all PowerPC processors, for example in second generation processors in the 603 family.

The PowerPC architecture includes the following:

- An address space shared by all processing elements in the system
- A weakly-ordered memory model that allows processors to improve performance by reordering loads and stores
- A set of explicit cache management and translation lookaside buffer (TLB) management instructions that can be broadcast by the processor to allow software control of caches in a single- or multiple-processor environment
- Instructions for synchronizing operations between different processors

Each processor has a separate address and data bus. In the basic transfer protocol, these separate buses may be used to implement coupled address and data tenures typical of low-end personal computers, or they may be used to implement advanced features such as address pipelining, which allows a new bus transaction to begin before the current transaction has finished, and split-bus transactions, which allows the address bus and data bus to have separate masters at the same time.

The processor bus supports full write-back cache coherency, bus snooping, transaction retry, and snoop copy-back operations, although it should be noted that each processor may not implement all such features and that some processors may implement such features in a more sophisticated manner.

The bus defines signals that support access from multiple masters, including other processors and devices, with arbitration provided by the system implementation.

1.2 PowerPC System Block Diagram

Figure 1-1 shows the processor bus in a typical system design. The bus provides a communications layer between one or more PowerPC processors, the memory controller, the system-provided arbiter, a bridge to an expansion bus for system I/O, and optionally a high-speed I/O adaptor such as a graphics adaptor. The processor component may have an external cache. This bus supports cache implementations that are in-line or lookaside and that are write-through or write-back.

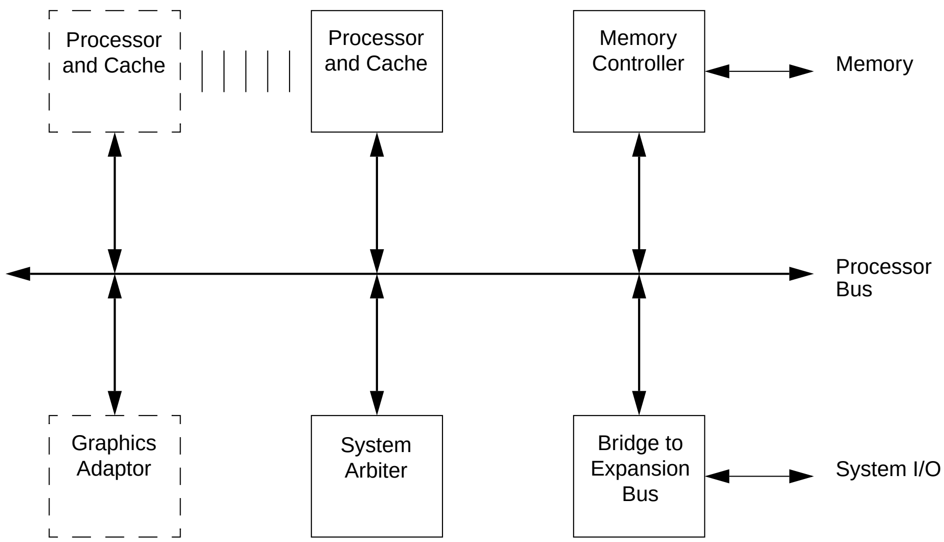

Figure 1-1. Typical System Diagram with Processor Bus

1.3 Processor Bus Features

The processor bus provides high performance and adaptability to various system environments. Features of this bus include the following:

- Bus operation greater than 66 MHz with 601, 603, and 604
- Maintenance of coherency for external cache
- Support for split transactions
- Support for pipelined bus transactions
- Support for address-only bus transactions used primarily for cache control
- Support for multiprocessor configurations
- Optional performance enhancements

Note that not all processors in the 60x family may support all available features.

1.4 Bus Interface Signals

Figure 1-2 shows the PowerPC processor view of all signals. Signals that are part of the basic set are shown as solid lines. Those that are optional and provide enhanced functions or performance are shown as dashed lines.

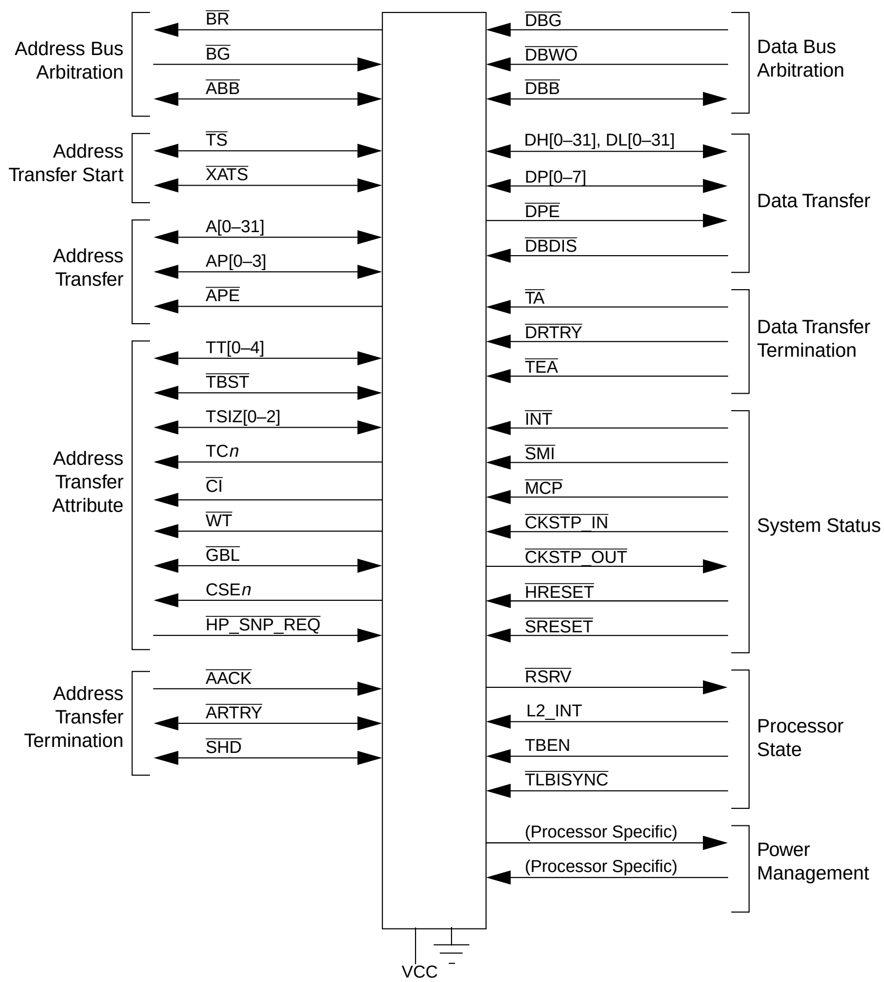

Figure 1-2. Processor Bus Signals

The signal groupings in Figure 1-2 are described in Table 1-1.

Table 1-1. 60x Signal Groupings

Signal Group	Functionality
Address bus arbitration	Used to arbitrate for the address bus
Address transfer start	Indicate that the bus master has begun a transaction on the address bus
Address transfer	Used to transfer the address and to ensure the integrity of the transfer
Address transfer attribute	Provide information about the type of transfer
Address transfer termination	Indicate the end of the address phase or the need to repeat the address phase
Data bus arbitration	Used to arbitrate for the data bus mastership
Data transfer	Used to transfer the data and ensure the integrity of the transfer
Data transfer termination	Indicate the end of a data transfer or that the data phase should be repeated
System status	Indicate interrupts and system resets
Processor state	Used to manage the processor state
Power management	Provide a means for a processor and system to cooperate in power management operations. The specific signals for each processor are identified in Table 1-2.

The evolution of the processors and the target market for the processors dictated that some of these signals are not supported on some processors, have different pin counts, or may operate differently on some processors. Those differences are described in Section 2.12, “Summary of Signal Differences.”

Table 1-2 briefly describes each signal function and provides a reference to the detailed description of the signal state meanings and timing considerations in Chapter 2, “Signal Descriptions.”

Table 1-2. Use and Reference for Bus Signals

Signal	I	O	Function	Application				Section
				Basic	L2	MP	Opt.	
Address Bus Arbitration Signals								
Bus request ($\overline{BR}$)		√	Requests mastership of the bus	√				2.1.1
Bus grant ($\overline{BG}$)	√		Indicates bus ownership if properly qualified			√		2.1.2
Address bus busy ($\overline{ABB}$)	√	√	Indicates whether the address bus is busy				√	2.1.3 2.1.4
Address Transfer Start Signals								
Transfer start ($\overline{TS}$)	√	√	Indicates that the master has begun a transaction to memory	√				2.2.1 2.2.2
Extended transfer start ($\overline{XATS}$)	√	√	Indicates that the master has begun a transaction to a direct-store address				√	2.2.3 2.2.4

Table 1-2. Use and Reference for Bus Signals (Continued)

Signal	I	O	Function	Application				Section
				Basic	L2	MP	Opt.	
Address Transfer Signals								
Address bus (A[0–31])	√	√	Indicates the real address of the bus transaction	√				2.3.1–2.3.4
Address parity (AP[0–3])	√	√	Gives odd parity for each address byte				√	2.3.5 2.3.6
Address parity error ($\overline{\text{APE}}$)		√	Indicates detection of address bus parity error				√	2.3.7
Address Transfer Attribute Signals								
Transfer type (TT[0–4])	√	√	Indicates the type of transfer in progress	√				2.4.1 2.4.2
Transfer burst ($\overline{\text{TBST}}$)	√	√	Indicates that a burst transfer is in progress	√				2.4.3 2.4.4
Transfer size (TSIZ[0–2])	√	√	Indicates the size in bytes of transfer in progress	√				2.4.5 2.4.6
Transfer code (TC*n*)		√	Gives information about the transaction for external cache operations		√			2.4.7
Cache inhibit ($\overline{\text{CI}}$)		√	Indicates whether a transfer can be cached		√			2.4.8
Write-through ($\overline{\text{WT}}$)		√	Indicates whether a transaction is write-through		√			2.4.9
Global ($\overline{\text{GBL}}$)	√	√	Indicates that a transaction is global and that data coherence is required			√		2.4.10 2.4.11
Cache set element (CSE*n*])		√	Represents the cache replacement set element of the current transaction				√	2.4.12
High-priority snoop request ($\overline{\text{HP\_SNP\_REQ}}$)	√		601 only: Used to indicate when the reserved position in the write queue is needed for a push operation resulting from a snoop hit				√	2.4.13
Address Transfer Termination Signals								
Address acknowledgment ($\overline{\text{AACK}}$)	√		Indicates that the address portion of a transaction is complete	√				2.5.1
Address retry ($\overline{\text{ARTRY}}$)	√	√	Asserted when the address tenure must be retried	√				2.5.2 2.5.3
Shared ($\overline{\text{SHD}}$)	√	√	As an output, indicates the master hit a shared cache block. As an input, indicates the incoming cache block should be marked shared (S)			√		2.5.4 2.5.5
Data Bus Arbitration Signals								
Data bus grant ($\overline{\text{DBG}}$)	√		Indicates the master may, with proper qualification, assume ownership of the data bus				√	2.6.1
Data bus write only ($\overline{\text{DBWO}}$)	√		Indicates an outstanding write may precede a pipeline read				√	2.6.2

Table 1-2. Use and Reference for Bus Signals (Continued)

Signal	I	O	Function	Application				Section
				Basic	L2	MP	Opt.	
Data bus busy ($\overline{\text{DBB}}$)	√	√	Indicates the data bus is busy				√	2.6.4
Data Transfer Signals								
Data bus (DH[0–31];DL[0–31])	√	√	Represents the data being transferred	√				2.7.1 2.7.2
Data bus parity (DP[0–7])	√	√	Represents odd parity for the data bytes				√	2.7.3 2.7.4
Data parity error ($\overline{\text{DPE}}$)		√	Forces processor to put data bus in high-impedance state during a write data tenure; other processor operations are unaffected.				√	2.7.5
Data bus disable ($\overline{\text{DBDIS}}$)	√		Indicates to the processor that a write transaction should be stopped				√	2.7.6
Data Transfer Termination Signals								
Transfer acknowledge ($\overline{\text{TA}}$)	√		Indicates that a single-beat data transfer completed successfully	√				2.8.1
Data retry ($\overline{\text{DRTRY}}$)	√		Invalidates read data sent to processor with $\overline{\text{TA}}$ in the previous cycle. On hard reset, is used to configure some alternate modes.				√	2.8.2
Transfer error acknowledgment ($\overline{\text{TEA}}$)	√		Indicates that a bus error occurred	√				2.8.3
System Status Signals								
Interrupt ($\overline{\text{INT}}$)	√		Indicates an external interrupt to the processor	√				2.9.1
System management interrupt ($\overline{\text{SMI}}$)	√		Indicates a system management interrupt to the processor				√	2.9.2
Machine check ($\overline{\text{MCP}}$)	√		Indicates a machine check exception				√	2.9.3
Checkstop input ($\overline{\text{CKSTP\_IN}}$)	√		Indicates the processor must stop operation (checkstop)				√	2.9.4
Checkstop output ($\overline{\text{CKSTP\_OUT}}$)		√	Indicates the processor has detected a checkstop condition				√	2.9.5
Hard reset ($\overline{\text{HRESET}}$)	√		Initiates a hard reset exception	√				2.9.6
Soft reset ($\overline{\text{SRESET}}$)	√		Initiates a soft reset exception				√	2.9.7
Processor State Signals								
Reservation ($\overline{\text{RSRV}}$)		√	Indicates that a reservation generated by a **lwarx** instruction exists in the processor		√			2.10.1
External cache intervention (L2_INT)	√		Indicates intervention from other bus masters		√			2.10.2
Time base enable (TBEN)	√		Indicates the time base should continue clocking				√	2.10.3

Table 1-2. Use and Reference for Bus Signals (Continued)

Signal	I	O	Function	Application				Section
				Basic	L2	MP	Opt.	
TLBI synchronization ($\overline{\text{TLBISYNC}}$)	√		603: Indicates execution should stop after a **tlbsync** instruction				√	2.10.4
Power Management Signals								
Quiescent request (QUIESC_REQ)		√	601: Indicates the 601 is ready to enter a soft stop state				√	2.11.1
System quiesced ($\overline{\text{SYS\_QUIESC}}$)	√		601: Indicates to the 601 that the system is ready for the soft stop state				√	2.11.2
Resume (RESUME)	√		601: Indicates to resume normal processing				√	2.11.3
Quiescent request ($\overline{\text{QREQ}}$)		√	603: Requests all bus activity requiring snooping to pause				√	2.11.4
Quiescent acknowledge ($\overline{\text{QACK}}$)	√		603: Indicates all bus activity that requires snooping has paused				√	2.11.5
Halted (HALTED)		√	604: Indicates the 604 has entered a low-power state				√	2.11.6
Run (RUN)	√		604: Indicates to keep snooping in low-power state				√	2.11.7

The four columns under the heading, ‘application’ in Table 1-2 are described as follows:

- Basic operations—Signals in the column labeled ‘Basic’ in Table 1-2 are required to build a simple, uniprocessor system with one bus, no external cache, and no support for bus pipelining. Within this set of signals, TT4 is optional and, as shown in Table 3-1, is used to identify additional transactions that can be snooped.
- L2 cache support—Signals in the ‘L2’ column in Table 1-2 are required to support an external cache. For example, TC[0–2] are necessary to indicate the type of transaction. However, some of these signals are optional for some system designs. For instance, a write-through external cache would not need the $\overline{\text{WT}}$ signal, or a cache that responds only to burst operations would not need the $\overline{\text{CI}}$ signal.
- Multiprocessor support—The signals $\overline{\text{GBL}}$, $\overline{\text{SHD}}$, and $\overline{\text{BG}}$, listed in the ‘MP’ column, support memory coherency for systems with masters other than the processor including multiprocessor systems. Chapter 4, “Memory Coherency,” provides detailed information on memory coherency. The $\overline{\text{BG}}$ signal would be used to assign the bus in systems in which the bus is shared by multiple devices, in which case the $\overline{\text{GBL}}$ signal would be interconnected between all devices to ensure cache coherency. Optionally the $\overline{\text{GBL}}$ and $\overline{\text{SHD}}$ signals could be connected to a bridge for snooping. The bridge would set it to a known state.

- Enhanced operation—The signals in the ‘optional’ column (labeled ‘Opt.’) provide additional functions and performance enhancements, including the following:
  - — Address bus arbitration—The $\overline{\text{ABB}}$ signal is optional because it can be derived from other signals if all masters refrain from taking the address bus from the beginning of $\overline{\text{TS}}$ through the end of $\overline{\text{AACK}}$.
  - — Parity signals—Address and data parity signals, AP[0–3], $\overline{\text{APE}}$, DP[0–7], and $\overline{\text{DPE}}$, are optional. Low-end and low-cost systems, for example some personal computers, may not check or generate parity on either addresses or data. Higher-cost systems may generate parity for only data. High-end systems may generate parity for both addresses and data and may use the processor-generated indications of parity errors to control other system components.
  - — Address pipeline—Separate data bus arbitration and granting signals allow independent operation of address and data bus tenures. The $\overline{\text{DBWO}}$ signal allows the processor to run a data bus tenure for an outstanding write address even if a read address is pipelined before it.
  - — Data retry—The $\overline{\text{DRTRY}}$ signal is used to support speculative forwarding of data.
  - — Interrupts—Some processors have an additional system management interrupt in addition to the hardware interrupt defined by the PowerPC architecture. This interrupt is signaled by asserting the $\overline{\text{SMI}}$ signal.
  - — Soft reset—The soft reset signal, $\overline{\text{SRESET}}$, is used to initiate a soft reset, a type of system reset that is not defined by the PowerPC architecture but implemented on most PowerPC processors.
  - — Power management—Some processors support the use of the QUIESC_REQ, $\overline{\text{SYS\_QUIESC}}$, RESUME, $\overline{\text{QREQ}}$, $\overline{\text{QACK}}$, RUN, and HALTED signals to control power consumption allowing power to be removed from certain portions of the processor when not in use.
  - — 603 address translation—Because the 603 is optimized for low-power, uniprocessor systems, hardware support is not provided for table search operations.
  - — Extended transfer start—The $\overline{\text{XATS}}$ signal supports the direct-store accesses. Chapter 7, “Direct-Store Interface,” describes the direct-store interface and the effect this protocol has on TT[0–4], $\overline{\text{TBST}}$, TSIZ[0–2], and A[0–31].

Chapter 2
Signal Descriptions

This chapter describes the external signals used by the PowerPC 601, PowerPC 603, and PowerPC 604 processors, identifying both the set of signals that are common to all 60x processors as well as indicating characteristics of individual processor implementations. 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 that the descriptions in this chapter are intended to provide a quick summary of signal functions. Subsequent chapters describe the operation of many of these signals in greater detail, both with respect to how individual signals function and how groups of signals interact.

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 AP[0–3] (address bus parity signals) and TT[0–4] (transfer type signals) are referred to as asserted when they are high and negated when they are low.

The clock, power, and test signals are not described in this document. Refer to the user's manual for the particular processor for this information.

The bus signal descriptions in this chapter are grouped by the categories shown in Figure 1-2. The section names in this chapter correspond to those groups as defined in Section 1.4, "Bus Interface Signals." The sections describe state and timing descriptions for each signal and indicate if a signal is an input or an output with respect to a PowerPC processor. If a signal is both, the output characteristics are described first. The description is from the perspective of the processor; no attempt is made to describe these signals as an arbiter, slave, or target would see them.

The differences between how signals are implemented on different processors is summarized in Section 2.12, "Summary of Signal Differences."

2.1 Address Bus Arbitration Signals

To access the address bus, a device must request and gain bus mastership. Bus arbitration signals are a collection of input and output signals bus devices use to request the address bus, recognize when the request is granted, and indicate to other devices when mastership is granted. For detailed descriptions and timing diagrams that show how these signals interact, see Section 3.2.1, “Address Bus Arbitration.”

2.1.1 Bus Request ($\overline{\text{BR}}$)—Output

Following are state and timing descriptions for the bus request ($\overline{\text{BR}}$) as an output signal.

State Meaning

Asserted—A device is requesting address bus mastership. $\overline{\text{BR}}$ can be asserted for one or more cycles and then deasserted due to an internal cancellation of the bus request (for example, due to the loss of a memory reservation).

Negated—No device is requesting the address bus. The device may have no bus operation pending, it may be parked, or the $\overline{\text{ARTRY}}$ input was asserted on the previous bus clock cycle.

Timing Comments

Assertion—A bus transaction is needed and the device does not have a qualified bus grant. This may occur even if the maximum (two for the 601and 603, three for the 604) possible pipeline accesses have occurred. For the 603, $\overline{\text{BR}}$ is asserted for one cycle during execution of a **dcbz** or of a load instruction that hits in the touch load buffer.

Negation—Occurs for at least one bus clock cycle after an accepted, qualified bus grant (see $\overline{\text{BG}}$ and $\overline{\text{ABB}}$), even if another transaction is pending. It is also negated for at least one cycle after the assertion of $\overline{\text{ARTRY}}$, unless that processor caused the assertion of $\overline{\text{ARTRY}}$ to perform a cache block push for that snoop operation.

2.1.2 Bus Grant ($\overline{\text{BG}}$)—Input

Following are state and timing descriptions for the bus grant ($\overline{\text{BG}}$) as an input signal.

State Meaning

Asserted—The device may, with the proper qualification, assume mastership of the address bus. A qualified bus grant occurs in a given cycle when the following conditions are met:

- $\overline{\text{BG}}$ is asserted.
- No address cycle is in progress (as marked by $\overline{\text{ABB}}$ or the $\overline{\text{TS}}$-through-$\overline{\text{AACK}}$ interval).
- $\overline{\text{ARTRY}}$ is negated and was negated on the previous cycle (not considered on 601).

The assertion of $\overline{\text{BR}}$ is not required for the qualified bus grant (for example, the parked case).

Note that the 601 recognizes a qualified bus grant on the cycle after $\overline{\text{AACK}}$ even if $\overline{\text{ARTRY}}$ is asserted as long as the 601 is asserting $\overline{\text{ARTRY}}$ and has exclusive ownership of the data associated with the snoop that caused the $\overline{\text{ARTRY}}$. The $\overline{\text{ABB}}$ and $\overline{\text{ARTRY}}$ signals are driven by the bus master. If the processor is parked, $\overline{\text{BR}}$ need not be asserted for the qualified bus grant.

Negated—The device is not the next potential address bus master.

Timing Comments Assertion—May occur at any time to indicate the device is free to use the address bus. After the processor gains bus mastership, it does not check for a qualified bus grant again until the cycle in which the address bus tenure completes (assuming it has another transaction to run). The processor does not accept a $\overline{\text{BG}}$ in the cycles between the assertion of any $\overline{\text{TS}}$ or $\overline{\text{XATS}}$ through to the assertion of $\overline{\text{AACK}}$.

Negation—May occur at any time to indicate the device cannot use the bus. However, the device still assumes mastership on the bus clock cycle $\overline{\text{BG}}$ is negated because, in the previous cycle, $\overline{\text{BG}}$ indicated to the device that it could take mastership (if qualified).

2.1.3 Address Bus Busy ($\overline{\text{ABB}}$)—Output

Following are state and timing descriptions for address bus busy ($\overline{\text{ABB}}$) as an output signal.

State Meaning Asserted—The device is the address bus master.

Negated—The device is not using the address bus. If $\overline{\text{ABB}}$ is negated in the bus clock cycle after a qualified bus grant, the device did not accept mastership, even if $\overline{\text{BR}}$ was asserted. This can occur if a potential transaction is aborted internally before it started.

Timing Comments Assertion—Occurs on the bus clock cycle after a qualified bus grant that is accepted by the device (see Negated).

Negation—Occurs for a fraction of the bus clock cycle after $\overline{\text{AACK}}$ is asserted. If $\overline{\text{ABB}}$ is negated in the bus clock cycle after a qualified $\overline{\text{BG}}$, the device did not accept mastership, even if $\overline{\text{BR}}$ was asserted.

High Impedance—Occurs during a fractional portion of the bus cycle in which $\overline{\text{ABB}}$ is negated. $\overline{\text{ABB}}$ is guaranteed by design to be high impedance by the end of the cycle in which it is negated. For specific information, see the particular processor’s user’s manual.

2.1.4 Address Bus Busy ($\overline{\text{ABB}}$)—Input

Following are state and timing descriptions for $\overline{\text{ABB}}$ as an input signal.

State Meaning Asserted—The address bus is being used by another master, which effectively keeps the device from assuming address bus ownership, regardless of the $\overline{\text{BG}}$ input. The processor will not take the address bus for the sequence of cycles beginning with $\overline{\text{TS}}$ and ending with $\overline{\text{AACK}}$, which effectively makes $\overline{\text{ABB}}$ optional if other bus masters respond in the same way as the processor.

Negated—The address bus is not owned by another bus device and is available when accompanied by a qualified bus grant.

Timing Comments Assertion—May occur when the other devices must be prevented from using the address bus (and the processor is not currently asserting $\overline{\text{ABB}}$).

Negation—May occur whenever the master can use the address bus.

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 ($\overline{\text{TS}}$) signal identifies the operation as a memory transaction; extended address transfer start ($\overline{\text{XATS}}$) identifies the transaction as a direct-store operation. For detailed information about how $\overline{\text{TS}}$ and $\overline{\text{XATS}}$ interact with other signals, refer to Section 3.2.2, “Address Transfer,” and Chapter 7, “Direct-Store Interface,” respectively.

2.2.1 Transfer Start ($\overline{\text{TS}}$)—Output

Following are state and timing descriptions for transfer start ($\overline{\text{TS}}$) as an output signal.

State Meaning Asserted—The master has begun a memory bus transaction and 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 $\overline{\text{TS}}$ output is an address-only operation).

Negated—Has no special meaning. However, $\overline{\text{TS}}$ is negated throughout an entire direct-store address tenure.

Timing Comments Assertion—Coincides with the assertion of $\overline{\text{ABB}}$.

Negation—Occurs one bus clock cycle after $\overline{\text{TS}}$ is asserted.

High Impedance—(601 and 603) Occurs one bus clock cycle after $\overline{\text{TS}}$ is negated, which is coincident with the negation of $\overline{\text{ABB}}$.

High Impedance—(604) Occurs one bus clock cycle after the negation of $\overline{\text{TS}}$. For the 604, the $\overline{\text{TS}}$ negation is only one bus cycle long, regardless of the $\overline{\text{TS}}$-to-$\overline{\text{AACK}}$ delay.

2.2.2 Transfer Start ($\overline{\text{TS}}$)—Input

Following are state and timing descriptions for $\overline{\text{TS}}$ as an input signal.

State Meaning Asserted—Another master began a bus transaction and the address bus and transfer attribute signals are valid for snooping (see $\overline{\text{GBL}}$).

Negated—No bus transaction is occurring.

Timing Comments Assertion—May occur any time outside the address tenure window: either the interval that includes the cycle of a previous $\overline{\text{TS}}$ assertion through the cycle after $\overline{\text{AACK}}$ or the cycles in which $\overline{\text{ABB}}$ is asserted for a previous address tenure, whichever is greater.

Negation—Must occur one bus clock cycle after $\overline{\text{TS}}$ is asserted.

2.2.3 Extended Address Transfer Start ($\overline{\text{XATS}}$)—Output (Direct-Store)

Following are state and timing descriptions for extended address transfer start ($\overline{\text{XATS}}$) as an output signal.

State Meaning Asserted—The master began a direct-store operation and the first address cycle is valid. When asserted with the appropriate extended address transfer code (XATC) signals, it is also an implied data bus request for certain direct-store operations (unless it is an address-only operation).

Negated—Has no special meaning; however, $\overline{\text{XATS}}$ remains negated throughout an entire memory address tenure.

Timing Comments Assertion—Coincides with the assertion of $\overline{\text{ABB}}$.
Negation—Occurs one bus clock cycle after the assertion of $\overline{\text{XATS}}$.

High Impedance—(601 and 603) Occurs one bus clock cycle after the negation of $\overline{\text{XATS}}$, which coincides with the negation of $\overline{\text{ABB}}$.

High Impedance—(604) Occurs one bus clock cycle after the negation of $\overline{\text{XATS}}$. For the 604, $\overline{\text{XATS}}$ negation is only one bus cycle long, regardless of the $\overline{\text{XATS}}$-to-$\overline{\text{AACK}}$ delay.

2.2.4 Extended Address Transfer Start ($\overline{\text{XATS}}$)—Input (Direct-Store)

Following are state and timing descriptions for $\overline{\text{XATS}}$ as an input signal.

State Meaning Asserted—The master must check for a direct-store operation reply.

Negated—There is no need to check for a direct-store reply.

Timing Comments Assertion—May occur at any time outside of the cycles that define the window of an address tenure. This window is marked by either the interval that includes the cycle of a previous $\overline{\text{XATS}}$ assertion through the cycle after $\overline{\text{AACK}}$ or by the cycles in which $\overline{\text{ABB}}$ is asserted for a previous address tenure, whichever is greater.

Negation—Must occur one bus clock cycle after $\overline{\text{XATS}}$ is asserted.

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 detailed descriptions of how these signals interact, see Section 3.2.2, “Address Transfer.”

2.3.1 Address Bus (A[0–31])—Output (Memory Operations)

Following are state and timing descriptions for the address bus (A[0–31]) as output signals during memory operations.

State Meaning Asserted/Negated—Represents the physical address of the data to be transferred. On burst transfers, the address bus presents the double-word–aligned address (quad-word–aligned for the 601) with the critical data that missed the cache on a read operation, or the first double word of the cache clock on a write operation. Note that the address output during burst operations is not incremented.

Timing Comments Assertion/Negation—Occurs on the bus clock cycle after a qualified bus grant (coincides with assertion of $\overline{\text{ABB}}$ and $\overline{\text{TS}}$).

High Impedance—Occurs one bus clock cycle after $\overline{\text{AACK}}$ is asserted.

2.3.2 Address Bus (A[0–31])—Input (Memory Operations)

Following are state and timing descriptions for A[0–31] as input signals for memory operations.

State Meaning Asserted/Negated—Carries the address of a snoop operation.

Timing Comments Assertion/Negation—Must occur on the same bus clock cycle as the assertion of $\overline{\text{TS}}$; is sampled by the processor only on this cycle.

2.3.3 Address Bus (A[0–31])—Output (Direct-Store Operations)

Following are state and timing descriptions for A[0– 31] as output signals for direct-store operations.

State Meaning Asserted/Negated—For direct-store operations from this device, the address tenure consists of two packets (each requiring a bus cycle). For packet 0, these signals convey control and tag information. For packet 1, they represent the physical address of the data to be transferred. For reply operations to other devices, the address bus carries control, status, and tag information.

Timing Comments Assertion/Negation—An address tenure consists of two beats. The first occurs on the bus clock cycle after a qualified bus grant, coinciding with $\overline{\text{XATS}}$. The address bus makes a transition to the second beat on the next bus clock cycle.

High Impedance—Occurs the bus clock cycle after $\overline{\text{AACK}}$ is asserted.

2.3.4 Address Bus (A[0–31])—Input (Direct-Store Operations)

Following are state and timing descriptions for A[0–31] as input signals for direct-store operations.

State Meaning Asserted/Negated—When the processor receiving A[0–31] signals is not the master, it snoops (and checks address parity) on only the first address beat of all direct-store operations for I/O reply operations whose receiver tags match the processor identification (PID) tag. See Section 7.1, "Direct-Store Transaction Protocol Details."

Timing Comments Assertion/Negation—The first beat of the I/O transfer address tenure coincides with $\overline{\text{XATS}}$, with the second address beat on the next cycle.

2.3.5 Address Bus Parity (AP[0–3])—Output

Following are state and timing descriptions for the address bus parity signals (AP[0–3]) as output signals.

State Meaning Asserted/Negated—Represents one bit of odd parity for each of four address bus bytes. Odd parity means an odd number of bits, including the parity bit, are driven high. Signal assignments are as follows:

AP0 A[0–7]
AP1 A[8–15]
AP2 A[16–23]
AP3 A[24–31]

For more information, see Section 3.2.2.1, "Address Bus Parity."

Timing Comments Assertion/Negation/High Impedance—The same as A[0–31].

2.3.6 Address Bus Parity (AP[0–3])—Input

Following are state and timing descriptions for AP[0–3] as input signals.

State Meaning Asserted/Negated—Represents one bit of odd parity for each of four address bus bytes for snooping and direct-store operations. Depending on MSR[ME] and various HID0 bits, detecting even parity either causes the processor to enter the checkstop state or take a machine check exception. If address parity check is enabled in HID0, detection of even parity unconditionally causes a checkstop in the 601. (See the $\overline{\text{APE}}$ signal description.)

Timing Comments Assertion/Negation—The same as A[0–31].

2.3.7 Address Parity Error ($\overline{\text{APE}}$)—Output

Following are state and timing descriptions for the address parity error ($\overline{\text{APE}}$) output signal. Note that $\overline{\text{APE}}$ is an open-drain type output and requires an external pull-up resistor to assure proper deassertion.

State Meaning Asserted—The processor detected incorrect address bus parity on a snoop for a transaction type it recognizes and can respond to, such as the first address beat of a direct-store operation. The 603 does not assert $\overline{\text{APE}}$ if address parity checking is disabled.

Negated—The processor did not detect even address bus parity.

Timing Comments Assertion—Occurs the second bus clock cycle after $\overline{\text{TS}}$ or $\overline{\text{XATS}}$ is asserted.

High Impedance—Occurs the third bus clock cycle after $\overline{\text{TS}}$ or $\overline{\text{XATS}}$ is asserted.

2.4 Address Transfer Attribute Signals

The transfer attribute signals further characterize the transfer—indicating such things as the transfer size, whether it is a read or write, and whether it is a burst or single-beat transfer. For a detailed description of how these signals interact, see Section 3.2.2, "Address Transfer." Some signals that function one way for memory operations may work differently for direct-store accesses; see Chapter 7, "Direct-Store Interface."

2.4.1 Transfer Type (TT[0–4])—Output

Following are state and timing descriptions for the transfer type signals (TT[0–4]) as output signals.

State Meaning Asserted/Negated—Table 2-1 defines the transactions identified by the TT[0–4] signals. The table gives the type of transaction the type of data transferred, the source or cause of the transfer, and the processors that support these transaction types as master or when snooping. Some codes in this table are reserved. Notice that the encoding has been chosen to simplify decoding. For example, TT1 is generally zero for writes and one for reads or TT3 is generally zero for an address-only operation.

For a full description of coherency actions, see Appendix E, "Coherency Action Tables."

For direct-store operations, these signals are part of the extended address transfer code (XATC) along with TSIZ*n* and $\overline{\text{TBST}}$:

XATC(0–7) = TT(0–3)||$\overline{\text{TBST}}$||TSIZ(0–2).

TT4 is driven negated as an output on the 601.

Timing Comments Assertion/Negation/High Impedance—The same as A[0–31].

2.4.2 Transfer Type (TT[0–4])—Input

Following are state and timing descriptions for TT[0–4] as input signals.

State Meaning Asserted/Negated—Table 2-1 defines the transactions identified by TT[0–4]. For a full description of coherency actions, see Appendix E, "Coherency Action Tables."

For direct-store operations, TT[0–3] form part of the XATC and are snooped if $\overline{\text{XATS}}$ is asserted.

Timing Comments Assertion/Negation—The same as A[0–31].

Table 2-1. Transfer Encoding for PowerPC 601, 603, 604 Processors

TT [0–4]	Bus Master Transactions			Processor Support	
	Transaction	**Transfer**	**Source**	**Initiator**	**Snooper**
00000	Clean block	Address only	**dcbst**	601, 604	601, 604
00100	Flush block	Address only	**dcbf**	601, 604	601, 604
01000	SYNC	Address only	**sync**	601, 604	601, 604
01100	Kill block	Address only	Store hit on shared block or **dcbz**, **dcbi**, or a 601 **icbi**	601/603/604	601/603/604
10000	Ordered I/O operation	Address only	**eieio**	604	—
10100	External control word write	Single-beat write	**ecowx**	601/603/604	—
11000	TLB invalidate	Address only	**tlbie**	601/604	601/604
11100	External control word read	Single-beat read	**eciwx**	601/603/604	—
00001	**lwarx** reservation set	Address only	**lwarx** cache hit at execution	604	—
00101	Reserved	—	—	—	—
01001	TLB synchronize	Address only	**tlbsync**	604	604
01101	Invalidate instruction cache copy	Address only	**icbi**	604	604
00010	Write-with-flush	Single-beat write or burst	Caching-inhibited or write-through store	601/603/604	601/603/604
00110	Write-with-kill	Burst	Snoop writeback, **dcbf**, **dcbst**, or castout hit modified data	601/603/604	601/603/604
01010	Read	Single-beat read or burst	Cacheable load miss (601/604), cacheable instruction miss or cache-inhibited load	601/603/604	601/603/604
01110	Read-with-intent-to-modify	Burst	Load miss (603) or store miss	601/603/604	601/603/604
10010	Write-with-flush-atomic	Single-beat write	**stwcx.**	601/603/604	601/603/604
10110	Reserved	—	—	—	—
11010	Read-atomic	Single-beat read or burst	**lwarx**	601/603/604	601/603/604

Table 2-1. Transfer Encoding for PowerPC 601, 603, 604 Processors (Continued)

TT [0–4]	Bus Master Transactions			Processor Support	
	Transaction	**Transfer**	**Source**	**Initiator**	**Snooper**
11110	Read-with-intent-to-modify-atomic	Burst	**stwcx.** miss with valid reservation	601/603/604	601/603/604
00X11	Reserved	—	—	—	—
01011	Read-with-no-intent-to-cache	Single-beat read or burst	Snooped only	—	603/604
01111	Reserved	—	—	—	—
1XXX1	Reserved for customer	—	—	—	—

2.4.3 Transfer Burst ($\overline{\text{TBST}}$)—Output

Following are state and timing descriptions for transfer burst ($\overline{\text{TBST}}$) as an output signal.

State Meaning Asserted—A burst transfer is in progress.

Negated—A burst transfer is not in progress.

Also, part of extended address transfer code (XATC); see Section 2.4.1, “Transfer Type (TT[0–4])—Output.”

For external control instructions (**eciwx**/**ecowx**), $\overline{\text{TBST}}$ outputs EAR[28], which is part of the resource ID ($\overline{\text{TBST}}$||TSIZ[0–2]).

Timing Comments Assertion/Negation/High Impedance—The same as A[0–31].

2.4.4 Transfer Burst ($\overline{\text{TBST}}$)—Input

Following are state and timing descriptions for $\overline{\text{TBST}}$ as an input signal.

State Meaning Asserted—For direct-store operations, $\overline{\text{TBST}}$ forms part of the XATC; see Section 2.4.2, “Transfer Type (TT[0–4])—Input.”

Negated—A burst transfer is not in progress.

Timing Comments Assertion/Negation/High Impedance—The same as A[0–31].

2.4.5 Transfer Size (TSIZ[0–2])—Output

Following are state and timing descriptions for the transfer size signals TSIZ[0–2] as output signals.

State Meaning Asserted/Negated—For memory accesses, these signals with $\overline{\text{TBST}}$ indicate the data transfer size for the current bus operation, as shown in Table 2-2. This table shows transfer sizes indicated by combinations of $\overline{\text{TBST}}$ and TSIZ[0–2]. Note that one combination is defined for system use. This combination could be generated by systems but would not be output from a PowerPC processor.

For direct-store operations, these signals form part of the extended address transfer code (XATC); see the description in Section 2.4.1, “Transfer Type (TT[0–4])—Output.”

The external control instructions, **eciwx**/**ecowx**, use these signals to output EAR[29–31], to form the resource ID ($\overline{\text{TBST}}$||TSIZ[0–2]).

Timing Comments Assertion/Negation/High Impedance—The same as A[0–31].

Table 2-2. Data Transfer Size

$\overline{\text{TBST}}$	TSIZ[0–2]	Transfer Size
Asserted	0 0 0	Reserved
Asserted	0 0 1	Burst (16 bytes) reserved for system use
Asserted	0 1 0	Burst (32 bytes)
Asserted	0 1 1	Reserved (64-byte bursts)
Asserted	1 x x	Reserved
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
Negated	1 1 0	6 bytes
Negated	1 1 1	7 bytes

2.4.6 Transfer Size (TSIZ[0–2])—Input

Following are state and timing descriptions for TSIZ[0–2] as input signals.

State Meaning Asserted/Negated—For direct-store operations, TSIZ[0–2] are part of the XATC; see Section 2.4.2, “Transfer Type (TT[0–4])—Input.”

Timing Comments Assertion/Negation—The same as A[0–31].

2.4.7 Transfer Code (TC*n*)—Output

The transfer code (TC*n*) consists of three output signals on the 604 (TC[0–2]) and two output signals for the 601 and 603 (TC[0–1]). These signals provide information about the current transaction that may be useful for implementing external caches. Following are state and timing descriptions for TC*n*.

State Meaning Asserted/Negated—Represents a special encoding for the transfer in progress and gives supplemental information for certain transaction types. See Table 2-3, Table 2-4, and Table 2-5.

Timing Comments Assertion/Negation/High Impedance—The same as A[0–31].

Table 2-3 shows the transfer code definitions for the 601.

Table 2-3. Transfer Code Signal Encoding for PowerPC 601 Processor

Signal	State	Definition
TC0	Asserted	Read: Bus operation is an instruction fetch.
		Write: Operation is invalidating the cache line in the 601.
		Kill block (address only): Operation is invalidating the cache block in the 601.
	Deasserted	Read: Bus operation is not an instruction fetch.
		Write: Operation is not invalidating the cache line in the 601.
		Kill block (address only): Operation is not invalidating the cache block in the 601.
TC1	Asserted	The next access is likely to be on the same page; a sector has been loaded and a low-priority load of the adjacent sector is queued.
	Deasserted	The next access isn't likely to be on the next page; no load to the adjacent sector is queued.

Table 2-4 shows the transfer code meanings for the 603.

Table 2-4. Transfer Code Signal Encoding for the PowerPC 603 Processor

TC[0–1]	Read	Write
0 0	Data transaction	Any write
0 1	Touch load	—
1 0	Instruction fetch	—
1 1	Reserved	—

Table 2-5 shows transfer code options for the 604 and gives the transaction type, the encoding of the $\overline{WT}$ and TC[0–2] signals, and the type of cycle.

Table 2-5. Transfer Code Signal Encoding for PowerPC 604 Processor

Transfer Type	$\overline{WT}$[1]	TC [0–2]	$\overline{BR}$ Asserted [2, 3]	From Write-Back Buffer	$\overline{TS}$ after $\overline{ARTRY}$d Snoop[4]	Final Cache State[5]	Comments
Write with kill	1	100	Never	Always	Don't care	I	Cache copy-back
	0	xx0	No	Yes	Yes	M, E, S, I	To distinguish between cache copy-back, block clean (**dcbst**), or block flush (**dcbf**), this transaction must be $\overline{ARTRY}$d. This transaction eventually returns (before anything but another snoop push directly from the data cache) indicating another $\overline{WT}$/TC code combination.
		100	No	Yes	No	I	Block flush (**dcbf**)
		000	No	Yes	No	M, E, I	Block clean (**dcbst**)—The **dcbst** instruction changes the cache state to E when the modified block is put in the copy-back buffer. Before the low-priority write-back buffer entry completes its address tenure, the cache state can be changed to M by a store or to I by a **dcbi** or a cache miss.
		010	Yes	No	Don't care	S, I	Snoop push[6] directly from data cache (read or read-atomic)—The read or read-atomic snoop changes the data cache state to S when the modified block is placed in the snoop-push buffer. Before the buffer completes its address tenure, the cache can be changed to I by a **dcbi** or cache miss.

Table 2-5. Transfer Code Signal Encoding for PowerPC 604 Processor (Continued)

Transfer Type	$\overline{WT}$[1]	TC [0–2]	$\overline{BR}$ Asserted [2, 3]	From Write-Back Buffer	$\overline{TS}$ after $\overline{ARTRY}$d Snoop[4]	Final Cache State[5]	Comments
Write with kill	0	010	Yes	Yes	Don’t care	S or I	Snoop push[6] from write-back buffer (read or read-atomic)—The data cache has a shared copy if the buffer held a block clean (**dcbst**) transaction. If it held a block flush (**dcbf**) or cache write-back transaction, the cache has no valid copy after the transaction. To know if the processor kept a shared copy or invalidated this block, this transaction must be $\overline{ARTRY}$d. If it originated from the write-back buffers and no new snoops occur, the transaction returns as the next $\overline{TS}$ and indicates a DCBF, DCBST, or write-back $\overline{WT}$/TC code. If it returns as a snoop push read, it came from the data cache.
		100	Yes	No	Don’t care	I	Snoop push[6] directly from data cache (RWITM, RWITM-atomic, flush, write w/flush, write w/flush-atomic, or kill)
		100	Yes	Yes	Don’t care	I	Snoop push[6] from write-back buffers (RWITM, RWITM-atomic, flush, write w/flush-atomic, write w/flush, write w/kill, or kill)
		000	Yes	No	Don’t care	M, E, I	Snoop push[6] from data cache (clean or RWNITC)—The clean or RWNITC snoop changes the data cache state to E when the modified block is put in the snoop-push buffer. Before the buffer completes its address tenure, the cache state can be changed to M by a store or to I by either a **dcbi** instruction or cache miss.
		000	Yes	Yes	Don’t care	M, E, I (**dcbst** in buffer) I (cache write-back or **dcbf** in buffer)	Snoop push[6] from write-back buffers (clean or RWNITC)—If this snoop hit on a block-flush (**dcbf**) or a cache write-back in the write-back buffers, the cache does not have a valid copy of this address after this transaction. If this snoop hits a block-store (**dcbst**) in the write-back buffers, the processor can keep an exclusive copy of the cache block.
Kill block	x	100	Never	No	Don’t care	I	Kill block deallocate (**dcbi**)
	1	000				M	Kill block and allocate no castout required (**dcbz**)
	1	001					Kill block and allocate castout required (**dcbz**)
	1	000					Kill block; write to block marked S

Table 2-5. Transfer Code Signal Encoding for PowerPC 604 Processor (Continued)

Transfer Type	$\overline{\mathrm{WT}}$[1]	TC [0–2]	$\overline{\mathrm{BR}}$ Asserted [2, 3]	From Write-Back Buffer	$\overline{\mathrm{TS}}$ after $\overline{\mathrm{ARTRY}}$d Snoop[4]	Final Cache State[5]	Comments
Read[7]	W[8]	0x0	Never	No	Don’t care	E, S	Data read, no castout required—The cache state is S if $\overline{\mathrm{SHD}}$ was asserted to the processor for a read or read-atomic transaction. If $\overline{\mathrm{SHD}}$ was not asserted or if the transaction was an RWITM or RWITM-atomic transaction, the cache state is E.
	W	0x1				E, S	Data read, castout required—The cache state is S if $\overline{\mathrm{SHD}}$ was asserted to the processor for a read or read-atomic transaction. If $\overline{\mathrm{SHD}}$ was not asserted, or if the transaction was an RWITM or RWITM-atomic transaction, the cache state is E.
	W	1x0				Valid	Instruction read
ICBI	x	100	Never	No	Don’t care	Invalid	Kill block deallocate (**icbi**[9])

Notes:

[1] The value in the $\overline{\mathrm{WT}}$ column reflects the logic value seen on the signal.

[2] The window for the assertion of $\overline{\mathrm{BR}}$ is defined as the second cycle after $\overline{\mathrm{AACK}}$ if $\overline{\mathrm{ARTRY}}$ were asserted the cycle after $\overline{\mathrm{AACK}}$.

[3] The full condition for this column is “The $\overline{\mathrm{BR}}$ corresponding to this transaction was asserted in the window for the last snoop to this address.”

[4] The full condition for this column is “This transaction is the first $\overline{\mathrm{TS}}$ asserted by this processor after one or more $\overline{\mathrm{ARTRY}}$d snoop transactions and the address of this transaction matches the address of at least one of those $\overline{\mathrm{ARTRY}}$d snoop transactions.”

[5] This column reflects the final MESI state in the processor of the line referenced by this transaction after the transaction completes successfully without $\overline{\mathrm{ARTRY}}$.

[6] This snoop push is guaranteed to push the most-recently modified data in the processor. No more snoop operations are required to ensure that this snoop has been fully processed by the processor.

[7] Read in this case encompasses all of read or RWITM, normal or atomic.

[8] W = write-through bit from translation

[9] **icbi** is distinguished from kill block by assertion of TT4.

2.4.8 Cache Inhibit ($\overline{\mathrm{CI}}$)—Output

Following are state and timing descriptions for the cache inhibit ($\overline{\mathrm{CI}}$) output signal.

State Meaning Asserted—Generally 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—Generally indicates that a burst transfer will allocate a data cache block. Set negated for castouts and pushes.

Section 4.8, “External WIM Bit Settings,” describes exceptions to the above.

Timing Comments Assertion/Negation/High Impedance—The same as A[0–31].

2.4.9 Write-Through ($\overline{\text{WT}}$)—Output

Following are state and timing descriptions for the write-through ($\overline{\text{WT}}$) output signal.

State Meaning Asserted—Generally indicates that a single-beat transaction is write-through, reflecting the value of the W bit for the block or page that contains the address of the current transaction.

Negated—Generally indicates a transaction is not write-through.

Section 4.8, "External WIM Bit Settings," describes exceptions to the above.

Timing Comments Assertion/Negation/High Impedance—The same as A[0–31].

2.4.10 Global ($\overline{\text{GBL}}$)—Output

Following are state and timing descriptions for global signal ($\overline{\text{GBL}}$) as an output signal. For the 604e, HID0[23] lets software control the behavior of $\overline{\text{GBL}}$ for instruction fetches through the address-translation mechanism; refer to 604e user documentation.

State Meaning Asserted—Generally indicates that a transaction is global, reflecting the setting of the M bit for the block or page that contains the address of the current transaction (except in the case of write-back operations, which are nonglobal.)

Negated—Generally indicates that a transaction is not global. For 603 and 604, this signal is negated on instruction fetches.

For exceptions, see Section 4.8, "External WIM Bit Settings."

Timing Comments Assertion/Negation/High Impedance—The same as A[0–31].

2.4.11 Global ($\overline{\text{GBL}}$)—Input

Following are state and timing descriptions for $\overline{\text{GBL}}$ as an input signal.

State Meaning Asserted—A transaction can be snooped; however, the processor will not snoop reserved transaction types, bus operations associated with the **eieio**, **eciwx**, or **ecowx** instructions, or address-only bus transactions associated with an **lwarx** reservation set. Note that the 603 snoops the reservation address register for global and nonglobal address transfers. This snooping is required for the 603's implementation of the **lwarx** and **stwcx.** instructions, which require snoops on castouts and snoop pushes (nonglobal). Snoops with $\overline{\text{GBL}}$ = 1 do not affect the cache state.

Negated—A transaction is not snooped by the processor.

Timing Comments Assertion/Negation—The same as A[0–31].

2.4.12 Cache Set Element (CSE*n*)—Output

The number of cache set element signals on each processor depends upon the cache associativity of that processor. There are three cache set element signals on the 601 (CSE[0–2]), one on the 603 (CSE), and two on the 603e, 604, and 604e (CSE[0–1]). Following are state and timing descriptions for the CSE*n* signals. In some documentation these signals are called cache set entry or cache set enable signals.

State Meaning Asserted/Negated—Represents the cache replacement set element (also referred to as the way or coherency class) for the current cache transaction. Can be used with the address bus and the transfer attribute signals to externally track the state of each cache block in the processor. The CSE*n* signals are not meaningful during data cache touch load operations on a 603.

Timing Comments Assertion/Negation/High Impedance—The same as A[0–31].

2.4.13 High-Priority Snoop Request (HP_SNP_REQ)–601 Only

Following are state and timing descriptions for the high-priority snoop request input signal (HP_SNP_REQ) on the 601. This signal is enabled by setting HID0[31].

State Meaning Asserted—The 601 may add an additional reserved queue position to the list of available queue positions for push transactions that are a result of a snoop hit.

Negated—The 601 will not make the reserved queue available for a snoop hit push resulting from a transaction. This is the normal mode.

Timing Comments Assertion/Negation—Must be valid throughout the address tenure.

2.5 Address Transfer Termination Signals

The address transfer termination signals indicate either that the address tenure has completed successfully or must be repeated, and when it should be terminated. Section 3.2.3, “Address Transfer Termination,”describes how these signals interact.

2.5.1 Address Acknowledge (AACK)—Input

Following are state and timing descriptions for the address acknowledge (AACK) as an input signal.

State Meaning Asserted—The address phase of a transaction is complete. The address bus goes to high-impedance state on the next bus clock cycle. The processor samples ARTRY on the bus clock cycle after assertion of AACK.

The 604 can sample ARTRY by the second cycle after TS is asserted.

Negated—During assertion of ABB, indicates the address bus and transfer attribute signals must remain driven.

Timing Comments Assertion—Can occur as soon as the bus clock cycle after $\overline{\text{TS}}$ or $\overline{\text{XATS}}$ is asserted, but can be delayed to extend address access time, for example, to support slow snooping devices.

Negation—Must occur one bus clock cycle after assertion of $\overline{\text{AACK}}$.

2.5.2 Address Retry ($\overline{\text{ARTRY}}$)—Output

Following are state and timing descriptions for address retry ($\overline{\text{ARTRY}}$) as an output signal.

State Meaning Asserted—The master detects a condition in which a snooped address tenure must be retried. If the processor must update memory as a result of the snoop that caused the retry, the processor asserts $\overline{\text{BR}}$ during that snoop window, which is defined as the second cycle after $\overline{\text{AACK}}$ if $\overline{\text{ARTRY}}$ was asserted the cycle after $\overline{\text{AACK}}$.

Also invalidates data in some cases; see Section 3.3.1.1, “Effect of ARTRY Assertion on Data Transfer and Arbitration on the PowerPC 604 Processor.”

High Impedance—The master does not need the snooped address tenure to be retried.

Timing Comments Assertion—Asserted the second bus cycle after the assertion of $\overline{\text{TS}}$ if a retry is required. Thus, when a retry is required, there is only one empty cycle between the assertions of $\overline{\text{TS}}$ and $\overline{\text{ARTRY}}$.

Negation—Occurs the second bus cycle after the assertion of $\overline{\text{AACK}}$. Because $\overline{\text{ARTRY}}$ can be simultaneously driven by multiple devices, it is driven negated in the following ways:

601—Occurs the second bus cycle after the assertion of $\overline{\text{AACK}}$. Since $\overline{\text{ARTRY}}$ may be simultaneously driven by multiple devices, it negates in a unique fashion. First the buffer goes to high impedance for one bus cycle, then it is driven high for one 2XPCLK cycle before returning to high impedance. This method of negation may be disabled by setting HID0[29].

603—Occurs the second bus cycle after the assertion of $\overline{\text{AACK}}$. Since $\overline{\text{ARTRY}}$ may 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), then it is driven negated for one bus cycle before returning to high impedance. This method of negation can be disabled by setting HID0[7].

604—$\overline{\text{ARTRY}}$ becomes high impedance for at least one-half bus cycle, then is driven high for approximately one bus cycle. $\overline{\text{ARTRY}}$ is then guaranteed by design to become high impedance at the latest by the start of third cycle after $\overline{\text{AACK}}$. This method of negation can be disabled by setting HID0[7].

2.5.3 Address Retry ($\overline{\text{ARTRY}}$)—Input

Following are state and timing descriptions for $\overline{\text{ARTRY}}$ as an input signal.

State Meaning Asserted—For the address bus master, $\overline{\text{ARTRY}}$ indicates the device must retry the preceding address tenure and immediately negate $\overline{\text{BR}}$ (if asserted). If the associated data tenure has begun, the 603 and 604 also abort the data tenure immediately even if burst data has been received. For devices that are not the address bus master, this input indicates they should immediately negate $\overline{\text{BR}}$ for one bus clock cycle after the assertion of $\overline{\text{ARTRY}}$ by the snooping bus master to allow a write-back operation.

Negated/High Impedance—The master need not retry the last address tenure.

Timing Comments Assertion—May occur as soon as the second cycle after $\overline{\text{TS}}$ or $\overline{\text{XATS}}$ is asserted; must occur by the bus clock cycle immediately after the assertion of $\overline{\text{AACK}}$ if an address retry is required.

Negation—Must occur in the second cycle after $\overline{\text{AACK}}$ is asserted.

2.5.4 Shared ($\overline{\text{SHD}}$)—Output

Following are state and timing descriptions for the shared ($\overline{\text{SHD}}$) as an output signal.

State Meaning Asserted—If $\overline{\text{ARTRY}}$ is negated, indicates that after this transaction completes successfully, the master will keep a valid shared copy of the address or that a reservation exists on this address. If $\overline{\text{SHD}}$ and $\overline{\text{ARTRY}}$ are asserted for a snooping master, the snoop hit modified data that will be pushed as the master's next address transaction.

Negated/High Impedance—After this address is transferred, the processor will not have a valid copy of the snooped address.

Timing Comments Assertion/Negation—Same as $\overline{\text{ARTRY}}$.

High Impedance—Same as $\overline{\text{ARTRY}}$.

Because it does not support the shared MESI state (S), the 603 does not implement $\overline{\text{SHD}}$.

2.5.5 Shared ($\overline{\text{SHD}}$)—Input

Following are state and timing descriptions for $\overline{\text{SHD}}$ as an input signal.

State Meaning Asserted—If $\overline{\text{ARTRY}}$ is not asserted, the master must allocate the incoming cache block as shared (S) for a self-generated transaction. Applies only to read and read atomic transactions.

Negated—If $\overline{\text{ARTRY}}$ is negated, the master can allocate the incoming cache block as exclusive (E) for a self-generated read or read-atomic transaction.

Timing Comments Assertion/Negation—The same as $\overline{\text{ARTRY}}$.

Because it does not support the shared (S) MESI state, the 603 does not implement $\overline{\text{SHD}}$.

2.6 Data Bus Arbitration Signals

Like address bus arbitration signals, data bus arbitration signals maintain an orderly process for determining data bus mastership. Note that there is no equivalent to the address bus arbitration signal BR (bus request), because, except for address-only transactions, TS and XATS imply data bus requests. For a detailed description on how these signals interact, see Section 3.3.1, "Data Bus Arbitration."

The DBWO signal lets the processor be configured dynamically to write data out of order with respect to read data.

2.6.1 Data Bus Grant (DBG)—Input

Following are state and timing descriptions for the data bus grant (DBG) as an input signal.

State Meaning Asserted—With proper qualification a device can become data bus master. Note that in some cases, assertion of ARTRY invalidates the data bus grant (see Section 3.3.1.1, "Effect of ARTRY Assertion on Data Transfer and Arbitration on the PowerPC 604 Processor"). The device achieves a qualified data bus grant when the following conditions are met:

- The data bus is not bus busy (DBB is negated). (This condition does not apply to the 604 (or 604e) in data streaming mode.)
- DRTRY is negated. (This condition does not apply for a processor using data streaming or no-DRTRY mode.)
- ARTRY is negated if ARTRY applies to the associated address tenure.

Negated—The master must hold off its data tenures.

Timing Comments Assertion—May occur any time to indicate that the device is free to assume data bus mastership. The processor can sample it as early as the cycle that TS or XATS is asserted.

For the 604 in data streaming mode, DBG must be asserted for exactly one cycle per data bus tenure, the cycle before the data tenure is to begin. The system cannot assert DBG earlier or park DBG, or assert it for consecutive cycles. The DBB signal does not participate in determining a qualified data bus grant. Therefore, the system must assert DBG in a way that prevents data tenure collisions from different masters. Also, the system must assert DBG so data tenures complete before providing another DBG. If a DBG is given early to the 604 in data streaming mode, the processor drops the current data tenure prematurely in the next cycle and begins any pending data tenure.

The 604e has less restrictive timing requirements in data streaming mode—$\overline{\text{DBG}}$ must be asserted no earlier than the cycle before 604e's data tenure is to begin only when another master currently owns the data bus (that is, when $\overline{\text{DBB}}$ would normally be asserted for a data tenure). If no other masters own the data bus (asserting $\overline{\text{DBB}}$), the 604e allows the system to park $\overline{\text{DBG}}$. $\overline{\text{DBB}}$ is still an output-only signal in data streaming mode (that is, $\overline{\text{DBB}}$ does not participate in a qualified data bus grant), requiring the system to use $\overline{\text{DBG}}$ to ensure that different masters don't collide on data tenures. If the system tries to stream back-to-back data tenures by asserting $\overline{\text{DBG}}$ with the final $\overline{\text{TA}}$ of the first data tenure, the processor accepts the $\overline{\text{DBG}}$ as a qualified data bus grant only if the current and next data tenures are both burst reads. Other combinations cannot be streamed.

Negation—May occur at any time to indicate that the master cannot assume control of the data bus.

2.6.2 Data Bus Write Only ($\overline{\text{DBWO}}$)—Input

Following are state and timing descriptions for $\overline{\text{DBWO}}$ as an input signal.

State Meaning Asserted—The processor can run the data bus tenure for an outstanding write address even if a read address is pipelined before the write address. If write data is not available, the processor performs the first pending read transfer. See Section 3.3.2, “Data Bus Write Only,” for detailed instructions for using $\overline{\text{DBWO}}$. Note that the 601 takes the bus only for a pending data bus write operation and not for a read operation.

Negated—The processor runs address and data tenures in the same order. Tying $\overline{\text{DBWO}}$ negated preserved address/data ordering.

Timing Comments Assertion—Must occur no later than a qualified $\overline{\text{DBG}}$ for a pending write tenure. The $\overline{\text{DBWO}}$ signal is recognized by the processor only on the clock cycles of a qualified data bus grant.

Negation—May occur any time after a qualified data bus grant and before the next qualified data bus grant.

2.6.3 Data Bus Busy ($\overline{\text{DBB}}$)—Output

Following are state and timing descriptions for data bus busy ($\overline{\text{DBB}}$) as an output signal.

State Meaning Asserted—The device is the data bus master. The processor always assumes data bus mastership if it needs the data bus and is given a qualified data bus grant (see $\overline{\text{DBG}}$).

Negated—The device is not using the data bus, unless the data tenure is being extended by the assertion of $\overline{\text{DRTRY}}$. Note that for the 604e in no-$\overline{\text{DRTRY}}$ mode, $\overline{\text{DRTRY}}$ is tied asserted and is ignored.

Timing Comments Assertion—Occurs in the bus clock cycle after a qualified $\overline{\text{DBG}}$.

Negation—Occurs for a fractional bus clock cycle after the assertion of the final $\overline{\text{TA}}$ or within two cycles of the assertion of $\overline{\text{TEA}}$.

High Impedance—Occurs during a fractional portion of the bus cycle in which $\overline{\text{DBB}}$ is negated. The $\overline{\text{DBB}}$ signal is designed to be high impedance by the end of the cycle in which it is negated. For specific information, see the appropriate user's manual.

2.6.4 Data Bus Busy ($\overline{\text{DBB}}$)—Input

Following are state and timing descriptions for $\overline{\text{DBB}}$ as an input signal. In data streaming mode, $\overline{\text{DBB}}$ is only an output and is not part of a qualified data bus grant; see Chapter 6, "Additional Bus Configurations."

State Meaning Asserted—Another device is data bus master. Note that $\overline{\text{DBB}}$ cannot be used in systems that use read data streaming.

Negated—The device is not using the data bus. If the arbiter is designed to assert $\overline{\text{DBG}}$ exactly one cycle before the next data tenure starts, $\overline{\text{DBB}}$ is unnecessary and may be pulled high.

Timing Comments Assertion—Must occur when the processor must be kept from using the data bus.

Negation—May occur whenever the data bus is available.

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 data transfer signals interact, see Section 3.3.3, "Data Transfer."

2.7.1 Data Bus (DH[0–31], DL[0–31])—Output

Following are state and timing descriptions for the DH and DL as output signals.

State Meaning Asserted/Negated—Represents the state of data during a data write. The data bus has two halves—data bus high (DH) and data bus low (DL). Table 2-6 shows data bus lane assignments. Direct-store operations use DH exclusively (there are no 64-bit, direct-store operations). Unselected byte lanes do not supply valid data.

Timing Comments Assertion/Negation—Initial beat coincides with $\overline{\text{DBB}}$ and, for bursts, transitions on the bus clock cycle after each assertion of $\overline{\text{TA}}$.The data bus is driven once for noncached transactions and four times for processor cache transactions (bursts).

High Impedance—Occurs on the bus clock cycle after the final assertion of $\overline{\text{TA}}$.

Table 2-6. 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

2.7.2 Data Bus (DH[0–31], DL[0–31])—Input

Following are state and timing descriptions for DH and DL as input signals.

State Meaning Asserted/Negated—Represents the state of data during a data read transaction. The data bus has two halves, data bus high (DH) and data bus low (DL). Table 2-6 shows byte lanes. Direct-store operations use DH exclusively (there are no 64-bit direct-store operations).

Timing Comments Assertion/Negation—Data must be valid on the same bus clock cycle that $\overline{\text{TA}}$ is asserted.The data bus is driven once for noncached transactions and four times for processor cache transactions (bursts).

2.7.3 Data Bus Parity (DP[0–7])—Output

Following are state and timing descriptions for the data bus parity (DP[0–7]) output signals.

State Meaning Asserted/Negated—Represents odd parity for each of the eight bytes of a data write transaction. Odd parity means that an odd number of bits, including the parity bit, are driven high. Table 2-7 shows signal assignments. All eight bits are driven with valid parity on all bus write operations except direct-store operations for which only DP[0–3] are driven with valid parity.

Timing Comments Assertion/Negation—The same as DL[0–31].
High Impedance—The same as DL[0–31].

Table 2-7. 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]

Table 2-7. DP[0–7] Signal Assignments (Continued)

Signal Name	Signal Assignments
DP4	DL[0–7]
DP5	DL[8–15]
DP6	DL[16–23]
DP7	DL[24–31]

2.7.4 Data Bus Parity (DP[0–7])—Input

Following are state and timing descriptions for DP[0–7] as input signals.

State Meaning Asserted/Negated—Represents one bit of odd parity for each byte of read data. Parity is checked on all data byte lanes during data read operations, regardless of the size of the transfer. During direct-store read operations, only the DP[0–3] signals (corresponding to byte lanes DH[0–31]) are checked for odd parity. If data parity errors are enabled, detected even parity causes a checkstop or a machine check exception (and assertion of $\overline{\text{DPE}}$) depending on the state of MSR[ME]. For the 601, if data parity check is enabled in HID0, detection of even parity unconditionally causes a checkstop.

Timing Comments Assertion/Negation—The same as DL0–DL31.

2.7.5 Data Parity Error ($\overline{\text{DPE}}$)—Output

Following are state and timing descriptions for the data parity error ($\overline{\text{DPE}}$) output signal. $\overline{\text{DPE}}$ is an open-drain type output and requires a pull-up resistor for proper deassertion.

State Meaning Asserted—The processor detected incorrect data bus parity on incoming read data.

Negated—Indicates correct data bus parity.

Timing Comments Assertion—Occurs on the second bus clock cycle after $\overline{\text{TA}}$ is asserted to the processor and is driven for one cycle.

2.7.6 Data Bus Disable ($\overline{\text{DBDIS}}$)—Input

Following are the state meanings and timing comments for the data bus disable ($\overline{\text{DBDIS}}$) input signal. This signal is not on the 601.

State Meaning Asserted—For a write transaction, the processor must release the data bus and DP[0–7] to high impedance in the next cycle. The data tenure remains active, $\overline{\text{DBB}}$ remains driven, and the transfer termination signals are still monitored by the processor. The $\overline{\text{DBDIS}}$ signal is ignored for read transactions.

Negated—The data bus should remain normally driven.

Timing Comments Assertion/Negation—Should be driven one cycle before the data bus can be driven by the processor. May be asserted on any clock cycle when the processor is driving, or will be driving, the data bus and may remain asserted multiple cycles.

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 3.3.4, "Data Transfer Termination."

2.8.1 Transfer Acknowledge ($\overline{\text{TA}}$)—Input

Following are state and timing descriptions for the transfer acknowledge ($\overline{\text{TA}}$) input signal.

State Meaning Asserted—A single-beat data transfer or a data beat in a burst transfer completed successfully (unless $\overline{\text{DRTRY}}$ is asserted on the next bus clock cycle for reads). The $\overline{\text{TA}}$ signal must be asserted for each data beat in a burst transaction.

Negated—Until $\overline{\text{TA}}$ is asserted, the master must continue driving the data for the current write or must wait to sample the data for reads.

Timing Comments Assertion—During a data tenure, which generally begins after a qualified data bus grant and continues through the period defined by $\overline{\text{DBB}}$ or $\overline{\text{DRTRY}}$. This period is affected by the $\overline{\text{ARTRY}}$ window. See Section 3.3.1.1, "Effect of ARTRY Assertion on Data Transfer and Arbitration on the PowerPC 604 Processor." The system can withhold asserting $\overline{\text{TA}}$ to indicate that the master should insert wait states to extend a data tenure.

Negation—Must occur after the bus clock cycle of the final (or only) data beat of the transfer. For a burst transfer, the system can assert $\overline{\text{TA}}$ for one bus clock cycle and then negate it to advance the burst transfer to the next beat and insert wait states during the next beat.

When the 603 is configured for 1:1 clock mode and is performing a burst read into data cache, the 603 requires one wait state between the assertion of $\overline{\text{TS}}$ and the first assertion of $\overline{\text{TA}}$ for that transaction. If no-$\overline{\text{DRTRY}}$ mode is also selected, the 603 requires two wait states.

2.8.2 Data Retry ($\overline{\text{DRTRY}}$)—Input

Following are state and timing descriptions for the data retry ($\overline{\text{DRTRY}}$) input signal.

State Meaning Asserted—The master must invalidate the data from the previous read operation. $\overline{\text{DRTRY}}$ is ignored for write transactions and is not defined for direct-store transfers.

Negated—Data presented with $\overline{\text{TA}}$ on the previous read operation is valid. This is essentially a late $\overline{\text{TA}}$ to allow speculative forwarding of data (with $\overline{\text{TA}}$) during reads.

Timing Comments

Assertion—Must occur during the bus clock cycle immediately after $\overline{\text{TA}}$ is asserted if a retry is required. The $\overline{\text{DRTRY}}$ signal can be held asserted for multiple bus clock cycles. When it is negated, data must have been valid on the previous clock with $\overline{\text{TA}}$ asserted.

Negation—Must occur during the bus clock cycle after a valid data beat. This may occur several cycles after $\overline{\text{DBB}}$ is negated, effectively extending the data bus tenure.

Start-up—For 603 and 604e, $\overline{\text{DRTRY}}$ is sampled at the negation of $\overline{\text{HRESET}}$; if $\overline{\text{DRTRY}}$ is asserted, no-$\overline{\text{DRTRY}}$ mode is selected. If $\overline{\text{DRTRY}}$ is negated at start-up, $\overline{\text{DRTRY}}$ is enabled. If no-$\overline{\text{DRTRY}}$ or data streaming mode is selected, $\overline{\text{DRTRY}}$ must be negated during normal operation (after $\overline{\text{HRESET}}$). The no-$\overline{\text{DRTRY}}$ mode provides a one-cycle faster read and the data streaming eliminates wasted cycles between data bursts. See Section 6.1, “No-DRTRY Mode (603 and 604e),” for a description of no-$\overline{\text{DRTRY}}$ mode or Chapter 6, “Additional Bus Configurations,” for a description of data streaming.

2.8.3 Transfer Error Acknowledge ($\overline{\text{TEA}}$)—Input

Following are state and timing descriptions for the transfer error acknowledge ($\overline{\text{TEA}}$) input signal.

State Meaning

Asserted—A bus error occurred that causes a machine check exception (or causes the processor to enter the checkstop state if the machine check enable bit is cleared (MSR[ME] = 0)). For more information, see Section 5.3, “Machine Check and Checkstops.” Assertion terminates the current transaction; that is, assertion of $\overline{\text{TA}}$ and $\overline{\text{DRTRY}}$ are ignored. Asserting $\overline{\text{TEA}}$ causes the negation/high impedance of $\overline{\text{DBB}}$ in the next clock cycle. However, data entering the GPR or the cache are not invalidated.

If $\overline{\text{TEA}}$ is asserted during a direct-store transaction, the machine check or checkstop action of the $\overline{\text{TEA}}$ is delayed and subsequent direct-store transactions continue until all transfers from the direct-store segment complete. The $\overline{\text{TEA}}$ signal must be asserted for every direct-store data tenure including the last one. The processor takes a machine check or a checkstop no sooner than the last direct-store data tenure has been terminated by the assertion of $\overline{\text{TEA}}$. A load or store reply is not necessary after the last data tenure receives a $\overline{\text{TEA}}$ assertion.

Negated—No bus error was detected.

Timing Comments Assertion—May be asserted while $\overline{\text{DBB}}$ is asserted or during the valid $\overline{\text{DRTRY}}$ window. In data streaming mode, the 604/604e does not recognize $\overline{\text{TEA}}$ the cycle after $\overline{\text{TA}}$ during a read operation due to the absence of a $\overline{\text{DRTRY}}$ assertion opportunity. $\overline{\text{TEA}}$ should be asserted for one cycle only.

Negation—$\overline{\text{TEA}}$ must be negated no later than the negation of $\overline{\text{DBB}}$ or the last $\overline{\text{DRTRY}}$. The processor deasserts $\overline{\text{DBB}}$ within one bus clock cycle after the assertion of $\overline{\text{TEA}}$.

2.9 System Status Signals

Most system interrupt, checkstop, and reset signals are input signals that indicate when exceptions are received, when checkstop conditions have occurred, and when the processor must be reset. The processor generates $\overline{\text{CKSTP\_OUT}}$ when it detects a checkstop condition. For detailed descriptions, see Chapter 5, “System Status Signals.”

2.9.1 Interrupt ($\overline{\text{INT}}$)—Input

Following are state and timing descriptions for the interrupt ($\overline{\text{INT}}$) input signal.

State Meaning Asserted—The processor initiates an external interrupt if MSR[EE] is set and $\overline{\text{INT}}$ remains asserted long enough; otherwise, the processor ignores the interrupt.

Negated—Normal operation should proceed. See Section 5.4, “External Interrupt Exception (0x00500).”

Timing Comments 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 exception is taken. For the 601, this signal can be negated after at least three processor clock cycles.

2.9.2 System Management Interrupt ($\overline{\text{SMI}}$)—Input

Following are state and timing descriptions for the system management interrupt ($\overline{\text{SMI}}$) input signal. This interrupt supports power management and is not on the 601.

State Meaning Asserted—The processor initiates a system management interrupt exception if MSR[EE] is set.

Negated—Normal operation should proceed. See Section 5.4, “External Interrupt Exception (0x00500).”

Timing Comments 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 exception is taken.

2.9.3 Machine Check Interrupt ($\overline{\text{MCP}}$)—Input

Following are state and timing descriptions for the machine check interrupt ($\overline{\text{MCP}}$) input signal. This signal is not on the 601.

State Meaning Asserted—The processor 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 processor must terminate operation by internally gating off all clocks and releasing all outputs (except $\overline{\text{CKSTP\_OUT}}$) to the high-impedance state. If HID0[EMCP] is cleared, the processor ignores the interrupt condition. The $\overline{\text{MCP}}$ signal must remain asserted for two bus clock cycles.

Negated—Normal operation should proceed.

Timing Comments Assertion—May occur at any time and may be asserted asynchronously to the input clocks. $\overline{\text{MCP}}$ is negative edge-sensitive.

Negation—May be negated two bus cycles after assertion.

2.9.4 Checkstop Input ($\overline{\text{CKSTP\_IN}}$)—Input

Following are state and timing descriptions for the checkstop input signal ($\overline{\text{CKSTP\_IN}}$).

State Meaning Asserted—The processor must terminate operation by internally gating off all clocks and releasing all outputs except $\overline{\text{CKSTP\_OUT}}$ to high-impedance state. Once asserted, $\overline{\text{CKSTP\_IN}}$ must remain asserted until the system has been reset.

Negated—Normal operation should proceed. See Section 5.3, “Machine Check and Checkstops.”

Timing Comments Assertion—May occur at any time and may be asserted asynchronously to the input clocks.

For the 601, $\overline{\text{CKSTP\_IN}}$ must be asserted at least three PCLK_EN clock cycles. Or it may be asserted synchronously meeting setup and hold times (specified in the hardware specifications) and must be asserted for at least two PCLK_EN clock cycles.

Negation—May occur any time after $\overline{\text{CKSTP\_OUT}}$ is asserted.

2.9.5 Checkstop Output ($\overline{\text{CKSTP\_OUT}}$)—Output

Following are state and timing descriptions for checkstop output ($\overline{\text{CKSTP\_OUT}}$) as an output signal. Note that $\overline{\text{CKSTP\_OUT}}$ is an open-drain type output and requires an external pull-up resistor to assure proper deassertion.

State Meaning Asserted—The processor detected a checkstop condition and ceased operation.

Negated—The processor is operating normally. See Section 5.3, “Machine Check and Checkstops.”

Timing Comments Assertion—Can occur at any time asynchronously to input clocks.

Negation—Is negated upon assertion of $\overline{\text{HRESET}}$.

2.9.6 Hard Reset ($\overline{\text{HRESET}}$)—Input

The hard reset ($\overline{\text{HRESET}}$) input signal must be used at power-on to properly reset the processor. This input has additional functionality in certain test modes. Following are state and timing descriptions for $\overline{\text{HRESET}}$.

State Meaning Asserted—Initiates a hard reset operation when $\overline{\text{HRESET}}$ transitions from asserted to negated. Causes a reset exception as described in Section 5.2.1.1, “Hard Reset Settings.” Output drivers are released to high impedance within five clocks (three clocks for the 601) after the assertion of $\overline{\text{HRESET}}$.

Negated—Normal operation should proceed.

Timing Comments Assertion—Can occur at any time and can be asynchronous with the processor input clock; must be held asserted for at least 255 (300 for the 601) clock cycles.

Negation—Can occur after the minimum reset pulse width is met.

2.9.7 Soft Reset ($\overline{\text{SRESET}}$)—Input

The soft reset ($\overline{\text{SRESET}}$) input signal has additional functionality in certain test modes. Following are state and timing descriptions for $\overline{\text{SRESET}}$.

State Meaning Asserted—Initiates processing for a soft reset exception as described in Section 5.2.2, “Soft Reset.”

Negated—Normal operation should proceed.

Timing Comments Assertion—Can occur at any time and can be asynchronous with the processor input clock. $\overline{\text{SRESET}}$ is negative edge-sensitive.

Negation—May occur any time after the minimum soft reset pulse width of two (10 for the 601) bus cycles is met.

2.10 Processor State Signals

The signals described in this section provide inputs for controlling the time base in the processor, external cache access by the processor, and an output signal from the processor to indicate that a memory reservation has been set.

2.10.1 Reservation ($\overline{\text{RSRV}}$)—Output

Following are state and timing descriptions for the reservation ($\overline{\text{RSRV}}$) output signal.

State Meaning Asserted/Negated—Reflects the state of the reservation coherency bit used by the **lwarx/stwcx.** instructions. See Section 4.5.1, “PowerPC 603 Processor lwarx/stwcx. Implementation.”

Timing Comments Assertion--Occurs synchronously one bus clock cycle after execution of an **lwarx** instruction that sets the internal reservation condition. On 604 and 604e, $\overline{\text{RSRV}}$ is asserted as late as the fourth cycle after $\overline{\text{AACK}}$ for a read-atomic operation if the **lwarx** instruction requires a read-atomic operation.

Negation—Occurs synchronously one bus clock cycle after execution of an **stwcx.** instruction that clears the reservation or as late as the second bus cycle after $\overline{\text{TS}}$ is asserted for a snoop that clears the reservation.

2.10.2 External Cache Intervention (L2_INT)—Input

Following are state and timing descriptions for the external cache intervention (L2_INT) input signal. This signal is not on the 601 or 603.

State Meaning Asserted—The current data transaction required intervention from other bus devices.

Negated—The current data transaction did not require intervention.

Timing Comments Assertion/Negation—This signal is sampled by the processor coincident with the first assertion of $\overline{\text{TA}}$ for a given data tenure.

2.10.3 Time Base Enable (TBEN)—Input

The time base enable (TBEN) input signal is essentially a count enable for the time base. Following are state and timing descriptions for TBEN. This signal is not on the 601.

State Meaning Asserted—The time base should continue clocking.

Negated—The time base should stop counting.

Timing Comments Assertion/Negation—May occur on any cycle and is synchronous with the system clock.

2.10.4 TLBI Synchronization ($\overline{\text{TLBISYNC}}$)—Input

Following are state and timing descriptions for the TLBI synchronization ($\overline{\text{TLBISYNC}}$) input signal. This signal is not on the 601 or 604.

State Meaning Asserted—Instruction execution should stop after **tlbsync** executes.

Negated—Instruction execution can resume after **tlbsync** completes. $\overline{\text{TLBISYNC}}$ is sampled when $\overline{\text{HRESET}}$ negates to select 32-bit data bus mode; if $\overline{\text{TLBISYNC}}$ is negated, 32-bit mode is disabled. See Section 6.3, “32-Bit Data Bus Mode (603).”

Timing Comments Assertion/Negation—May occur on any cycle.

2.11 Power Management Signals

Each processor has input and output signals defined to support low-power modes for the processor and system. These signals are not the same between processors. The signals for each processor are described in this section.

2.11.1 Quiescent Request (QUIESC_REQ)—Output

Following are state and timing descriptions for the quiescent request (QUIESC_REQ) output signal, which the 601 uses to request the system to enter a soft-stop state.

State Meaning Asserted—The 601 is requesting a soft stop state for the system.

Negated—The 601 is not requesting a soft stop state.

Timing Comments Assertion/Negation—May occur at any time.

2.11.2 System Quiesced ($\overline{\text{SYS\_QUIESC}}$)—Input

Following are state and timing descriptions for the system quiesced ($\overline{\text{SYS\_QUIESC}}$) input signal which the system uses to indicate to the 601 that it is ready to enter the soft-stop state.

State Meaning Asserted—Enables soft stop in the 601.

Negated—The soft-stop state is not enabled in the 601. Systems that do not use $\overline{\text{SYS\_QUIESC}}$ should tie it low.

Timing Comments Assertion/Negation—Must meet setup and hold times described in the *PowerPC 601 RISC Microprocessor Hardware Specifications*.

2.11.3 Resume (RESUME)—Input

Following are state and timing descriptions for the RESUME input signal, which the system uses to indicate to the 601 that it can resume normal operations.

State Meaning Asserted—The 601 can resume normal operations after a soft stop.

Negated—The 601 cannot resume normal operations if a soft stop has occurred. Systems that do not use this signal should tie it low.

Timing Comments Assertion—Can occur any time. If asserted asynchronously to the 601 input clock, it must be asserted for at least three clock cycles. If asserted synchronously, it must be asserted at least two clock cycles.

Negation—Can occur after the minimum pulse width has been met.

2.11.4 Quiescent Request ($\overline{\text{QREQ}}$)—Output

Following are state and timing descriptions for the quiescent request ($\overline{\text{QREQ}}$) output signal, which the 603 uses to request that the system enter quiescent state.

State Meaning Asserted—The 603 is requesting all bus activity normally required to be snooped to terminate or to pause so the 603 may enter a low-power (nap or sleep) state. Once the 603 has entered this state it no longer snoops bus activity.

Negated—The 603 is not requesting to enter the quiescent state.

Timing Comments Assertion—Can occur at any time to indicate the request to enter the quiescent state, during which the 603 keeps asserting $\overline{\text{QREQ}}$.

Negation—Can occur whenever quiescent state is not requested.

2.11.5 Quiescent Acknowledge ($\overline{\text{QACK}}$)—Input

Following are state and timing descriptions for the quiescent acknowledge ($\overline{\text{QACK}}$) input, which the system uses to indicate to the 603 that it is ready to enter a low-power state.

State Meaning Asserted—All bus activity that requires snooping has terminated or paused so the 603 can enter a low-power state.

Negated—The 603 cannot enter a low-power state.

Timing Comments Assertion/Negation—May occur on any cycle after the assertion of $\overline{\text{QREQ}}$ and must be held for a minimum of one bus clock cycle.

Start-up—$\overline{\text{QACK}}$ is sampled at the negation of $\overline{\text{HRESET}}$ to select the reduced-pinout mode; if $\overline{\text{QACK}}$ is asserted at start-up, reduced-pinout mode is disabled. See Section 6.4, “Reduced-Pinout Mode (603),” for a description of the reduced pinout mode.

2.11.6 Halted (HALTED)—Output

Following are state and timing descriptions for the HALTED output signal which the 604 uses to indicate to other system components that the processor has been halted.

State Meaning Asserted—The 604 enters idle state as a result of the nap mode. Dispatch and execution stops and the processor bus is idle.

Negated—The processor is not in the idle state.

Timing Comments Assertion/Negation—Synchronous with the processor clock.

2.11.7 Run (RUN)—Input

Following are state and timing descriptions for the RUN input signal, which is used to notify the 604 that snooping is required.

State Meaning Asserted—Forces the internal processor clocks to continue running, even if nap mode is active, allowing bus snooping to occur. HALTED is deasserted to indicate any bus activity and is reasserted to indicate when the processor is idle and when RUN can be deasserted.

Negated—Internal processor clocks can stop running in nap mode.

Timing Comments Assertion—May occur at any time asynchronously to the input clocks. The maximum latency between RUN being asserted and the starting of the internal processor clocks is three bus clock cycles.

Negation—Can occur after the HALTED signal is asserted.

2.11.7.1 Going from Normal to Doze State (604e)

The only state transition allowed from the normal state is to doze state. This transition requires system support. The system must assert RUN for at least 10 bus cycles before the software power management sequence can begin. RUN does not affect 604e operation in the normal state, but does affect operation during the transition from normal to doze state. The software power management sequence is the following code:

sync
mtmsr
isync
branch to the **sync** instruction

The **mtmsr** instruction should modify the power management bit MSR[POW] only. All other MSR values such as the external interrupt enable should be set up before the software power management sequence is begun. When **mtmsr** is executed, the processor waits for its internal state to be idle and then asserts HALTED, at which point the processor is in doze state. When entering doze state, the system must assert RUN for at least 10 bus cycles after HALTED is asserted. When the processor is in doze state, HALTED is deasserted when a snoop-triggered write-back is in progress. The system must keep RUN asserted whenever HALTED is deasserted in doze mode due to a snoop write-back operation.

If the software power management sequence is initiated from the normal state with RUN not asserted, the processor would attempt to go directly to nap state. This transition is not supported and may cause the system to hang later when the processor leaves nap state.

2.11.7.2 Going from Doze to Nap State

For the processor to go from doze to nap state, the system must first ensure that the bus is idle and that HALTED is asserted for at least 10 bus cycles. The system should then deassert RUN and continue to prevent bus grants for at least 10 additional bus cycles, at which point the processor is in nap state and bus transactions can be resumed. The processor does not snoop any subsequent bus transactions.

In going from doze to nap state, the 604e must see the bus idle, which here means that the 604e cannot receive any $\overline{\text{TS}}$ or $\overline{\text{XATS}}$ assertions. The system can ensure this by negating address bus grants to other bus devices.

2.11.7.3 Going from Nap to Doze State

For the processor to go from nap to doze state, the system should ensure the bus is idle for at least 10 bus cycles, assert RUN, and withhold bus grants for at least 10 additional bus cycles. At this point the processor is in the doze state and all bus transactions are snooped.

2.12 Summary of Signal Differences

Table 2-8 lists each signal and describes any substantive differences between different implementations. The clock, power, and test signals are not described in this document. Refer to the user's manual for the particular processor for this information.

Table 2-8. Processor Bus Signal Differences

Signal(s)	Difference
Address Bus Arbitration Signals	
Bus request ($\overline{BR}$)	As an output, assertion occurs when a bus transaction is needed and the device does not have a qualified bus grant. This may occur even if the maximum (two for the 601 and 603, three for the 604) possible pipeline accesses have occurred**.** For the 603, $\overline{BR}$ is asserted for one cycle during the execution of **dcbz** or of a load instruction that hit in the touch load buffer.
Bus grant ($\overline{BG}$)	The 601 recognizes a qualified bus grant on the cycle after $\overline{AACK}$ even if $\overline{ARTRY}$ is asserted as long as the 601 is asserting $\overline{ARTRY}$ and has exclusive ownership of the data associated with the snoop which caused the $\overline{ARTRY}$.
Address bus busy ($\overline{ABB}$)	—
Address Transfer Start Signals	
Transfer start ($\overline{TS}$)	Output—601 and 603—High Impedance occurs one bus clock cycle after $\overline{TS}$ is negated which is coincident with the negation of $\overline{ABB}$. 604—High Impedance occurs one bus clock cycle after the negation of $\overline{TS}$. For the 604, negation is only one bus cycle long, regardless of the $\overline{TS}$-to-$\overline{AACK}$ delay.
Extended address transfer start ($\overline{XATS}$)	Later generations of the 603 do not support direct-store operations. Output—601/603: High Impedance occurs one bus clock cycle after $\overline{XATS}$ is negated which is coincident with the negation of $\overline{ABB}$. 604: High Impedance occurs one bus clock cycle after negation of $\overline{XATS}$. Negation lasts only one bus cycle regardless of the $\overline{XATS}$-to-$\overline{AACK}$ delay.
Address Transfer Signals	
Address bus (A[0–31])	603/604—For bursts, the address presented is double-word–aligned. 601—The address presented is quad-word–aligned.
Address parity (AP[0–3])	601—If address parity check is enabled in the HID0 register, detection of even parity unconditionally causes a checkstop.
Address parity error ($\overline{APE}$)	—
Address Transfer Attribute Signals	
Transfer type (TT[0–4])	Exact meanings of TT[0–4] vary among processors. TT4 is output-only on the 601.
Transfer burst ($\overline{TBST}$)	—

Table 2-8. Processor Bus Signal Differences (Continued)

Signal(s)	Difference
Transfer size (TSIZ[0–2])	—
Transfer code (TC*n*)	The 601 and 603 support only TC[0–1]. The 604 supports TC[0–2]. The exact meanings of these signals vary from processor to processor.
Cache inhibited ($\overline{\text{CI}}$)	—
Write through ($\overline{\text{WT}}$)	—
Global ($\overline{\text{GBL}}$)	Output—603/604: Negated on instruction fetches. 604e: HID0[23] controls $\overline{\text{GBL}}$ for instruction fetches through the address translation mechanism. Input—The 603 must snoop the reservation address register for global and nonglobal address transfers because **lwarx**/**stwcx.** require snoops on castouts and snoop pushes (nonglobal). Snoops with $\overline{\text{GBL}}$ = 1 do not affect cache state.
Cache set element (CSE*n*)	The number of CSE signals corresponds to the cache structure. 601: CSE[0–2]; 603 (not the 603e): CSE; 603e/604: CSE[0–1]. CSE signals are not meaningful during data cache touch load operations on a 603.
High-priority snoop request ($\overline{\text{HP\_SNP\_REQ}}$)	601 only
Address Transfer Termination Signals	
Address acknowledge ($\overline{\text{AACK}}$)	Input—The 604 supports sampling $\overline{\text{ARTRY}}$ as early as the second cycle after $\overline{\text{TS}}$.
Address retry ($\overline{\text{ARTRY}}$)	Negation timing is processor specific.
Shared ($\overline{\text{SHD}}$)	The 603 does not support shared data.
Data Bus Arbitration Signals	
Data bus grant ($\overline{\text{DBG}}$)	Some conditions do not apply to the 604/604e for data streaming mode.
Data bus write only ($\overline{\text{DBWO}}$)	—
Data bus busy ($\overline{\text{DBB}}$)	—
Data Transfer Signals	
Data bus (DH[0–31];DL[0–31])	—
Data bus parity (DP[0–7])	For the 601, if data parity check is enabled in the HID0 register, detection of even parity unconditionally causes a checkstop in the 601.
Data parity error ($\overline{\text{DPE}}$)	—
Data bus disable ($\overline{\text{DBDIS}}$)	Signal defined after 601.
Data Transfer Termination Signals	
Transfer acknowledge ($\overline{\text{TA}}$)	Input/negation—When the 603 is configured for 1:1 clock mode and is performing a burst read into data cache, the 603 requires one wait state between the assertion of $\overline{\text{TS}}$ and the first assertion of $\overline{\text{TA}}$ for that transaction. If no-$\overline{\text{DRTRY}}$ mode is also selected, the 603 requires two wait states.

Table 2-8. Processor Bus Signal Differences (Continued)

Signal(s)	Difference
Data retry (DRTRY)	Input/start-up—Used at power-on to select no-DRTRY mode for 603, data streaming mode for 604, and data streaming mode or no-DRTRY mode for 604e. For 603 and 604, DRTRY is sampled at the negation of HRESET; if DRTRY is asserted, no-DRTRY mode is selected (603/604e). If DRTRY is negated at start-up, DRTRY is enabled. If no-DRTRY or data streaming mode is selected, DRTRY must be negated during normal operation (after HRESET). No-DRTRY mode provides a one-cycle faster reads; data streaming allows consecutive bursts.
Transfer error acknowledge (TEA)	604—In data streaming mode, the 604 does not recognize TEA the cycle after TA during a read operation due to the absence of a DRTRY assertion opportunity. TEA should be asserted for one cycle only.
System Status Signals	
Interrupt (INT)	601—INT may be negated after a minimum of three processor clock cycles.
System management interrupt (SMI)	Supports the system management interrupt not defined by the PowerPC architecture; not implemented on the 601.
Machine check interrupt (MCP)	This signal is not defined for the 601.
Checkstop input (CKSTP_IN)	Early versions of the 603 identified this signal as CKSTP.
Checkstop output (CKSTP_OUT)	Early versions of the 603 identified this signal as CHECKSTOP.
Hard reset (HRESET)	After assertion, output drivers are released to high impedance within five clocks (three clocks for the 601) after the assertion of HRESET.
Soft reset (SRESET)	Negation may occur any time after the minimum soft reset pulse width of 2 (10 for the 601) bus cycles has been met.
Processor State Signals	
Reservation (RSRV)	604/604e. RSRV is asserted as late as the fourth cycle after AACK for a read-atomic operation if the **lwarx** instruction requires a read-atomic operation.
External cache intervention (L2_INT)	New feature on 604
Time base enable (TBEN)	Time base did not exist on 601.
TLBI synchronization (TLBISYNC)	Supports a 603-specific instruction; used at power-on to select 32-bit bus mode
Power Management Signals	
Quiescent request (QUIESC_REQ)	601 only
Quiescent request (QREQ)	603 only. This signal is used at power-on to select a reduced pin mode.
Halted (HALTED)	Power management for the 604
System quiesced (SYS_QUIESC)	601 only
Resume (RESUME)	604 only
Quiescent acknowledge (QACK)	603 only
Run (RUN)	604 only

Chapter 3
Memory Access Protocol

Memory accesses can occur in single (1–8 bytes) and four-beat (32 bytes) burst data transfers. System components can direct these accesses to the system memory hierarchy or to I/O devices as memory-mapped I/O. The address and data buses are decoupled for memory accesses to support pipelining and split transactions. The PowerPC 601 and 603 processors can pipeline as many as two transactions; the PowerPC 604 processor can pipeline as many as three. These processors have limited support for out-of-order split transactions.

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

The 601 and 604 provide multiprocessor support through coherency mechanisms that provide snooping, external control of the on-chip cache and translation lookaside buffers (TLBs), and support for a secondary cache. Multiprocessor software support is provided through the use of atomic memory operations.

Typically, memory accesses are weakly-ordered—sequences of operations, including load/store string and load/store multiple instructions, do not necessarily complete in the order they begin—maximizing the efficiency of the bus without sacrificing coherency of the data. The processors allow read operations to precede store operations (except where a dependency exists). A processor can be signaled to perform a pending write ahead of pending reads. The 604 performs snoop push operations ahead of all other bus operations. Because the processor can dynamically optimize run-time ordering of load/store traffic, overall performance is improved.

The Synchronize (**sync**) or Enforce In-Order Execution of I/O (**eieio**) instructions can be used to enforce strong ordering.

The following sections describe how the processor interface operates, provide detailed timing diagrams that illustrate how the signals interact, and include a collection of more general timing diagrams of typical bus operations.

Figure 3-1 is a legend of conventions used in the timing diagrams.

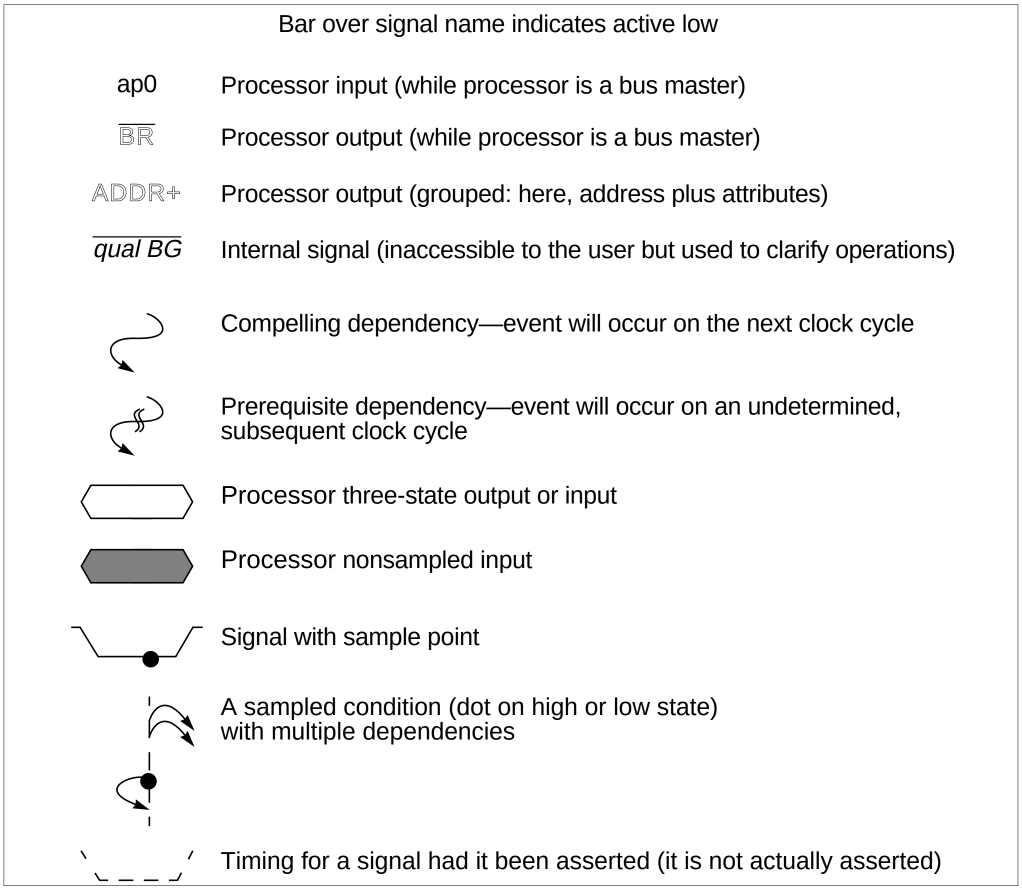

Figure 3-1. Timing Diagram Legend

Signals on this interface are synchronous—all processor input signals are sampled and output signals are driven on the rising edge of the bus clock cycle (see the processor hardware specifications for exact timing information).

3.1 Bus Protocol

Figure 3-2 shows the memory access bus protocol for the 601, 603, and 604. Memory accesses are divided into address and data tenures, each of which is comprised of three phases—bus arbitration, transfer, and termination. Address and data tenures are independent and, as indicated in Figure 3-2, can overlap due to the ability to start a data tenure before the address tenure ends. The independence of these operations permits address pipelining and split-bus transactions to be implemented at the system level. These processors support one- and four-beat transfers. Figure 3-2 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 blocks are also supported, and on the 603, eight-beat bursts can be used to transfer an eight-word cache block when the processor is operating in 32-bit data bus mode. Burst operations require data transfer termination signals for each beat.

Address-only transactions are used to broadcast synchronizing and cache control operations, especially in multiprocessor systems.

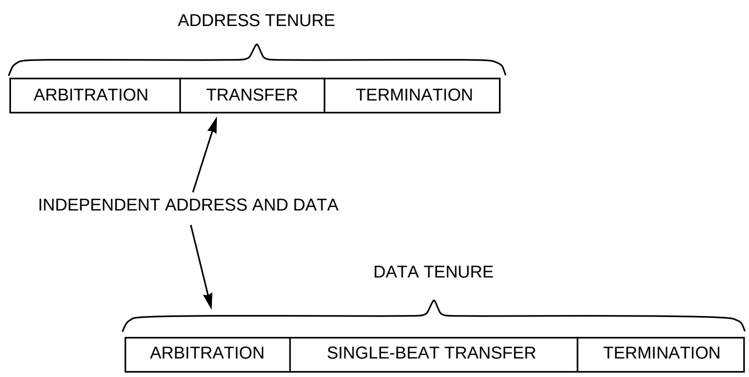

Figure 3-2. Overlapping Tenures on the Processor Bus for a Single-Beat Transfer

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 mastership is obtained, the address bus master transfers the address, transfer attributes, and parity information on the address bus. Address signals and transfer attribute signals control the address transfer. 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 a data tenure, the master arbitrates for data bus mastership.
  - — Transfer: After mastership is obtained, the data bus master 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. In single-beat transactions, data termination signals also indicate the end of the tenure, while in burst accesses, data termination signals apply to individual beats and indicate the end of the tenure only after the final data beat.

Processors can generate address-only bus operations during execution of certain instructions (for example **dcbz**, **sync**, **eieio**, **tlbie**, and **lwarx**). Address-only operations are given more support on processors intended for multiprocessor systems. The ability to retry address tenures provides an efficient snooping protocol for maintaining coherency in systems with multiple memory systems (including caches).

Although address and data transfers are separate, there is no explicit tagging mechanism to associate a data transfer with its address transfer. Addresses and data are generally transferred in the same order. However, the data bus write only ($\overline{\text{DBWO}}$) signal allows writes to transfer ahead of reads. The designer of a multiple processor system can provide any ordering, as long as each processor transfers its addresses and data in same order and memory is kept coherent.

3.1.1 Arbitration Signals

Arbitration for both address and data bus mastership is performed by a central, external arbiter and minimally by the arbitration signals shown in Section 2.1, “Address Bus Arbitration Signals,” and Section 2.6, “Data Bus Arbitration Signals.” Most arbiter implementations require additional signals to coordinate bus master/slave/snooping activities. Note that address bus busy ($\overline{\text{ABB}}$) and data bus busy ($\overline{\text{DBB}}$) are bidirectional signals. They are processor inputs unless it is master of one or both buses; they must be connected high through pull-up resistors so that they remain negated when no devices have control of the buses. Table 3-1 shows the bus arbitration signals.

Table 3-1. Number of Bus Arbitration Signals

Signal	I/O	Signal Connection Requirements
$\overline{\text{BR}}$	Output	One per master
$\overline{\text{BG}}$	Input	One per master
$\overline{\text{ABB}}$	Both	Common among masters
$\overline{\text{DBG}}$	Input	One per master
$\overline{\text{DBWO}}$	Input	One per processor
$\overline{\text{DBB}}$	Both	Common among masters (one per master if data streaming is used across multiple masters)

Address bus arbitration signals are described as follows:

- $\overline{\text{BR}}$ (bus request)—Assertion indicates that a device wants address bus mastership.
- $\overline{\text{BG}}$ (bus grant)—Assertion indicates the device can, with the proper qualification, take mastership of the address bus. See Section 2.1.2, “Bus Grant (BG)—Input.”
- $\overline{\text{ABB}}$ (address bus busy)—Assertion identifies the address bus master.

Data bus arbitration signals are described as follows:

- $\overline{\text{DBG}}$ (data bus grant)—Indicates that the device can, with the proper qualification, take data bus mastership. See Section 2.6.1, “Data Bus Grant (DBG)—Input.”
- $\overline{\text{DBWO}}$ (data bus write only)—Assertion indicates that the processor may perform the data bus tenure for an outstanding write address even if a read address is pipelined before the write address.
- $\overline{\text{DBB}}$ (data bus busy)—Assertion indicates that the device is data bus master. Processors assume data bus mastership if they need the data bus and are given a qualified data bus grant.

Note that when the 604 uses data streaming, $\overline{\text{DBB}}$ works only as an output and is driven in the same manner as before. If 604 systems use data streaming across multiple devices, $\overline{\text{DBB}}$ must not be common among processors to avoid contention problems when one processor negates $\overline{\text{DBB}}$ while another asserts it.

3.1.2 Address Pipelining and Split-Bus Transactions

This protocol provides independent address and data bus capability to support pipelined and split-bus transaction system organizations. Pipelining allows the address tenure of a bus transaction to begin before the data tenure of the previous transaction finishes. Split-bus transactions allow other bus activity to occur (either from the same or from different devices) between the address and data tenures of a transaction.

Although it does not inherently reduce memory latency, address pipelining and split-bus transactions can greatly improve bus/memory throughput, and are especially effective in multiprocessor implementations where bus bandwidth is an important measurement of system performance.

The design of the external arbiter affects pipelining by regulating address bus grant ($\overline{\text{BG}}$), data bus grant ($\overline{\text{DBG}}$), and address acknowledge ($\overline{\text{AACK}}$) signals. For example, a one-level pipeline is enabled by asserting $\overline{\text{AACK}}$ to the current address bus master and granting address bus mastership to the next requesting device before the current data bus tenure completes. For example, a two-level pipeline lets two additional address tenures occur before the current data bus tenure completes.

The 604 can pipeline its transactions to a depth of two levels (intraprocessor pipelining) and the 601 and 603 can pipeline transactions to a depth of one level. The bus protocol does not limit the levels of pipelining between multiple devices (interprocessor pipelining); the external arbiter controls pipeline depth and synchronization between masters and slaves.

In a pipelined implementation, data bus tenures stay in strict order with respect to address tenures except when $\overline{\text{DBWO}}$ is used to move write data tenures ahead of read data tenures. However, external hardware can further decouple the address and data buses, allowing data tenures to occur out of order with respect to address tenures. This requires some form of system tag to associate an out-of-order data transaction with its address transaction (not defined for this processor interface). Each processor's bus requests and data bus grants can be used to implement tags to support interprocessor, out-of-order transactions.

3.2 Address Bus Tenure

This section describes the three phases of the address tenure—address bus arbitration, address transfer, and address termination.

3.2.1 Address Bus Arbitration

When a device needs bus access but does not have a qualified bus grant, it asserts $\overline{\text{BR}}$ until the bus is available and the device is granted mastership. The external arbiter must grant master-elect status to the potential master by asserting $\overline{\text{BG}}$. The device requesting the bus determines that the bus is available as a qualified bus grant; refer to Section 2.1.2, "Bus Grant (BG)—Input." The processor assumes address bus mastership and asserts $\overline{\text{ABB}}$ when it receives a qualified bus grant as shown in Figure 3-3.

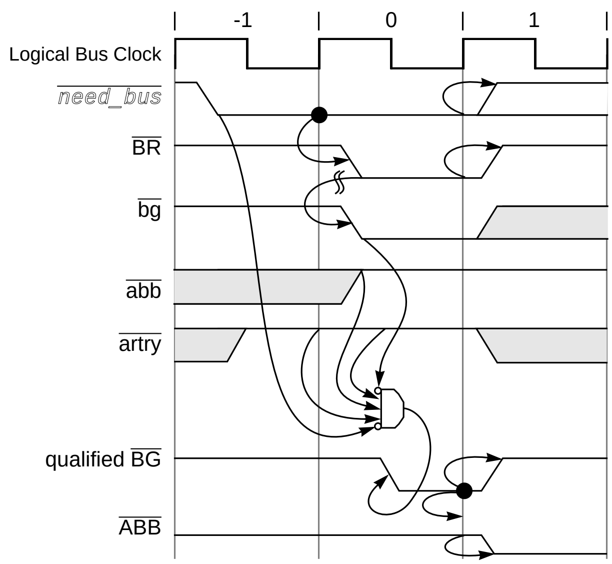

Figure 3-3. Address Bus Arbitration Showing Qualified Bus Grant

External arbiters must allow only one device at a time to be address bus master. In systems in which no other device can be a master, $\overline{\text{BG}}$ can be grounded (always asserted) to continually grant address bus mastership to the processor.

Figure 3-4 shows bus parking; a qualified bus grant exists on the clock edge following a need_bus condition eliminating the two bus clock cycles required for arbitration. The processor negates ABB for at least one bus clock cycle after AACK is asserted, even if it is parked and another transaction is pending. Typically, the most recent bus master remains parked; however, system designers can choose other schemes, such as providing unrequested bus grants in situations where it is easy to correctly predict the next device requesting bus mastership.

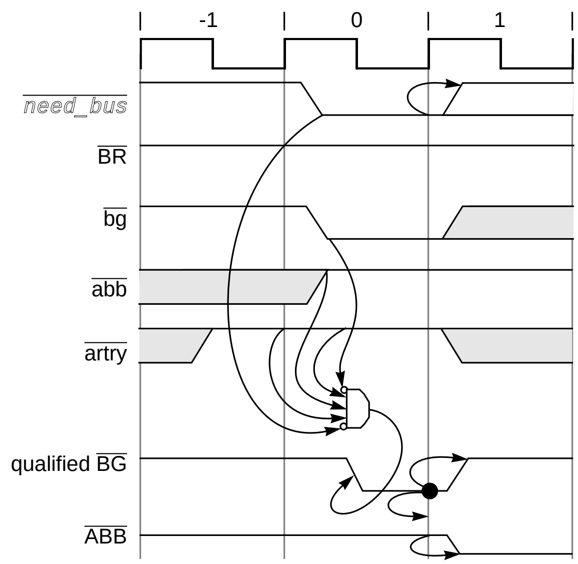

Figure 3-4. Address Bus Arbitration Showing Bus Parking

When the processor receives a qualified bus grant, it assumes address bus mastership by asserting $\overline{\text{ABB}}$ and negating the $\overline{\text{BR}}$ output signal. Meanwhile, the processor drives the requested address onto the address bus and asserts $\overline{\text{TS}}$ to indicate the start of a new transaction. To avoid the bus hogging these processors, always assert $\overline{\text{ABB}}$ and $\overline{\text{TS}}$ simultaneously and negate $\overline{\text{ABB}}$ the clock cycle following assertion of $\overline{\text{AACK}}$; however, the processors accommodate systems in which $\overline{\text{ABB}}$ is asserted early or removed late.

When designing external bus arbitration logic, note that the processor may assert $\overline{\text{BR}}$ without using the bus after it receives the qualified bus grant. For example, if the 604 snoops an access that cancels the reservation associated with a queued read-with-intent-to-modify-atomic (RWITMA) operation and for which it has asserted $\overline{\text{BR}}$, when the 604 is granted the bus, it no longer needs to perform the RWITMA operation; therefore, the 604 does not assert $\overline{\text{ABB}}$ and does not use the bus for the read operation.

The 604 asserts $\overline{\text{BR}}$ for at least one clock cycle in these instances.

3.2.2 Address Transfer

During an address transfer, the physical address and transfer attributes pass from the bus master to the slave device(s). Snooping logic may monitor the transfer to enforce cache coherency. The signal groups used in address transfers include the following:

- Address transfer start signal—Transfer start ($\overline{TS}$). See Section 2.2, "Address Transfer Start Signals."
- Address transfer signals—Address bus (A[0–31]), address parity (AP[0–3]), and address parity error (APE); see Section 2.3, "Address Transfer Signals."
- Address transfer attribute signals—Transfer type (TT[0–4]), transfer burst (TBST), transfer size (TSIZ[0–2]), transfer code (TC*n*), cache inhibit (CI), write-through ($\overline{WT}$), global ($\overline{GBL}$), and cache set element (CSE*n*); see Section 2.4, "Address Transfer Attribute Signals."

Figure 3-5 shows the timing for all of these signals. Except for $\overline{TS}$ and $\overline{APE}$, address transfer and address transfer attribute signal timing is identical. These signals are represented by the line labeled 'ADDR+'. Asserting $\overline{TS}$ indicates the master has begun an address transfer and that the address and transfer attributes are valid (within the context of a synchronous bus). These processors always assert $\overline{TS}$ coincident with $\overline{ABB}$. As an input to the processors from other system masters, $\overline{TS}$ need not coincide with assertion of $\overline{ABB}$, but can be asserted after it is asserted; these processors track this scenario correctly.

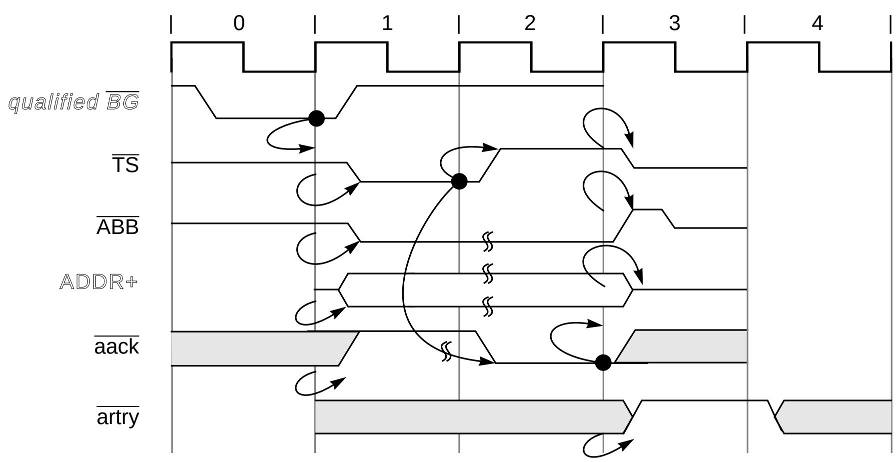

Figure 3-5. Address Bus Transfer

The address is transferred in bus clock cycles 1 and 2 (arbitration occurs in clock cycle 0). $\overline{TS}$ is asserted in clock cycle 1 and then negated. Address and attribute signals are driven valid coincident with the asserting of $\overline{TS}$ and held until the address transfer ends. The processor asserts $\overline{ABB}$ during the transfer. $\overline{AACK}$ is asserted to the processor the cycle after assertion of $\overline{TS}$ (shown by the dependency line). This is the shortest duration of an address transfer; it can be extended by a slave delaying assertion of $\overline{AACK}$.

3.2.2.1 Address Bus Parity

The 60x processors always generate one bit of correct odd-byte parity for each of the four bytes of address when a valid address is on the bus. The calculated values are placed on the AP[0–3] outputs when the processor is address bus master. If the processor is not the master, $\overline{\text{TS}}$ and $\overline{\text{GBL}}$ are asserted together, the transaction type is one the processor snoops, and the calculated values are compared with inputs AP[0–3]. If address bus parity checking is enabled (refer to the HID description for each processor), a parity error causes a machine check (checkstop on the 601) if MSR[ME] is set or checkstop if it is cleared. If address bus parity checking is disabled, no action is taken. In either case, $\overline{\text{APE}}$ is asserted if even parity is detected. The 603 does not assert $\overline{\text{APE}}$ if address parity checking is disabled.

3.2.2.2 Address Transfer Attribute Signals

The address transfer attribute signals, TT[0–4], $\overline{\text{TBST}}$, TSIZ[0–2], and TC*n*, are fully described in Section 2.4, "Address Transfer Attribute Signals," and are summarized below.

3.2.2.2.1 Transfer Type (TT[0–4]) Signals

Snooping logic should fully decode the transfer type signals if $\overline{\text{GBL}}$ is asserted. Slave devices can sometimes use individual transfer type signals without fully decoding the group. Table 2-1 describes encodings for the transfer type signals.

3.2.2.2.2 Transfer Size (TSIZ[0–2]) Signals

The TSIZ[0–2] signals indicate the size of the requested data transfer as shown in Table 2-2. These signals can be used with $\overline{\text{TBST}}$ and A[29–31] to determine which portion of the data bus has valid data for a write transaction or which portion of the bus should contain valid data for a read transaction. In general, processors do not produce 5-, 6-, or 7-byte transfers. The 601 allows unaligned floating-point operations to produce 5-, 6-, or 7-byte transfers, but use of this feature is discouraged.

The PowerPC architecture allows storage combining, but it is not supported in the 601 and 603. The 604 combines only stores to adjacent aligned words resulting from a cache-inhibited store multiple word (**stmw**) instruction. These combined words are presented to the bus as a normal double-word store in memory order. Storage combining of other sizes (for example, three adjacent half words to make a 6-byte transfer) are not implemented.

Coherency size is defined as 32 bytes (one cache block). Data transfers that cross a 32-byte–aligned boundary must present a new address to the bus at that boundary (for coherency consideration) or must operate as noncoherent data with respect to the processor.

3.2.2.3 Burst Ordering during Data Transfers

During burst data transfer operations for these processors, 32 bytes of data (one cache block) are transferred to or from the processor cache in a specific order. Burst transfers are always presented by the processor with a double-word–aligned address (in other words A[29–31] is 0b000). For burst reads, these processors request the critical double word. A memory controller must transfer this word first, followed by those words in increasing memory addresses, and wrapping around to the beginning of the cache block as required. Table 3-2 describes burst read orderings. For burst writes, processors present the first address of the block (A[27–31] is 0b00000).

Table 3-2. Processor Read Burst Ordering

Data Transfer	Processor 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 processor.

3.2.2.4 Effect of Alignment in Data Transfers

This section describes the various combinations of transfer size, address, and byte lanes used by these processors. Also shown is the difference in behavior of PowerPC processors with an 8-byte data bus and the 603 with a 4-byte data bus mode. Aligned transfers are those whose address is an integer multiple of the data's size. For example, a 4-byte transfer has an address of 0bx...xx00. The PowerPC architecture allows flexibility to handle alignment errors either in hardware or software (a program exception). See the user's manual for each processor. Table 3-3 lists aligned transfers (shown by an A) generated by a PowerPC processor with a 64-bit data bus. For example, 1-byte data is always aligned. The table also shows byte lanes used for a 4-byte word transfer, and that only two addresses are aligned.

Table 3-3. Aligned Data Transfers for 64-Bit Data Bus

Transfer Size	TSIZ[0–2]	A[29–31]	Data Bus Byte Lane(s)							
			0	1	2	3	4	5	6	7
Byte	0 0 1	000	A	—	—	—	—	—	—	—
	0 0 1	001	—	A	—	—	—	—	—	—
	0 0 1	010	—	—	A	—	—	—	—	—
	0 0 1	011	—	—	—	A	—	—	—	—
	0 0 1	100	—	—	—	—	A	—	—	—
	0 0 1	101	—	—	—	—	—	A	—	—
	0 0 1	110	—	—	—	—	—	—	A	—
	0 0 1	111	—	—	—	—	—	—	—	A
Half word	0 1 0	000	A	A	—	—	—	—	—	—
	0 1 0	010	—	—	A	A	—	—	—	—
	0 1 0	100	—	—	—	—	A	A	—	—
	0 1 0	110	—	—	—	—	—	—	A	A
Word	1 0 0	000	A	A	A	A	—	—	—	—
	1 0 0	100	—	—	—	—	A	A	A	A
Double word	0 0 0	000	A	A	A	A	A	A	A	A

Notes: A: Byte lane used

—: Byte lane not used

Table 3-4 lists aligned transfers that can occur on the bus and are generated by a 603 in 32-bit data bus mode. Note that the two aligned word transfers are always transferred on byte lanes 0–3 and that a double-word transfer takes two beats.

Table 3-4. Aligned Data Transfers for 32-Bit Data Bus

Transfer Size	Required Bus Transfers	TSIZ [0–2]	A[29–31]	Data Bus Byte Lane(s)							
				0	1	2	3	4	5	6	7
Byte	One access	0 0 1	000	A	—	—	—	X	X	X	X
	One access	0 0 1	001	—	A	—	—	X	X	X	X
	One access	0 0 1	010	—	—	A	—	X	X	X	X
	One access	0 0 1	011	—	—	—	A	X	X	X	X
	One access	0 0 1	100	A	—	—	—	X	X	X	X
	One access	0 0 1	101	—	A	—	—	X	X	X	X
	One access	0 0 1	110	—	—	A	—	X	X	X	X
	One access	0 0 1	111	—	—	—	A	X	X	X	X
Half word	One access	0 1 0	000	A	A	—	—	X	X	X	X
	One access	0 1 0	010	—	—	A	A	X	X	X	X
	One access	0 1 0	100	A	A	—	—	X	X	X	X
	One access	0 1 0	110	—	—	A	A	X	X	X	X
Word	One access	1 0 0	000	A	A	A	A	X	X	X	X
	One access	1 0 0	100	A	A	A	A	X	X	X	X
Double word	First access	0 0 0	000	A	A	A	A	X	X	X	X
	Second access	0 0 0	000	A	A	A	A	X	X	X	X

Notes: A: Byte lane used
X: Byte lane not used in 32-bit mode
—: Byte lane not used

The processors support misaligned memory operations to varying degrees, however, it is strongly recommended that software attempt to align code and data where possible. In particular, load/store multiple and load/store string instructions that generate misaligned accesses can greatly affect 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, 0bx...x1). Although most of these operations hit in the primary cache (or generate burst memory operations if they miss), the processor interface supports misaligned transfers. There are three approaches for handling these transfers depending upon the processor and the data bus width.

The 601 transfers misaligned data in one or two bus cycles, as shown in Table 3-5. Misaligned data that does not cross a double-word boundary is transferred in a single access. Those that cross a double-word boundary take two accesses. Misaligned double-word floating-point loads and stores are outside the architecture and are not shown in this table even though they are supported by the 601.

Table 3-5. Misaligned Data Transfers for the PowerPC 601 Processor

Transfer Size	Required Bus Transfers	TSIZ[0–2]	A[29–31]	Data Bus Byte Lanes							
				0	1	2	3	4	5	6	7
Two bytes	One access	0 1 0	0 0 1	—	A	A	—	—	—	—	—
	One access	0 1 0	0 1 1	—	—	—	A	A	—	—	—
	One access	0 1 0	1 0 1	—	—	—	—	—	A	A	—
	First access	0 0 1	1 1 1	—	—	—	—	—	—	—	A
	Second access	0 0 1	0 0 0	A	—	—	—	—	—	—	—
Three bytes	One access	0 1 1	0 0 0	A	A	A	—	—	—	—	—
	One access	0 1 1	0 0 1	—	A	A	A	—	—	—	—
	One access	0 1 1	0 1 0	—	—	A	A	A	—	—	—
	One access	0 1 1	0 1 1	—	—	—	A	A	A	—	—
	One access	0 1 1	1 0 0	—	—	—	—	A	A	A	—
	One access	0 1 1	1 0 1	—	—	—	—	—	A	A	A
	First access	0 1 0	1 1 0	—	—	—	—	—	—	A	A
	Second access	0 0 1	0 0 0	A	—	—	—	—	—	—	—
	First access	0 0 1	1 1 1	—	—	—	—	—	—	—	A
	Second access	0 1 0	0 0 0	A	A	—	—	—	—	—	—
Four bytes	One access	1 0 0	0 0 1	—	A	A	A	A	—	—	—
	One access	1 0 0	0 1 0	—	—	A	A	A	A	—	—
	One access	1 0 0	0 1 1	—	—	—	A	A	A	A	—
	First access	0 1 1	1 0 1	—	—	—	—	—	A	A	A
	Second access	0 0 1	0 0 0	A	—	—	—	—	—	—	—
	First access	0 1 0	1 1 0	—	—	—	—	—	—	A	A
	Second access	0 1 0	0 0 0	A	A	—	—	—	—	—	—
	First access	0 0 1	1 1 1	—	—	—	—	—	—	—	A
	Second access	0 1 1	0 0 0	A	A	A	—	—	—	—	—

Notes: A: Byte lane used; —: Byte lane not used

The 603 and 604 transfer misaligned data in one or two bus accesses, as shown in Table 3-6. As long as the misaligned transfer does not cross a word boundary, these processors can transfer the data for the misaligned address in one access. The two-byte transfer at address 0bx...x001 is such a case. An attempt to address misaligned data that crosses a word boundary requires two bus transfers to access the data.

Table 3-6. Misaligned Data Transfers for PowerPC 603/ 604 Processors

Transfer Size	Required Bus Transfers	TSIZ[0–2]	A[29–31]	Data Bus Byte Lanes							
				0	1	2	3	4	5	6	7
Two bytes	One access	0 1 0	0 0 1	—	A	A	—	—	—	—	—
	First access	0 0 1	0 1 1	—	—	—	A	—	—	—	—
	Second access	0 0 1	1 0 0	—	—	—	—	A	—	—	—
	One access	0 1 0	1 0 1	—	—	—	—	—	A	A	—
	First access	0 0 1	1 1 1	—	—	—	—	—	—	—	A
	Second access	0 0 1	0 0 0	A	—	—	—	—	—	—	—
Three bytes	One access	0 1 1	0 0 0	A	A	A	—	—	—	—	—
	One access	0 1 1	0 0 1	—	A	A	A	—	—	—	—
	First access	0 1 0	0 1 0	—	—	A	A	—	—	—	—
	Second access	0 0 1	1 0 0	—	—	—	—	A	—	—	—
	First access	0 0 1	0 1 1	—	—	—	A	—	—	—	—
	Second access	0 1 0	1 0 0	—	—	—	—	A	A	—	—
	One access	0 1 1	1 0 0	—	—	—	—	A	A	A	—
	One access	0 1 1	1 0 1	—	—	—	—	—	A	A	A
	First access	0 1 0	1 1 0	—	—	—	—	—	—	A	A
	Second access	0 0 1	0 0 0	A	—	—	—	—	—	—	—
	First access	0 0 1	1 1 1	—	—	—	—	—	—	—	A
	Second access	0 1 0	0 0 0	A	A	—	—	—	—	—	—

Notes: A: Byte lane used; —:Byte lane not used

Table 3-6. Misaligned Data Transfers for PowerPC 603/ 604 Processors (Continued)

Transfer Size	Required Bus Transfers	TSIZ[0–2]	A[29–31]	Data Bus Byte Lanes							
				0	1	2	3	4	5	6	7
Four bytes	First access	0 1 1	0 0 1	—	A	A	A	—	—	—	—
	Second access	0 0 1	1 0 0	—	—	—	—	A	—	—	—
	First access	0 1 0	0 1 0	—	—	A	A	—	—	—	—
	Second access	0 1 0	1 0 0	—	—	—	—	A	A	—	—
	First access	0 0 1	0 1 1	—	—	—	A	—	—	—	—
	Second access	0 1 1	1 0 0	—	—	—	—	A	A	A	—
	First access	0 1 1	1 0 1	—	—	—	—	—	A	A	A
	Second access	0 0 1	0 0 0	A	—	—	—	—	—	—	—
	First access	0 1 0	1 1 0	—	—	—	—	—	—	A	A
	Second access	0 1 0	0 0 0	A	A	—	—	—	—	—	—
	First access	0 0 1	1 1 1	—	—	—	—	—	—	—	A
	Second access	0 1 1	0 0 0	A	A	A	—	—	—	—	—

Notes: A: Byte lane used; —:Byte lane not used

In 32-bit data bus mode, the 603 transfers misaligned data in one or two bus cycles using only byte lanes 0–3, as shown in Table 3-7. If the attempted transfer does not cross a word boundary, the processor can transfer the data for the misaligned address in one access. The two-byte transfer at address 0bx...x001 is such a case. Accessing data that crosses a word boundary, such as a two-byte transfer at address 0bx...x011, takes two bus transfers.

Table 3-7. Misaligned Data Transfers for 603 in 32-Bit Mode

Transfer Size	Required Bus Transfers	TSIZ[0–2]	A[29–31]	Data Bus Byte Lanes							
				0	1	2	3	4	5	6	7
Two bytes	One access	0 1 0	0 0 1	—	A	A	—	X	X	X	X
	First access	0 0 1	0 1 1	—	—	—	A	X	X	X	X
	Second access	0 0 1	1 0 0	A	—	—	—	X	X	X	X
	One access	0 1 0	1 0 1	—	A	A	—	X	X	X	X
	First access	0 0 1	1 1 1	—	—	—	A	X	X	X	X
	Second access	0 0 1	0 0 0	A	—	—	—	X	X	X	X
Three bytes	One access	0 1 1	0 0 0	A	A	A	—	X	X	X	X
	One access	0 1 1	0 0 1	—	A	A	A	X	X	X	X
	First access	0 1 0	0 1 0	—	—	A	A	X	X	X	X
	Second access	0 0 1	1 0 0	A	—	—	—	X	X	X	X
	First access	0 0 1	0 1 1	—	—	—	A	X	X	X	X
	Second access	0 1 0	1 0 0	A	A	—	—	X	X	X	X
	One access	0 1 1	1 0 0	A	A	A	—	X	X	X	X
	One access	0 1 1	1 0 1	—	A	A	A	X	X	X	X
	First access	0 1 0	1 1 0	—	—	A	A	X	X	X	X
	Second access	0 0 1	0 0 0	A	—	—	—	X	X	X	X
	First access	0 0 1	1 1 1	—	—	—	A	X	X	X	X
	Second access	0 1 0	0 0 0	A	A	—	—	X	X	X	X
Four bytes	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
	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
	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
	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
	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
	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

Notes: A: Byte lane used; X: Byte lane not used in 32-bit mode; —: Byte lane not used in transfer

3.2.2.4.1 Alignment of External Control Instructions

The **eciwx** and **ecowx** instructions always transfer four bytes of data. However, if the **eciwx** or **ecowx** addresses data that crosses a double-word boundary on the 601 or any word boundary on the 603 or 604, the processor generates two bus operations, each transferring fewer than four bytes.

For the first bus operation, bits A[29–31] equals EA[29–31] of the instruction, (0b101, 0b110, or 0b111 for the 601 or 0bx01, 0bx10, or 0bx11 for the 603 or 604). The size associated with the first bus operation is 3, 2, or 1 bytes, respectively. For the second bus operation, the system must determine how many bytes were transferred on the first bus operation to determine the size of the second operation. Address bits A[29–31] equal 0b000 and the operation transfers 1, 2, or 3 bytes, respectively. For both operations, $\overline{\text{TBST}}$ and TSIZ[0–2] are redefined to specify the resource ID (RID), copied from EAR[28–31]. For **eciwx**/**ecowx** operations, the state of EAR[28] is presented by the $\overline{\text{TBST}}$ signal without inversion (if EAR[28] = 1, $\overline{\text{TBST}}$ is asserted).

Furthermore, the two bus operations associated with such a misaligned external control instruction are not atomic. That is, the processor can initiate other types of memory operations between the two transfers. Also, the two bus operations associated with a misaligned **ecowx** can be interrupted by an **eciwx** bus operation, and vice versa. The processor guarantees that the two operations associated with a misaligned **ecowx** cannot be interrupted by another **ecowx** operation; and likewise for **eciwx**.

Because a misaligned external control address is considered a programming error, the system may choose to assert $\overline{\text{TEA}}$ or otherwise cause an exception when a misaligned external control bus operation occurs.

3.2.3 Address Transfer Termination

An address tenure is terminated when completed with the assertion of $\overline{\text{AACK}}$. The processor does not terminate the address transfer until the $\overline{\text{AACK}}$ input is asserted; therefore, the system can extend the address transfer phase by delaying assertion of $\overline{\text{AACK}}$. The $\overline{\text{AACK}}$ signal can be asserted as early as the bus clock cycle following $\overline{\text{TS}}$ (see Figure 3-5), for a minimum address tenure of two bus cycles. Note that $\overline{\text{AACK}}$ must be asserted for only one bus clock cycle.

The address transfer can be terminated with the requirement to retry if $\overline{\text{ARTRY}}$ is asserted any time during the address tenure and through the cycle following $\overline{\text{AACK}}$. If an address retry is required, the $\overline{\text{ARTRY}}$ response is asserted by a bus snooping device as early as the second cycle after $\overline{\text{TS}}$ is asserted. Once asserted, $\overline{\text{ARTRY}}$ must remain asserted through the cycle after the assertion of $\overline{\text{AACK}}$. The assertion of $\overline{\text{ARTRY}}$ during the cycle after the assertion of $\overline{\text{AACK}}$ is called a qualified $\overline{\text{ARTRY}}$. Assertion of $\overline{\text{ARTRY}}$ during the address tenure is referred to as an early $\overline{\text{ARTRY}}$.

If the bus master recognizes an $\overline{\text{ARTRY}}$ and the data tenure has begun, it terminates the data tenure immediately even if data has been received. If the assertion of $\overline{\text{ARTRY}}$ is received up to or on the bus cycle following the first (or only) assertion of $\overline{\text{TA}}$ for the data tenure, the processor ignores the first data beat; if it is a load operation, it does not forward data internally to the cache and execution units. If the 604 is in fast-L2 mode, $\overline{\text{TA}}$ should not be asserted prior to the valid $\overline{\text{ARTRY}}$ cycle. If $\overline{\text{ARTRY}}$ is asserted after the first (or only) assertion of $\overline{\text{TA}}$, improper operation of the bus interface may result.

As a bus master, the processor responds to an assertion of $\overline{\text{ARTRY}}$ by aborting the bus transaction and re-requesting the bus. The assertion causes both the address and data tenures to be rerun. After recognizing an assertion of $\overline{\text{ARTRY}}$ and aborting a transaction, the processor may not run the same transaction the next time it is granted the bus.

As a snooping device, the processor asserts $\overline{\text{ARTRY}}$ 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 shown in Figure 3-6, $\overline{\text{ARTRY}}$ is asserted for one bus clock cycle, three-stated for half of the next bus clock cycle, driven high till the following bus cycle, and finally three-stated. Section 2.5.2, “Address Retry (ARTRY)—Output,” describes $\overline{\text{ARTRY}}$ timing for different processors.

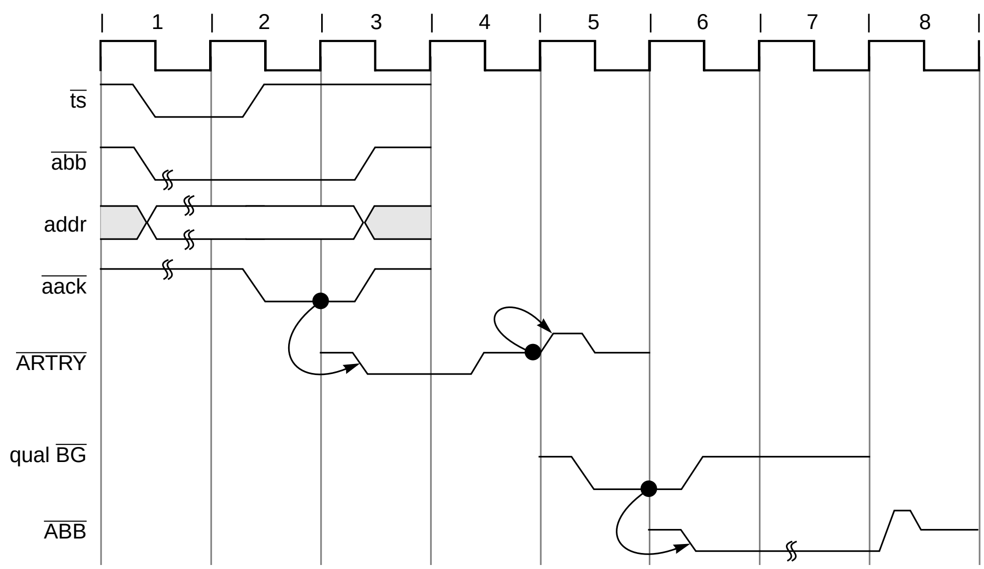

Figure 3-6. Snooped Address Cycle with $\overline{\text{ARTRY}}$

The snoop push window occurs two cycles after the assertion of $\overline{\text{AACK}}$. Coherency protocol provides that only one device can get a snoop hit due to modified data for any given address tenure. If $\overline{\text{ARTRY}}$ is asserted during the cycle after the assertion of $\overline{\text{AACK}}$, then in the following cycle, no processor asserts $\overline{\text{BR}}$ unless a snoop hit requires it to do a push. To guarantee that a snoop push gets an immediate opportunity to obtain the address bus, the external arbiter must grant the bus to the snooping device next.

A processor with a snoop hit that requires a push uses the window to request the address bus. After it gains the address bus, it uses the address and data tenures only to perform a push. In some cases, a processor may have a queued snoop push and receive a snoop hit that requires another push. The processor can use the window to perform the queued push and not queue the second push. If the processor is parked in the cycle after AACK, a processor with a snoop does not generate a fast push; instead, it acts as if it were not parked.

The SHD signal can also be asserted either coincident with ARTRY or alone to indicate that another bus device has a copy of the requested data and that the requesting device should mark its corresponding cache block as shared (S).

3.3 Data Bus Tenure

This section describes the data bus arbitration, transfer, and termination phases, which are nearly identical to address tenure phases.

3.3.1 Data Bus Arbitration

Data bus arbitration uses the data arbitration signal group—DBG, DBWO, and DBB. Additionally, the combination of TS and TT[0–4] provides information about the data bus request to external logic. Asserting TS is an implied data bus request; the arbiter must qualify TS with the transfer type (TT[0–4]) encodings to determine if the current address transfer is an address-only operation (see Table 2-1). If the data bus is needed, the arbiter grants data bus mastership by asserting DBG to the processor. As with the address bus arbitration phase, the processor must qualify DBG before assuming bus mastership, as described in Section 2.6.1, “Data Bus Grant (DBG)—Input.” As shown in Figure 3-7, the processor asserts DBB on the bus clock cycle after recognition of a qualified data bus grant.

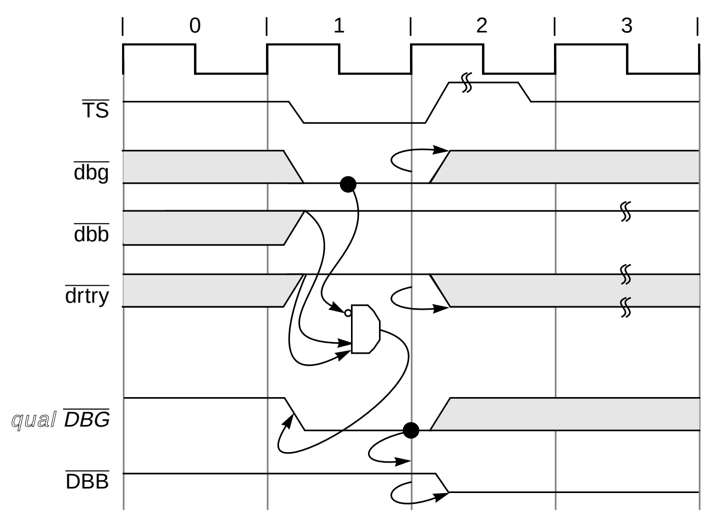

Figure 3-7. Data Bus Arbitration

When a data tenure overlaps its associated address tenure, a qualified $\overline{\text{ARTRY}}$ assertion coincident with a $\overline{\text{DBG}}$ signal does not result in data bus mastership ($\overline{\text{DBB}}$ is not asserted). Because the processor can pipeline outstanding data tenures when a new address tenure is retried, the processor becomes data bus master to complete the previous transaction.

3.3.1.1 Effect of $\overline{\text{ARTRY}}$ Assertion on Data Transfer and Arbitration on the PowerPC 604 Processor

The system designer must define the beginning of the window in which the snoop response is valid and ensure that data is not transferred until one cycle before that window, or until the same cycle as the beginning of that window in fast-L2 mode. The processors support a snoop response window as early as two cycles after assertion of $\overline{\text{TS}}$. In fast-L2 mode, data cannot be transferred earlier than the first cycle of the assertion of $\overline{\text{ARTRY}}$.

Asserting $\overline{\text{ARTRY}}$ can invalidate a previous or current data transfer and terminate the data cycle, invalidate a qualified data bus grant, or cancel a future data transfer. The possible scenarios are described as follows:

- If data is transferred (via assertion of $\overline{\text{TA}}$) two or more cycles before the beginning of the snoop window in the normal mode, or one or more cycles before the beginning of the snoop window in data streaming mode, then data is transferred too early to be cancelled by $\overline{\text{ARTRY}}$. Therefore, systems in which $\overline{\text{ARTRY}}$ can be asserted must not attempt data transfers (assert $\overline{\text{TA}}$) before this cycle.
- If data is transferred in the cycle before the beginning of the snoop response window, asserting $\overline{\text{ARTRY}}$ invalidates the data transfer in a similar fashion to assertion of $\overline{\text{DRTRY}}$ except that the data tenure is aborted rather than extended. If data streaming mode is active, data cannot be transferred in this cycle.
- If data is transferred in the first cycle of the snoop response window, asserting $\overline{\text{ARTRY}}$ invalidates the data transfer. This is like deasserting $\overline{\text{TA}}$ except that the data tenure is aborted instead of continued.
- If $\overline{\text{DBG}}$ has not been asserted, asserting $\overline{\text{ARTRY}}$ effectively negates the implied data bus request associated with the address transfer, and the processor does not expect a transfer. The system must not assert $\overline{\text{DBG}}$ for this transfer if any other processor data transfers are pending.
- If $\overline{\text{ARTRY}}$ is asserted during a data transfer, it is terminated after the first cycle of $\overline{\text{ARTRY}}$ assertion. Therefore, a burst transfer can be cut short.
- Asserting $\overline{\text{ARTRY}}$ in the same cycle as its corresponding $\overline{\text{DBG}}$ disqualifies the data bus grant in that cycle so the 604 cannot start a data transaction on the following cycle regardless of whether other data transactions are queued. However, on the cycle after the $\overline{\text{ARTRY}}$ assertion, the 604 responds to a qualified data bus grant if it has queued data transactions. Figure 3-8 shows a write address tenure that receives an $\overline{\text{ARTRY}}$ snoop response in the same cycle the system asserts $\overline{\text{DBWO}}$ and $\overline{\text{DBG}}$ (cycle 6) to grant the write data tenure before a previously-requested read data tenure. Following the $\overline{\text{ARTRY}}$ assertion, the qualified $\overline{\text{DBG}}$ assertion to the processor in cycle 7 is accepted for the read data tenure.

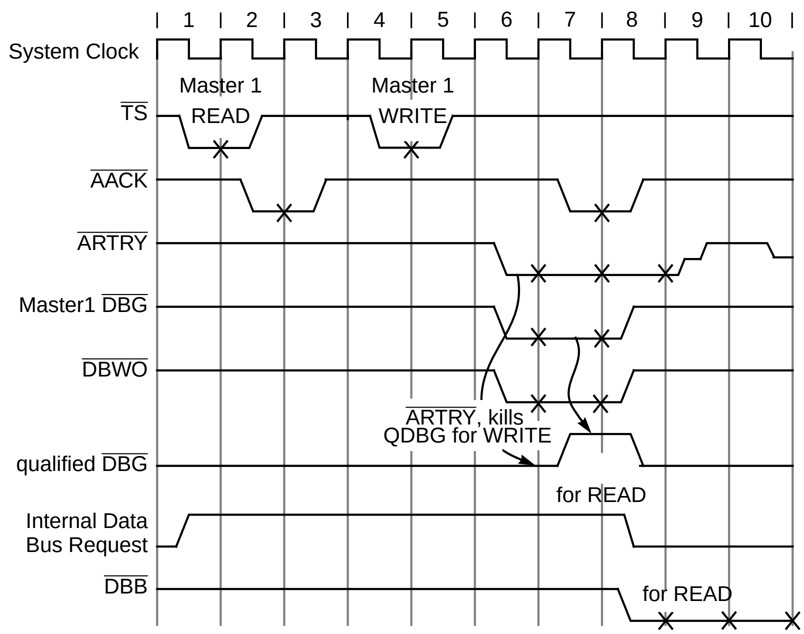

Figure 3-8. Qualified $\overline{\text{DBG}}$ Generation Following $\overline{\text{ARTRY}}$

3.3.1.2 Using the $\overline{\text{DBB}}$ Signal

The $\overline{\text{DBB}}$ signal should be connected between potential masters if data tenure scheduling is left to them. Optionally, the memory system can schedule data tenures directly with $\overline{\text{DBG}}$. However, the system can ignore $\overline{\text{DBB}}$ if it 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 asserted only to the next bus master on the cycle before the next bus master may actually begin its data tenure, rather than asserting it earlier (usually during another master's data tenure) and allowing $\overline{\text{DBB}}$ negation to be the final gating signal for a qualified data bus grant.

If the 604 is in data streaming mode, $\overline{\text{DBB}}$ is an output-only signal and is not sampled by the processor. Even if $\overline{\text{DBB}}$ is ignored in the system, the processor always recognizes its own assertion of $\overline{\text{DBB}}$ (except in data streaming mode) and requires one cycle after data tenure completion to negate its own $\overline{\text{DBB}}$ before recognizing a qualified data bus grant for the next data tenure.

If $\overline{\text{DBB}}$ is not required, it must be connected to a pull-up resistor on the processor to ensure proper operation. If the multiple 604s perform data streaming, each processor's $\overline{\text{DBB}}$ should be connected to the memory arbiter.

3.3.2 Data Bus Write Only

Because of address pipelining, a processor can queue up to three (two for the 601 and 603) data tenures to perform when it receives a qualified $\overline{\text{DBG}}$. Generally, data tenures should be performed in the order their address tenures were performed. However, the processor supports a limited out-of-order capability with the data bus write only ($\overline{\text{DBWO}}$) input. Using $\overline{\text{DBWO}}$ can avoid deadlocks that can occur in certain system designs. When recognized on the clock of a qualified $\overline{\text{DBG}}$, $\overline{\text{DBWO}}$ can direct the processor to perform the next pending data write tenure even if a pending read tenure normally would have been performed first. See Section 2.6.2, "Data Bus Write Only (DBWO)—Input."

The processor always accepts data bus mastership to perform a pending data tenure when it recognizes a qualified $\overline{\text{DBG}}$. If $\overline{\text{DBWO}}$ is asserted with a qualified $\overline{\text{DBG}}$ and no write tenure is queued, the 603 and 604 still take mastership of the data bus to perform the next pending read data tenure. If the processor has multiple queued writes, asserting $\overline{\text{DBWO}}$ reorders the write operation whose address was sent first.

Generally, $\overline{\text{DBWO}}$ should be used only to allow a copy-back operation (burst write) to occur before a pending read operation. If $\overline{\text{DBWO}}$ is used for single-beat write operations, it may negate the effect of the **eieio** instruction by allowing a write operation to precede a program-scheduled read operation.

3.3.3 Data Transfer

Data transfer signals include DH[0–31], DL[0–31], DP[0–7], and $\overline{\text{DPE}}$. The DH and DL signals form a 64-bit data path for read and write operations. The processor transfers data in either single- or four-beat burst transfers (eight-beat when the 603 is in 32-bit bus mode). Single-beat operations transfer from one to eight bytes within a double word at a time and can be misaligned; see Section 3.2.2.4, "Effect of Alignment in Data Transfers." Burst operations always transfer eight words and are aligned on eight-word address boundaries. Burst transfers give significantly higher bus throughput than single-beat transfers.

The type of transaction initiated by the processor depends on whether the code or data is cacheable and, for store operations, whether the memory accessed is marked write-back or write-through mode. Software controls this mode on 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 processor output TBST indicates to the system whether the current transaction is a single-beat or a burst transfer (except during **eciwx**/**ecowx** transactions, when it signals the state of EAR[28]). A burst transfer has an assumed address order. For load or store operations that miss the cache and are marked cacheable (stores are also marked as write-back) in the MMU, the processor uses the double-word–aligned (quad-word–aligned for the 601) address associated with the critical code or data that initiated the transaction.

This minimizes latency by allowing critical data to be forwarded to the processor before the rest of the cache block is filled. For all other burst operations, however, cache block transfers start with the oct-word–aligned data.

Bus masters including these processors may generate byte-wise odd parity for their outgoing data and drive this information onto the DP[0–7] lines coincidentally with their data. The processors check this parity whenever they read data and assert the $\overline{\text{DPE}}$ signal and take a machine check or checkstop exception if an error is detected. Parity checking can be disabled within the processors by setting a bit in the HID register.

The processors do not directly support dynamic memory access interfacing to subsystems with less than a 64-bit data path. Other system components must provide any required translation to devices with less than 64-bit data paths. The 601 provides limited data mirroring for noncachable transfers of less than a word.

3.3.4 Data Transfer Termination

Data bus transactions can be terminated by one of the four signals, $\overline{\text{TA}}$, $\overline{\text{DRTRY}}$, $\overline{\text{TEA}}$, or $\overline{\text{ARTRY}}$, which are described as follows:

- Asserting $\overline{\text{TA}}$ indicates normal termination of data transactions. It must be asserted on the bus cycle coincident with the data it qualifies. The slave can withhold $\overline{\text{TA}}$ for any number of clock cycles until valid data is ready to be supplied or accepted.
- $\overline{\text{DRTRY}}$ indicates invalid read data in the previous bus clock cycle. $\overline{\text{DRTRY}}$ extends the current data beat and does not terminate it. If it is asserted after the last (or only) data beat, the processor negates $\overline{\text{DBB}}$ but still considers the data beat active and waits for another assertion of $\overline{\text{TA}}$. $\overline{\text{DRTRY}}$ is ignored on write operations.

  Upon receiving a final (or only) termination condition, the processor negates $\overline{\text{DBB}}$ for one cycle, except when data streaming is used. If $\overline{\text{DRTRY}}$ is asserted to extend the last (or only) data beat past the negation of $\overline{\text{DBB}}$, the memory system should three-state the data bus on the clock after the final assertion of $\overline{\text{TA}}$, even though it negates $\overline{\text{DRTRY}}$ on that clock. This prevents a momentary data bus conflict if a write access begins on the following cycle.
- Asserting $\overline{\text{TEA}}$ signals a nonrecoverable error during a data transfer. It is recognized at any time during assertion of $\overline{\text{DBB}}$ or when a valid $\overline{\text{DRTRY}}$ could be sampled. Asserting $\overline{\text{TEA}}$ ends the data tenure immediately even if it is in the middle of a burst; however, it does not prevent incorrect data that has been acknowledged with $\overline{\text{TA}}$ from being written into the processor's cache or GPRs. Asserting $\overline{\text{TEA}}$ causes either a machine check exception or a checkstop condition depending on the MSR setting.
- Asserting $\overline{\text{ARTRY}}$ for the address tenure associated with the current data tenure ends the data tenure immediately. It may not be due to address pipelining. If $\overline{\text{ARTRY}}$ is connected for the processor, the earliest allowable assertion of $\overline{\text{TA}}$ to the processor depends directly on the earliest possible assertion of $\overline{\text{ARTRY}}$ to the processor; see Section 3.3.1.1, "Effect of ARTRY Assertion on Data Transfer and Arbitration on the PowerPC 604 Processor."

3.3.4.1 Normal Single-Beat Termination

Single-beat data read operations normally end when $\overline{\text{TA}}$ is asserted by a responding slave. Figure 3-9 shows that $\overline{\text{TEA}}$ and $\overline{\text{DRTRY}}$ must remain negated during the transfer.

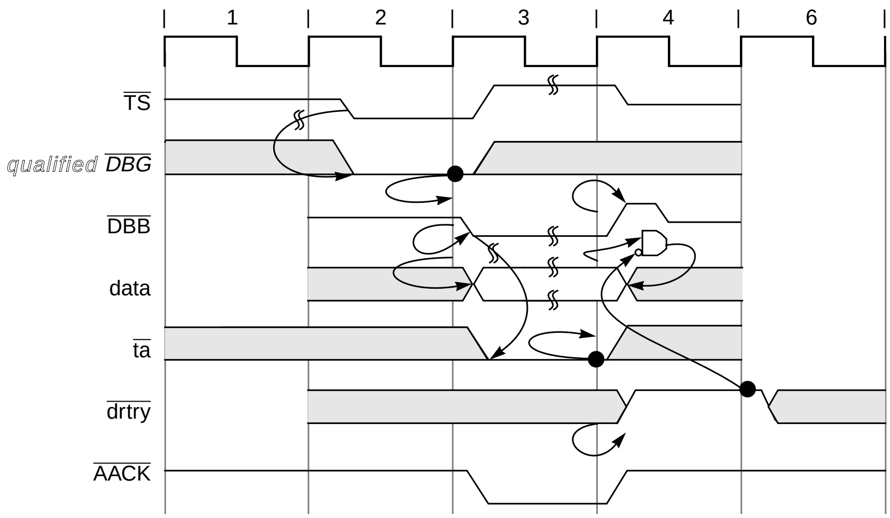

Figure 3-9. Normal Single-Beat Read Termination

Normal termination of a single-beat data write transaction occurs when $\overline{\text{TA}}$ is asserted by a responding slave. $\overline{\text{TEA}}$ must remain negated during the transfer. As shown in Figure 3-10, $\overline{\text{DRTRY}}$ is not sampled during data writes.

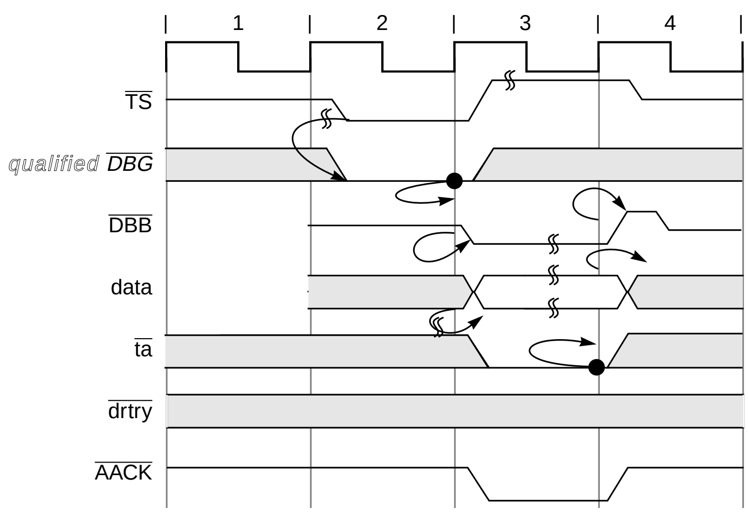

Figure 3-10. Normal Single-Beat Write Termination

Normal burst transfer termination occurs when $\overline{\text{TA}}$ is asserted for four bus clock cycles, shown in Figure 3-11. To pace data transfer beats, clock cycles in which $\overline{\text{TA}}$ is asserted need not be consecutive. To terminate read bursts, $\overline{\text{TEA}}$ and $\overline{\text{DRTRY}}$ must remain negated during the transfer. For successful write bursts, $\overline{\text{DRTRY}}$ is ignored and $\overline{\text{TEA}}$ must remain negated.

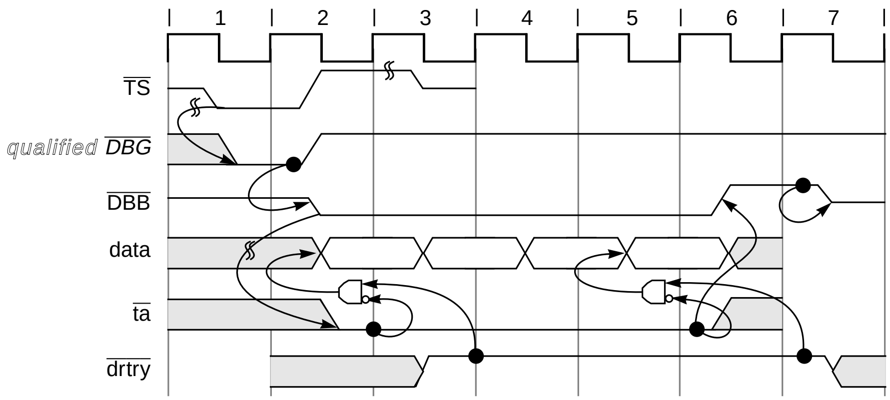

Figure 3-11. Normal Burst Transaction

For read bursts, $\overline{\text{DRTRY}}$ may be asserted one bus clock cycle after $\overline{\text{TA}}$ is asserted to signal that the associated data is invalid. It stays asserted until the cycle after valid data is sent by the slave (see Figure 3-12).

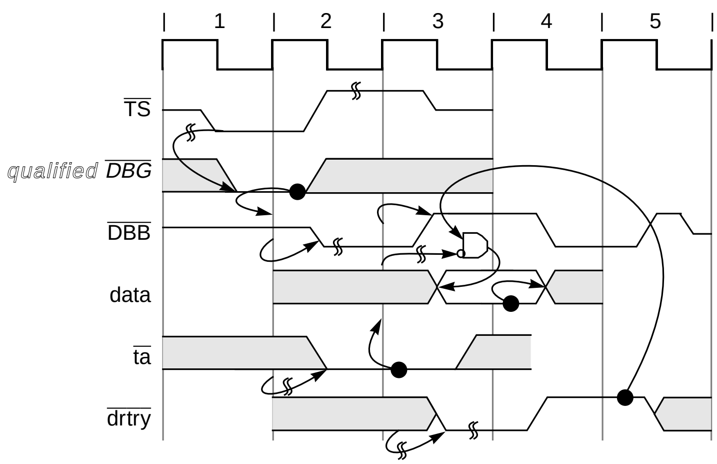

Figure 3-12. Termination with $\overline{\text{DRTRY}}$

Thus, a data beat can be terminated speculatively with TA and confirmed one bus clock cycle later by negating DRTRY (valid only for read transactions). TA must be asserted on the clock cycle before the first bus clock cycle of the assertion of DRTRY; otherwise results are undefined. Asserting DRTRY extends data bus mastership such that no other processors can use the data bus until DRTRY is negated. Therefore, in Figure 3-12, DBB cannot be asserted until clock cycle 5. This is true for both read and write operations, although DRTRY is ignored by the processors for write operations.

Figure 3-13 shows the effect of using $\overline{\text{DRTRY}}$ during a burst read. It also shows the effect of using $\overline{\text{TA}}$ to pace the data transfer rate; in clock cycle 3, $\overline{\text{TA}}$ is negated for the second data beat. The processor data pipeline proceeds in clock cycle 4 when $\overline{\text{TA}}$ is reasserted.

Note that $\overline{\text{DRTRY}}$ is useful for systems that implement speculative data forwarding (for example, those with direct-mapped, second-level caches where hit/miss is determined on the following bus clock cycle) or for parity- or ECC-checked memory systems. Figure 3-13 shows the data transferred in cycle 5 invalidated by the assertion of $\overline{\text{DRTRY}}$ in cycle 6. Its negation in cycle 7 and 8 and the assertion of $\overline{\text{TA}}$ indicates valid data beats.

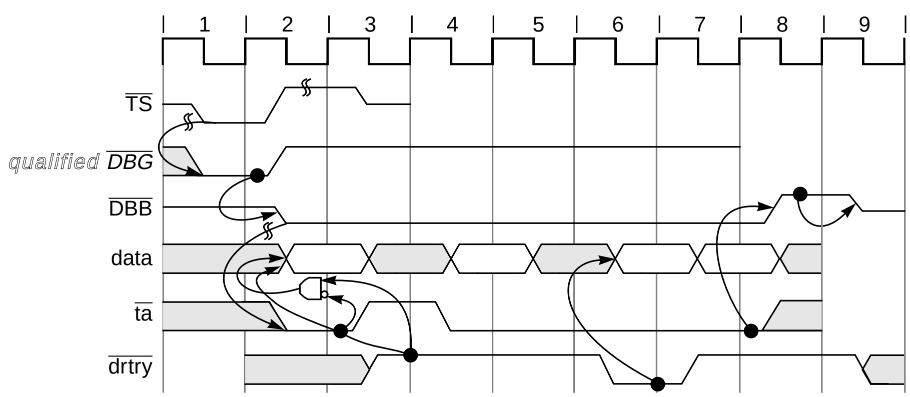

Figure 3-13. Read Burst with $\overline{\text{TA}}$ Wait States and $\overline{\text{DRTRY}}$

3.3.4.2 Data Transfer Termination Due to a Bus Error

To indicate that a bus error occurred, $\overline{\text{TEA}}$ can be asserted while $\overline{\text{DBB}}$ is asserted or when a valid $\overline{\text{DRTRY}}$ could be recognized by the processor. Asserting $\overline{\text{TEA}}$ to the processor terminates the transaction; that is, further assertions of $\overline{\text{TA}}$ and $\overline{\text{DRTRY}}$ are ignored and $\overline{\text{DBB}}$ is negated. If the system asserts $\overline{\text{TEA}}$ for a data transaction on the same cycle or before $\overline{\text{ARTRY}}$ is asserted for the corresponding address transaction, the processor ignores the effects of $\overline{\text{ARTRY}}$ on the address transaction and considers it successfully completed.

From a bus standpoint, asserting TEA causes nothing worse than the early termination of the data tenure in progress. All the system logic involved in processing the data transfer prior to the TEA must return to the normal nonbusy state following the TEA so that the bus operations associated with a machine check exception can proceed. Due to bus pipelining in the 604, all outstanding bus operations, including queued requests, complete in normal fashion following the assertion of TEA. The machine check exception can be taken while these transactions are in progress.

Asserting $\overline{\text{TEA}}$ causes a machine check exception (and possibly a checkstop condition within the processor). See Section 5.3.1, “Checkstop State (MSR[ME] = 0).” Because these processors do not implement a synchronous error capability for memory accesses, the exception instruction pointer points not to the memory access that caused the assertion of $\overline{\text{TEA}}$ but to the instruction about to be executed (perhaps several instructions later). However, assertion of $\overline{\text{TEA}}$ does not invalidate data entering the GPR or the cache. Additionally, the corresponding address of the access that caused $\overline{\text{TEA}}$ to be asserted is not latched by the processor. To recover, the exception handler must either identify and correct the error that caused $\overline{\text{TEA}}$ to be asserted or the processor must be reset; therefore, this function should be used only to flag fatal system conditions to the processor (such as parity or uncorrectable ECC errors).

After the processor has committed to run a transaction, that transaction must eventually complete. Address retry causes the transaction to be restarted. Although, $\overline{\text{TA}}$ wait states and $\overline{\text{DRTRY}}$ assertion for reads delay termination of individual data beats, eventually the system must either terminate the transaction or assert $\overline{\text{TEA}}$ (to generate a machine check exception). Therefore, software must check for the end of physical memory and the location of certain system facilities to avoid memory accesses that might cause $\overline{\text{TEA}}$ to be asserted.

If MSR[ME] is clear when $\overline{\text{TEA}}$ is asserted, a true checkstop condition occurs (instruction execution halted and processor clock stopped); a machine check exception occurs if MSR[ME] is set.

3.4 Timing Examples

This section shows timing diagrams for various scenarios. Figure 3-14 illustrates the fastest single-beat reads possible for these processors, showing both minimal latency and maximum single-beat throughput. By delaying the data bus tenure, latency increases, but, because of split-transaction pipelining, the overall throughput is not affected unless the data bus latency causes the fourth (third for 601 and 603) address tenure to be delayed.

Note that all bidirectional signals are three-stated between bus tenures.

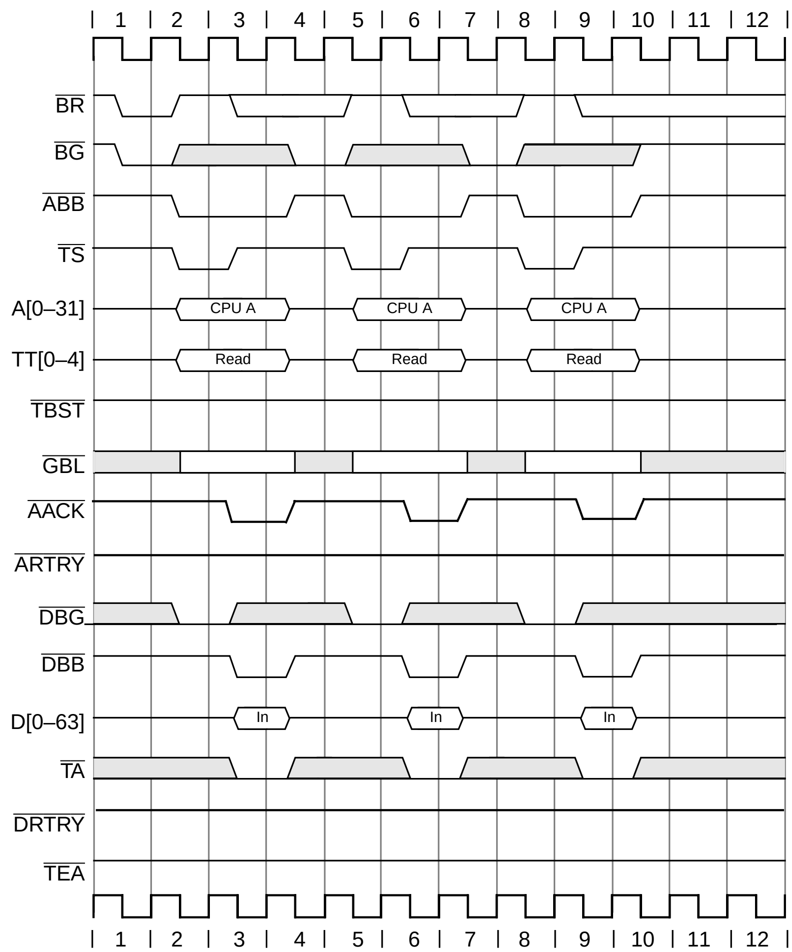

Figure 3-14. Fastest Single-Beat Reads

Figure 3-15 shows the fastest single-beat writes supported by these processors. The TT[1–4] signals are binary encoded 0bx0010 (TT0 can be either 0 or 1).

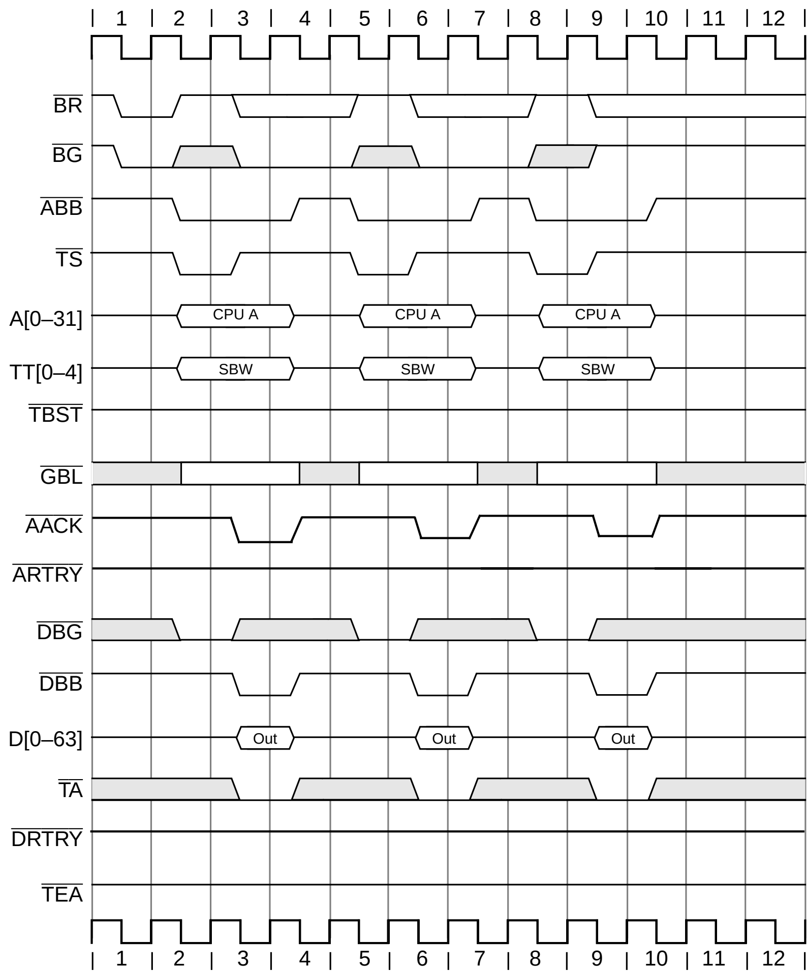

Figure 3-15. Fastest Single-Beat Writes

Figure 3-16 shows three ways to delay single-beat reads showing 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 that all bidirectional signals are three-stated between bus tenures.

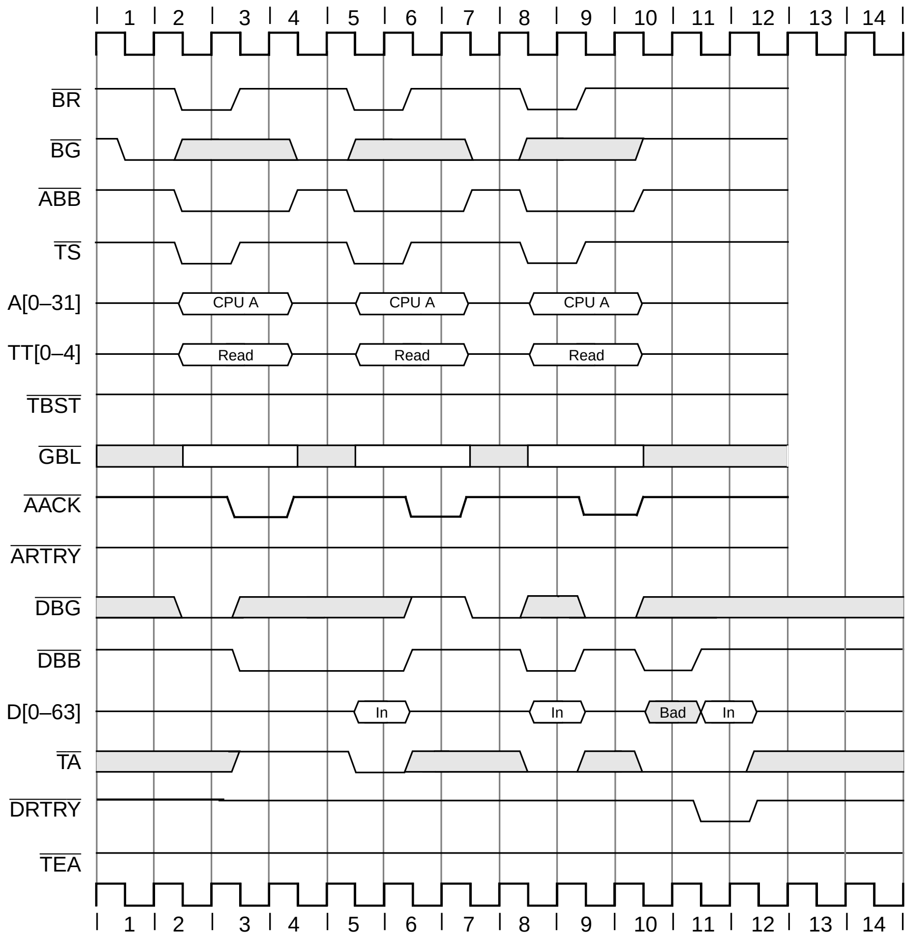

Figure 3-16. Single-Beat Reads Showing Data-Delay Controls

Figure 3-17 shows data-delay controls in a single-beat write operation. Bidirectional signals are three-stated between bus tenures. Data transfers are delayed in the following ways:

- $\overline{\text{TA}}$ 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).

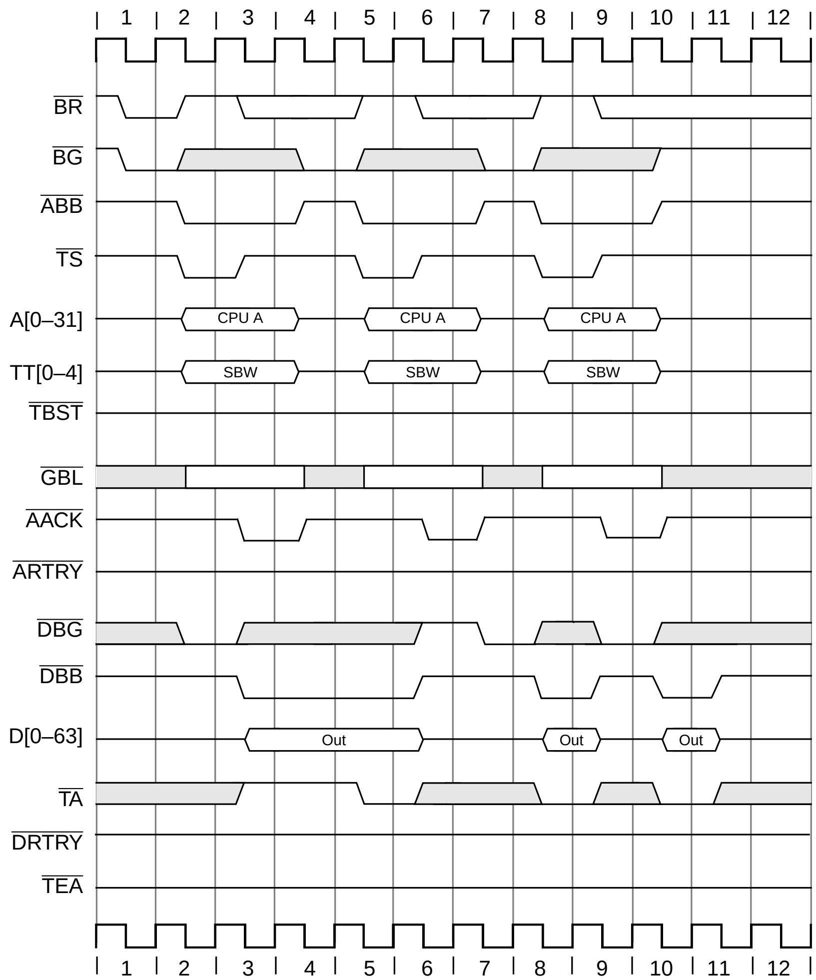

Figure 3-17. Single-Beat Writes Showing Data Delay Controls

Figure 3-18 shows the use of data-delay controls with burst transfers. All bidirectional signals are three-stated between bus tenures. Note the following:

- The first data beat of bursted read data (clock 3) is the critical double word (quad word for 601).
- 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.

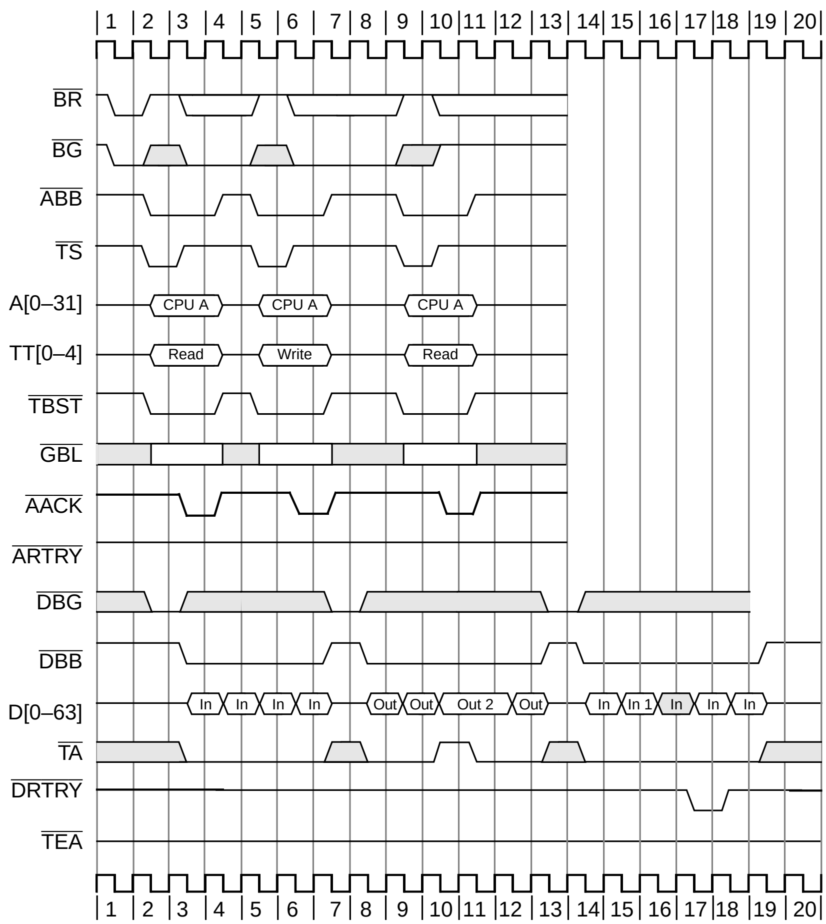

Figure 3-18. Burst Transfers with Data Delay Controls

Figure 3-19 shows the use of the $\overline{TEA}$ signal. Note that all bidirectional signals are three-stated between bus tenures. Note the following:

- The first data beat of the read burst (in clock 3) is the critical double word (quad word for 601).
- The $\overline{TEA}$ signal cancels the burst write transfer on the third data beat.
- The processor eventually causes an exception to be taken on the $\overline{TEA}$ event.

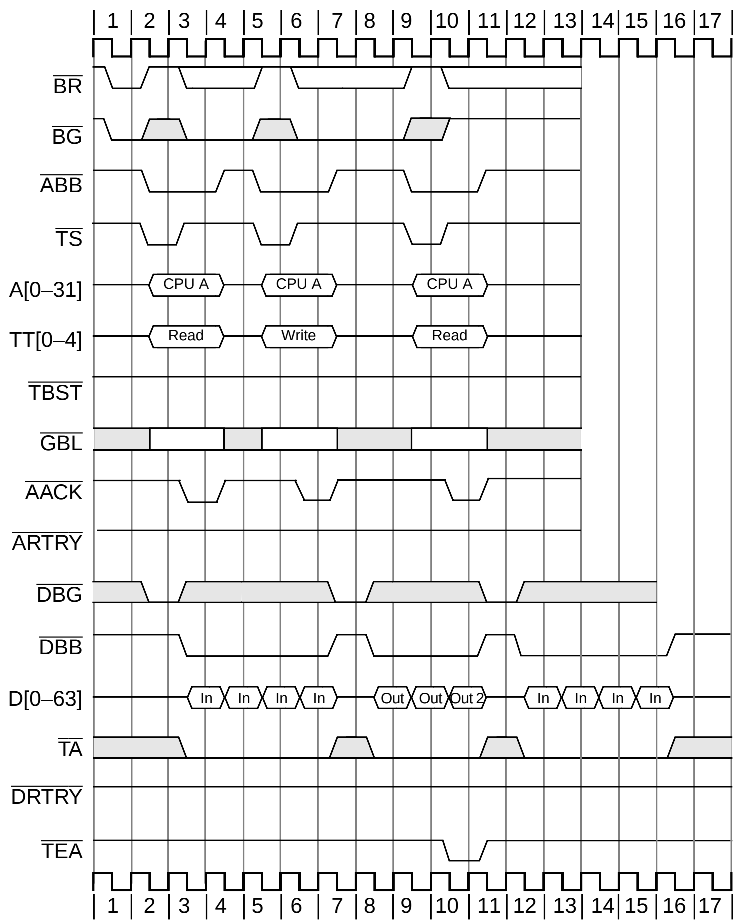

Figure 3-19. Use of Transfer Error Acknowledge ($\overline{TEA}$)

Chapter 4
Memory Coherency

This chapter describes hardware resources defined by the 60x bus definition that maintain memory coherency such that all devices that share memory in a system using a PowerPC processor have an accurate view of memory. Although the PowerPC architecture memory model requires memory to be kept coherent, it does not define either the snooping protocol or the use of MESI coherency states commonly used on PowerPC processors. The 60x processors provide resources that support memory coherency by snooping bus transactions. This chapter provides an overview of how the 60x processors implement the MESI protocol and the bus operations implemented by the 60x processors that ensure cache coherency.

Note that there are unique characteristics to the cache implementations of each of the PowerPC processors, which are summarized in the following sections.

4.1 Overview of Cache Implementations

To support a wide variety of processor implementations, the cache model defined by the PowerPC architecture is very flexible. Although it supports Harvard architecture caches, that is separate instruction and data caches, this is not required. Processor caches can vary greatly with respect to size, organization, and set-associativity, However, these considerations do not affect the bus design in a substantial way. For example, the number of sets in a processor's cache implementation determines the number of cache set element (CSE*n*) signals that must be implemented.

Major areas where a processor's cache structure affects the bus design are L2 cache support and the level of support given to multiprocessing concerns such as snooping, coherency-related bus operations, and MESI state logic. For example, the PowerPC 603 processor is not optimized for use in multiprocessor systems and therefore does not support the $\overline{\text{SHD}}$ bus signal or the shared (S) MESI state. Differences in how processors implement coherency-related bus operations are described in Section 4.10, "Overview of Implementation Differences."

The following section provides an overview of the cache implementations in the PowerPC 601, 603, and 604 processors.

4.1.1 PowerPC 601 Processor Cache Organization

The 601 implements a single unified cache that is configured as eight sets of 64 lines, each consisting of two sectors, four state bits (two per sector), an address tag, and several bits to maintain the LRU function. The two state bits implement the four-state MESI (modified-exclusive-shared-invalid) protocol. Each sector contains eight 32-bit words. Note that PowerPC architecture defines the cacheable unit as a block, which is a sector in the 601.

To maintain the flow of instructions through the instruction queue, the instruction unit accesses the cache frequently. The queue is eight words (one sector) long, so an entire sector can be loaded into the instruction unit on a single clock cycle.

The cache organization is shown in Figure 4-1. Replacement strictly follows an LRU algorithm; that is, the least-recently used sector is used, which may mean that a modified sector is replaced on a miss if it is the least-recently used, even if invalid sectors are available. However, for performance reasons, certain conditions (for example, the execution of some cache instructions) generate accesses to the cache without modifying the bits that perform the LRU function.

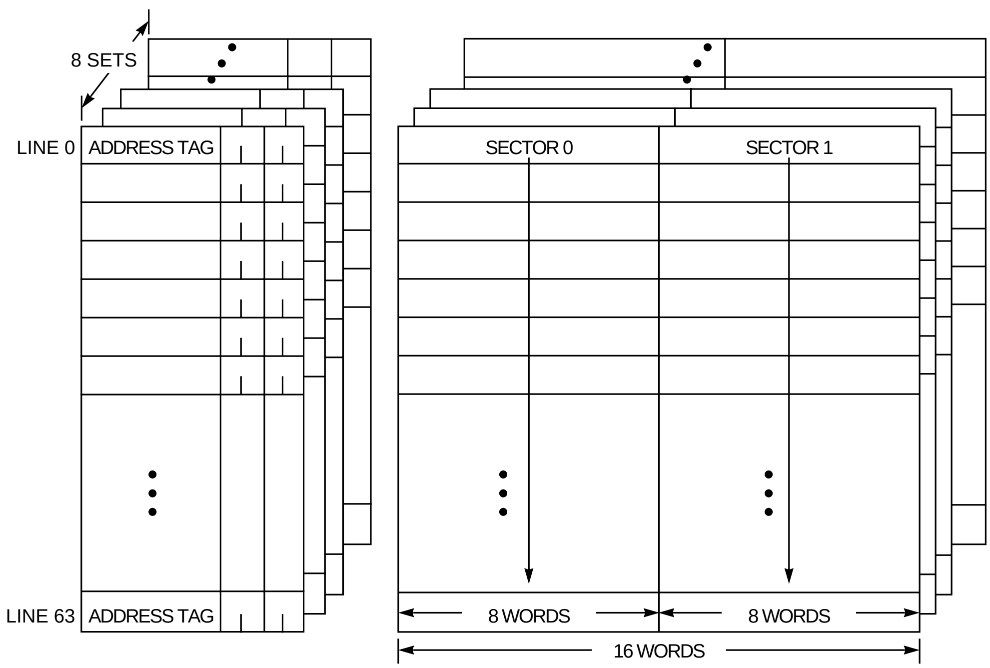

Figure 4-1. PowerPC 601 Processor Cache Organization

Each cache block contains 16 contiguous words from memory that are loaded from a 16-word boundary (that is, bits A26–A31 of the logical (effective) addresses are zero); as a result, cache lines are aligned with page boundaries.

Note that address bits A20–A25 provide an index to select a line. Bits A26–A31 select a byte within a line. The tags consists of bits PA0–PA19. Address translation occurs in parallel, such that higher-order bits (the tag bits in the cache) are physical.

4.1.2 PowerPC 603 Processor Cache Organization

The 603 has separate instruction and data caches. The organization of the 603 data and instruction caches is shown in Figure 4-2.

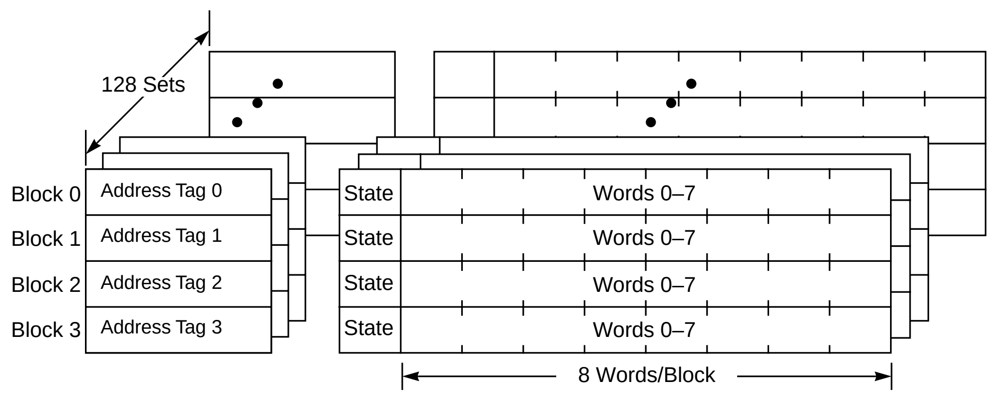

Figure 4-2. PowerPC 603 Processor Cache Organization

Each cache block has eight contiguous words from memory that are loaded from an eight-word boundary, that is, bits A27–A31 of the logical (effective) addresses are zero. As a result, cache blocks are aligned with page boundaries.

Address bits A20–A26 provide an index to select a set. Bits A27–A31 select a byte within a block. The tags consist of bits PA0–PA19. Address translation occurs in parallel, such that higher-order bits (the tag bits in the cache) are physical. Replacement strictly follows an LRU algorithm; that is, the least-recently used block is updated on a cache miss.

The 603 instruction cache, is like that of the data cache, although bits are not provided to maintain MEI cache coherency.

4.1.3 PowerPC 603e Processor Cache Enhancements

The 603e provides the following enhancements to the 603 cache implementation:

- The instruction cache is blocked only until the critical load completes (hit under reloads allowed).
- The critical double word is simultaneously written to the cache and forwarded to the requesting unit, thus minimizing stalls due to load delays.

- Provides for an optional data cache operation broadcast feature (enabled by the HID0[ABE] bit) that allows for correct system management using an external copy-back L2 cache.
- Optional broadcast of cache control instructions **dcbi**, **dcbf**, and **dcbst** through configuration of HID0[ABE] bit.

4.1.4 PowerPC 604 Processor Cache Organization

The 604 cache implementation consists of separate 16-Kbyte instruction and data caches (Harvard architecture). The 604 instruction and data cache organization is shown in Figure 4-3.

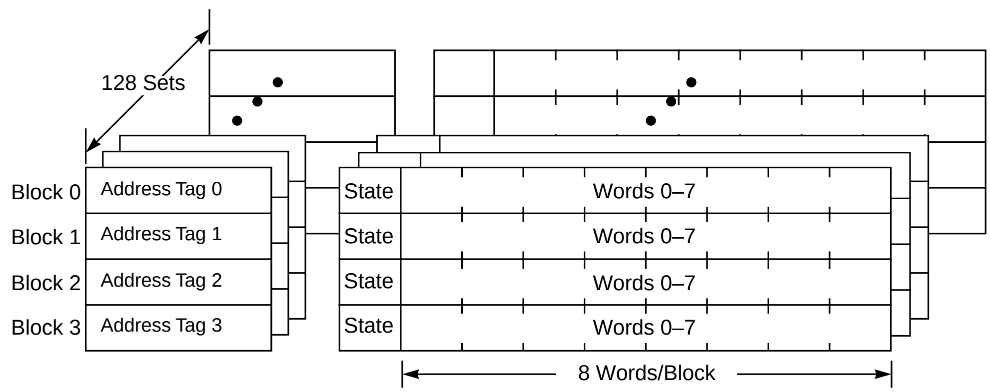

Figure 4-3. PowerPC 604 Processor Cache Organization

Both caches are four-way set associative and implement an LRU replacement algorithm within each set. The cache directories are physically addressed with the physical (real) address tag stored in a cache directory.

Both the instruction and data caches have 32-byte cache blocks. The coherency state bits for each block of the data cache allow encoding for all four possible MESI states. The coherency state bit for each cache block of the instruction cache allows encoding for two possible states:

— Invalid (INV)
— Valid (VAL)

Each cache can be invalidated or locked by setting appropriate bits in the hardware implementation dependent register 0 (HID0).

The 604 uses eight-word burst transactions to transfer cache blocks to and from memory. When requesting burst reads, the 604 presents a double-word–aligned address. Memory controllers are expected to transfer this double word of data first, followed by double words from increasing addresses, wrapping back to the beginning of the eight-word block as required. Burst misses can be buffered into two eight-word line-fill buffers before being loaded into the cache. Cache block writes for copy-back operations always present the first address of the block and transfer data beginning at the start of the block. However, this does not keep other masters from transferring critical double words first on the bus for writes.

4.1.5 PowerPC 604e Processor Cache Enhancements

The 604e has separate 32-Kbyte data and instruction caches. This is double the size of the 604 caches. The 604e caches are logically organized as a four-way set with 256 sets compared to the 604’s 128 sets. The physical address bits that determine the set are 19 through 26 with 19 being the most-significant bit of the index. If bit 19 is zero, the block of data is an even 4-Kbyte page that resides in sets 0–127; otherwise, bit 19 is one and the block of data is an odd 4-Kbyte page that resides in sets 128–255. Because the caches are four-way set-associative, the cache set element (CSE[0–1]) signals remain unchanged from the 604. Figure 4-4 shows the organization of the 604e caches.

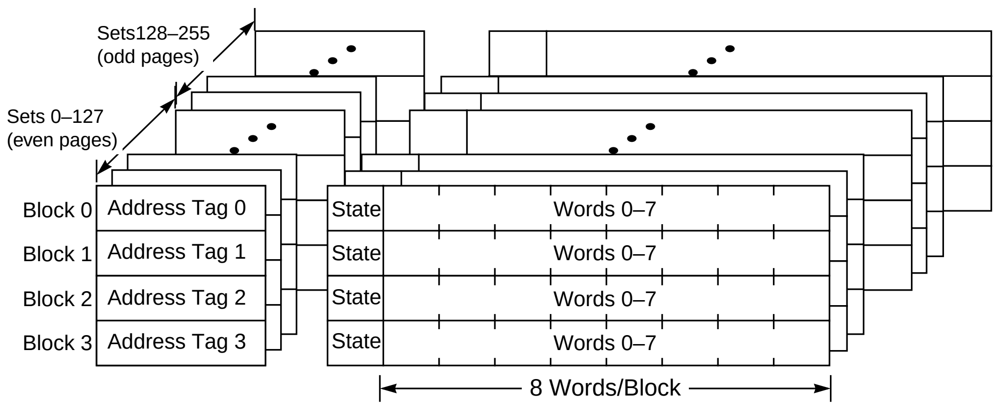

Figure 4-4. PowerPC 604e Processor Cache Organization

4.2 Cache Coherency Overview

A coherent memory system provides the same image of memory to all devices that share a system’s memory. This is important for multiprocessor systems because it allows for synchronization, task migration, and the cooperative use of shared resources. An incoherent memory system could easily produce unreliable results depending on when and which processor executed a task. Maintaining coherency is a concern primarily for data cache implementations. For example, if a processor does not have exclusive access to an addressed block before performing a store operation, another processor could have a copy of the old (or stale) data. Two processors reading from the same memory location would get different data.

To maintain a coherent memory system, each processor follows simple rules for managing the cache state such as broadcasting its intention to read a cache block not in the cache and its intention to write into a block not owned exclusively. Other devices respond by snooping the broadcast addresses and reporting cache status back to the originating processor.

The status returned includes a shared indicator (the $\overline{\text{SHD}}$ signal) and an address retry indicator (the $\overline{\text{ARTRY}}$ signal). The snooping processor asserts $\overline{\text{SHD}}$ if it has a copy of the addressed block; it asserts $\overline{\text{ARTRY}}$ if it has a modified copy of the addressed cache block that must be written back to memory or if another processor had a problem that kept it from snooping the address. For additional information about snooping, see Section 4.7.1, "General Comments on 60x Snooping."

To maximize performance, the 601 and 604 provide a second path into the data cache directory for snooping that allows the mainstream instruction processing to operate concurrently with snooping. Instruction processing is affected only when snoop-control logic requires a snoop push of modified data to maintain memory coherency.

4.3 Memory Coherency—MESI Protocol

Each cache block is in one of the four MESI states. Addresses presented to the cache are indexed into the cache directory and are compared against the cache directory tags. If no tags match, the result is a cache miss. If a tag match occurs, a cache hit has occurred and the directory indicates the state of the block through three state bits kept with the tag.

The four possible states for a cache block are invalid (I), shared (S), exclusive (E), and modified (M), which are defined in Table 4-1.

Table 4-1. MESI State Definitions

MESI State	Definition
Modified (M)	The addressed block is valid in the cache and in only this cache. The block is modified with respect to system memory—that is, the modified data in the block has not been written back to memory. Note that some documentation identifies this as XM (exclusive modified) state.
Exclusive (E)	The addressed block is in this cache only. The data in this block is consistent with system memory. Note that some documentation identifies this as XU (exclusive unmodified) state.
Shared (S)	The addressed block is valid in the cache and in at least one other cache. This block is always consistent with system memory. That is, the shared state is shared-unmodified; there is no shared-modified state. The 603, which is not optimized for multiprocessor implementations, does not support the shared (S) state.
Invalid (I)	This state indicates that the addressed block is not resident in the cache and/or any data contained is considered not useful.

Figure 4-5 illustrates the basic relationships of the MESI states.

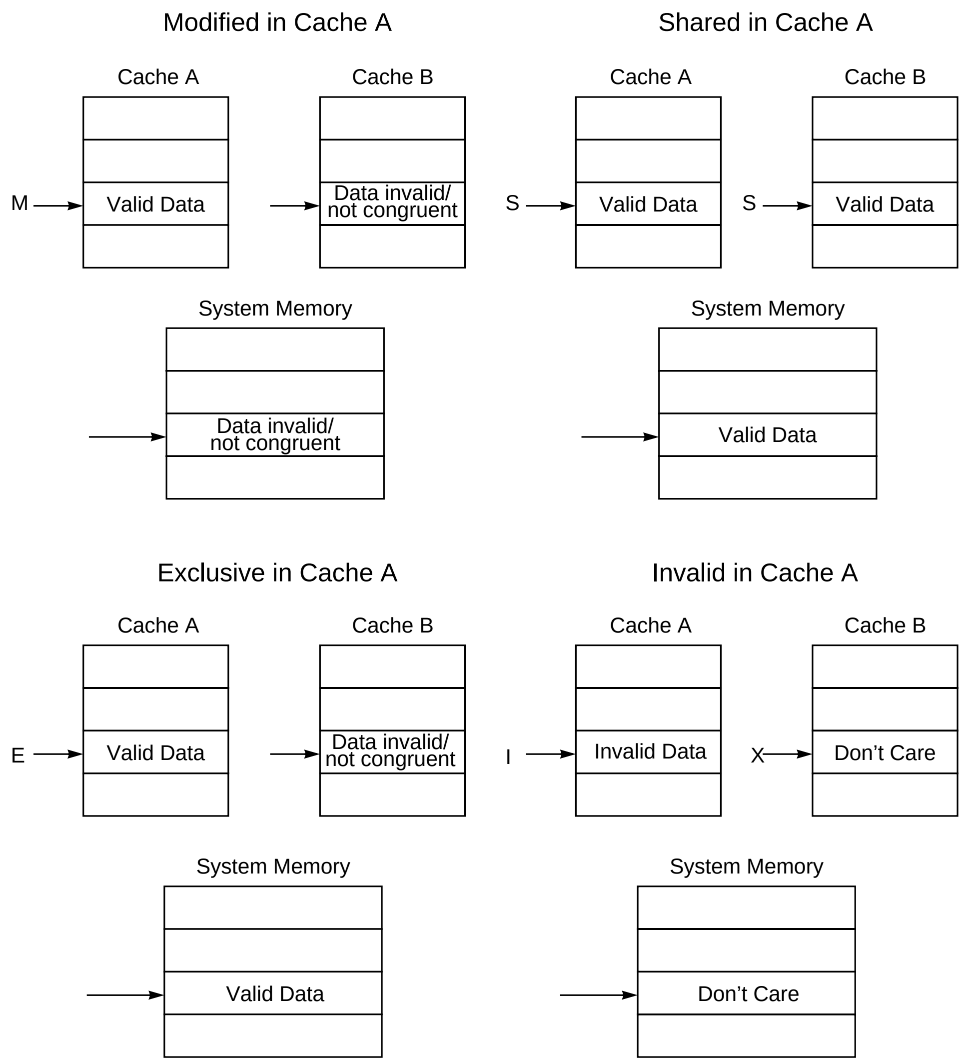

Figure 4-5. MESI States

Although memory space designated for instructions is rarely updated, data in memory space designated for data is continually being changed as the results from instruction execution are stored in memory. Therefore, maintaining coherency in data caches requires greater hardware support.

The 604 and 601 have dedicated hardware to provide memory coherency by snooping bus transactions. The address retry capability enforces the four-state, MESI cache coherency protocol (see Figure 4-6).

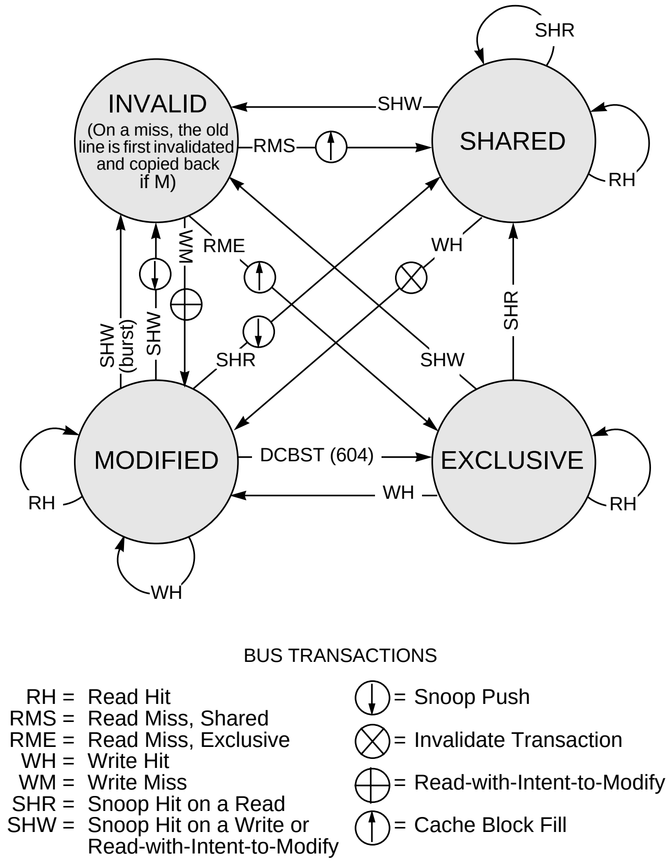

Figure 4-6. MESI Cache Coherency Protocol (601/604)—State Diagram (WIM = 001)

The global ($\overline{\text{GBL}}$) output signal indicates whether the current transaction must be snooped by other devices. 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) and should be snooped. If $\overline{\text{GBL}}$ is not asserted for the transaction, that transaction is not snooped.

Normally, $\overline{\text{GBL}}$ reflects the M-bit value specified for the memory reference in the corresponding translation descriptor(s). 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 a processor is not the address bus master, $\overline{\text{GBL}}$ is an input. The 604 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 604 cache occurs if the snooped transaction is not marked global. This includes invalidation cycles.

When the processor detects a qualified snoop condition, the address associated with the $\overline{\text{TS}}$ is compared against the data cache tags through a dedicated cache tag port. Snooping completes if no hit is detected. If, however, the address hits in the cache, the processor reacts according to the MESI protocol shown in Figure 4-6, assuming the WIM bits are set to write-back mode, caching allowed, and coherency enforced (WIM = 001).

Write hits to modified cache blocks of nonglobal pages do not generate invalidate broadcasts. Several bus transactions involve moving data that can no longer access the TLB M bit (for example, replacement cache block copy-back or a snoop push). In these cases, because hardware cannot determine whether the cache block was originally marked global, the processor marks these transactions as nonglobal to avoid retry deadlocks.

See Table 4-2 for the CSE[0–1] encodings for the 604.

4.4 Coherency Timing

60x processors communicate the results of their snooping over the snoop response lines, $\overline{\text{ARTRY}}$ and $\overline{\text{SHD}}$. These signals are defined to be valid at least the cycle after assertion of $\overline{\text{AACK}}$. A 60x that tries to acquire a memory block is considered to have acquired it after it has successfully completed the address tenure requesting the data (no-$\overline{\text{ARTRY}}$ indication). After that cycle, it snoops for that address. Likewise, a 60x processor that is flushing data from its cache is considered to have completed the transfer from the standpoint of memory coherency after it has successfully completed the address tenure for the push or copy-back. Once this has occurred, it no longer snoops for this address.

Note that this has implications to system design. For example, after a 60x pushes a cache block, it may be some time before the block is actually stored in memory. If a read of the same block occurs after the push address tenure is completed, it is not snooped by the 60x performing the push. The system must ensure that proper ordering is maintained so that the correct data is read.

4.5 Coherency Protocol

The 60x bus supports a four-state (MESI) cache coherency protocol through the use of address retry (see Figure 4-6). The 601 and 604 implement the protocol to the extent required to support multiprocessor systems. Because it does not support the shared state, the 603 supports a three-state subset of the MESI protocol, (MEI) protocol, which assures coherency in a a single-processor system. All references to the shared state do not apply to the 603.

When the 60x is not the bus master, it monitors the bus. If $\overline{\text{GBL}}$ is asserted, the 601/604 snoop address transfers. Due to the 603 **lwarx**/**stwcx.** implementation, which is based on the MEI protocol, the bus is snooped regardless of the state of $\overline{\text{GBL}}$. See Section 8.8, "lwarx/stwcx. Considerations," for more information.

The 604 snoops its own nonglobal or global transfers (internally, not across the bus) in the case of the address-only operations, ICBI, SYNC, TLBIE, and TLBSYNC, and can assert $\overline{\text{ARTRY}}$ in response.

Normally, $\overline{\text{GBL}}$ reflects the value of the M bit provided by the translation mechanism in the master processor. See Section 4.8, "External WIM Bit Settings," for details of the conditions under which $\overline{\text{GBL}}$ does not reflect the state of the M bit. Figure 4-6 shows the MESI protocol implemented by the 601 and 604; Figure 4-7 shows the MEI cache coherency protocol for the 603.

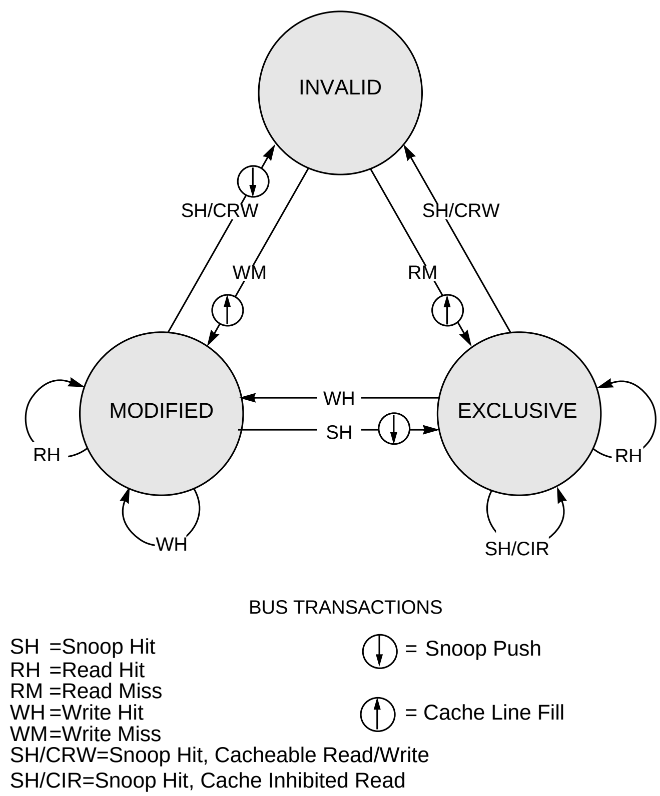

Figure 4-7. MEI Cache Coherency Protocol (603)—State Diagram (WIM = 001)

4.5.1 PowerPC 603 Processor lwarx/stwcx. Implementation

Due to the 603's three-state MEI protocol and absence of address-only broadcast transfers, the **lwarx/stwcx.** instruction pair is different from the 601 and 604. All global reads snooped by the 603 (except for a RWNITC) invalidate a cache block, and all reads originating from the 603 are RWITMs (except for reads from cache-inhibited pages). Therefore, a potential deadlock can occur if RWITMs cancel a reservation as in the 601 and 604. The following operations are required for the 603 **lwarx/stwcx.** implementation:

- RWITM invalidates the cache, but does not clear the reservation.
- Only writes on the bus can clear a reservation.
- **stwcx.** is treated as a write-through bus operation.

Clearing the reservation on all writes including castouts and snoop pushes (nonglobal) require snooping for all global and nonglobal address transfers for reservation address register monitoring; however, a snoop on a nonglobal address transfer does not change any cache states.

4.5.2 Cache Set Element Signals

The cache set element signals, CSE*n*, are output signals that indicate which set member of the cache is involved for cache block reads and writes. Note that because of the different sizes and structures of the caches, the number of signals required to identify a cache set vary from processor to processor. There are three cache set element signals on the 601 (CSE[0–2]), one on the 603 (CSE), and two on the 604 (CSE[0–1]).Table 4-2 defines these signals for a four-way set-associative cache, such as is implemented in the 604. For more information, see Section 2.4.12, "Cache Set Element (CSEn)—Output."

Table 4-2. CSE[0–1] Signals

CSE[0–1]	Cache Set Element
00	Block 0
01	Block 1
10	Block 2
11	Block 3

4.5.3 Address Retry Sources

Snooping devices use $\overline{\text{SHD}}$ and $\overline{\text{ARTRY}}$ to respond to snoop requests. Because these signals are wire-ORed among many potential snoopers that can have different snoop responses, little significance can be attached to the particular combination of the two bits that appears on the bus.

An assertion of $\overline{\text{ARTRY}}$, regardless of whether $\overline{\text{SHD}}$ is asserted, indicates either that at least one snooper had a pipeline collision or a snoop hit to a modified block and that the address must be retried. Assertion of $\overline{\text{SHD}}$ alone indicates that at least one snooper had a snoop hit on a shared cache block. The 603 does not implement $\overline{\text{SHD}}$.

4.6 Memory Coherency Actions—PowerPC 60x Processor-Initiated Operations

Table 4-3 roughly describes the behavior of the 60x with respect to cacheable load and store operations. All reads originating from the 603 except those where caching is inhibited are RWITMs. Also, all reads on the bus except RWNITC invalidate a cache block.

Table 4-3. Memory Coherency Actions on Load Operations

Cache State	Bus Operation	Snoop Response	Action
I	Read	¬$\overline{\text{ARTRY}}$, ¬$\overline{\text{SHD}}$	Load data and mark E
		¬$\overline{\text{ARTRY}}$, SHD	Load data and mark S
		$\overline{\text{ARTRY}}$	Retry read operation
M, E, S	None	Don’t care	Read from cache

Table 4-4 roughly describes the behavior of the 60x with respect to load and store operations to cacheable, write-back memory. Note that the state of the $\overline{\text{SHD}}$ signal is not important in this table.

Table 4-4. Memory Coherency Actions on Store Operations

Cache State	Bus Operation	Snoop Response	Action
I	RWITM	¬$\overline{\text{ARTRY}}$	Load data, modify it, mark M
		$\overline{\text{ARTRY}}$	Retry the RWITM
S	Kill	¬$\overline{\text{ARTRY}}$	Modify cache, mark M
		$\overline{\text{ARTRY}}$	Retry the kill
E	None	Don’t care	Modify cache, mark M
M	None	Don’t care	Modify cache

For detailed descriptions of these operations, see, Section E.1, “Load Operations,” and Section E.2, “Store Operations.”

4.6.1 Cache Control Instructions

Table 4-5 lists bus operations performed by the 601 and 604 when they execute cache control instructions.

Table 4-5. PowerPC 601 and 604 Processor Bus Operations Initiated by Cache Control Instructions

Instruction	Current Cache State	Next Cache State	Bus Operation	Comment
sync	Don't care	No change	SYNC	First clears memory queue
icbi	Don't care	I	604: ICBI; 601: Kill	—
dcbi	Don't care	I	Kill	—
dcbf	E, S, I	I	Flush	—
	M	I	Write w/ kill	Marked as WT
dcbst	E, S, I	No change	Clean	—
	M	E	Write w/kill	Marked as WT
dcbz	I	M	Kill	A write-back may be required
	S	M	Kill	—
	E, M	M	None	Write over modified data
dcbt, **dcbtst**	I	E, S	Read	State change on reload
	M, E, S	No change	None	—

Figure 4-6 shows the operations for the 603.

Table 4-6. PowerPC 603 Bus Operations Initiated by Cache Control Instructions

Instruction	Current Cache State	Next Cache State	Bus Operation	Comment
sync	Don't care	No change	None	First clears memory queue
dcbi, **icbi**	Don't care	I	None	—
dcbf	E, I	I	None	—
	M	I	Write w/killL	—
dcbst	E, I	No change	None	—
	M	E	Write w/killL	—
dcbz[1]	I	M	RWITM	Possible castout on cache miss
	E, M	M	RWITM	Write over modified data
dcbt	I	E	RWITM	State change on reload [2]
	E, M	No change	None	—

[1] RWITM on DCBZ serves both as a substitute for a DCBZ broadcast and as a mechanism to zero out the cache block (data from RWITM ignored).

[2] 603 has a touch load buffer. A cache reload is delayed until a hit in the touch load buffer occurs.

Table 4-5 and Table 4-6 give a general sense of the basic behavior of the processor. For example, it does not address noncacheable or write-through cases, nor does it completely describe the exact mechanisms for the operations described.For a complete listing of cache coherency operations, see Appendix E, “Coherency Action Tables.”

4.6.2 TLB Invalidate Entry Instruction Processing

Executing a **tlbie** instruction causes a processor to invalidate any TLB entry that corresponds to that instruction’s effective address. It also causes a TLBIE operation to be broadcast onto the bus (except on the 603).

4.6.2.1 TLBIE Bus Operation

The TLBIE bus operation is an address-only transaction. The address that is transmitted contains at least bits EA[12–19] in their correct bit positions. Processors that receive this transaction use the address to index into their TLB(s) and invalidate an entire congruence class. Any other device that implements its own TLB must process the TLBIE bus operation.

To avoid system deadlock conditions, devices that process TLBIE bus operations must start the operation only after the bus operation has been completed without an $\overline{\text{ARTRY}}$ response. Because participating devices take an unspecified amount of time to perform their invalidations, completion of the entire invalidation sequence is not guaranteed until completion of a synchronization operation, as described in Section 4.7, “Descriptions of Bus Transactions and Snoop Responses.” The 601 uses the **sync** instruction to synchronize TLBIE operations; the 604 uses **tlbsync**.

4.7 Descriptions of Bus Transactions and Snoop Responses

This is a summary of bus transactions and snoop responses. Causes and effects of these operations are given in Appendix E, “Coherency Action Tables.”

4.7.1 General Comments on 60x Snooping

When 60x processors are not bus master, they monitor bus traffic and perform cache and memory queue snooping as appropriate. Snooping is triggered by the receipt of a qualified snoop request, as indicated by the simultaneous assertion of the $\overline{\text{TS}}$ and $\overline{\text{GBL}}$.

Processors drive two snoop status signals, $\overline{\text{ARTRY}}$ and $\overline{\text{SHD}}$, in response to qualified snoop requests. These signals provide information about the state of the addressed block with respect to 60x for the current bus operation. These signals are described in more detail earlier in this document. The following additional comments apply:

- Any bus transaction that does not have $\overline{\text{GBL}}$ asserted can be ignored by all bus snoopers. Such transactions are ignored by 60x processors (except 603). For more information, refer to Chapter 8, “System Interface Operation,” in the *PowerPC 603e RISC Microprocessor User’s Manual*.

- Several bus transactions (write with flush, read, and read with intent to modify) are defined twice, once with TT0 clear and once with it set (for atomic operations). These operations behave in the same manner with respect to bus snooping.
- The receiving processor may assert $\overline{\text{ARTRY}}$ in response to any bus transaction due to internal conflicts that prevent the appropriate snooping.

4.7.2 Clean Block

Clean block is an address-only transaction a 60x processor issues after executing a **dcbst** instruction. If $\overline{\text{GBL}}$ is asserted, a clean block transaction causes 60x processors to respond as follows:

- If the addressed block is in the I, S, or E state, no further action is taken.
- If the addressed block is in the M state, the modified block is copied back to memory and the state of the block is changed to E.

The 603 does not broadcast or snoop clean block operations.

4.7.3 Flush Block

Flush block is an address-only transaction that a processor issues after executing a **dcbf** instruction. If $\overline{\text{GBL}}$ is asserted, a flush block transaction causes 60x processors to respond as follows:

- If the addressed block is in the S or E state, the state of the addressed block is changed to I.
- If the addressed block is in the M state, the snooping device asserts $\overline{\text{ARTRY}}$ and $\overline{\text{SHD}}$, the modified block is pushed out of the cache, and its state is changed to I.

The 603 does not broadcast or snoop flush block operations.

4.7.4 Write with Flush, Write with Flush Atomic

Write with flush and write with flush atomic are issued by a processor after executing stores or **stwcx.** respectively to memory in a variety of different states, particularly noncacheable and write-through. 60x processors do not use this transaction code for burst transfers, but system use for bursts is not precluded. If they appear on the bus and the $\overline{\text{GBL}}$ signal is asserted, the 60x processors have the same snoop response as for flush block, except that a hit on the reservation address causes loss of the reservation.

4.7.5 Kill Block

A kill block is an address-only transaction that a processor generates by executing a **dcbi** instruction (or an **icbi** instruction in a 601), a **dcbz** to an I or S line, or a write to an S line. If $\overline{\text{GBL}}$ is asserted when a transaction appears on the bus, an addressed block in the cache is forced to the I state.

The 603 does not broadcast or snoop kill operations.

4.7.6 Write with Kill

A processor typically issues a write-with-kill operation whenever it performs a cache block write back. 60x processors use this transaction code for burst transfers. If they appear on the bus and the $\overline{\text{GBL}}$ signal is asserted, the 60x processors have the same snoop response as for kill block.

4.7.7 Read, Read Atomic

Read is used by most single-beat or burst bus operations. If $\overline{\text{GBL}}$ is asserted, 60x processors respond to read operations as follows:

- If the addressed block is present and in the I state, the 60x takes no action.
- If the addressed block is present and in the S state, the 60x asserts $\overline{\text{SHD}}$.
- If the addressed block is present and in the E state, the 60x asserts $\overline{\text{SHD}}$ and changes the cache state from E to S.
- If the addressed block is present in the cache in the M state, the 60x asserts both $\overline{\text{ARTRY}}$ and $\overline{\text{SHD}}$. In addition, it changes the state of that cache block from M to S.

Read atomic operations appear in response to an **lwarx** instruction and receive the same snooping treatment as a read operation.

4.7.8 Read with Intent to Modify (RWITM)

The RWITM transaction is issued to acquire exclusive use of a memory location, for the purpose of modifying it. One example is a processor that writes to a block that is not currently in its cache. RWITM transactions on the bus, when $\overline{\text{GBL}}$ is asserted, cause 60x processors respond as follows:

- If the addressed block is not present in the cache, the 60x takes no action.
- If the addressed block is present in the cache in the S or E state, the 60x changes the state of that cache block to I.
- If the addressed block is present in the cache in the M state, the 60x asserts both $\overline{\text{ARTRY}}$ and $\overline{\text{SHD}}$, pushes the modified block out of the cache, and changes the state of that cache block from M to I.

RWITM atomic appears on the bus in response to an **stwcx.** instruction and receives the same snooping treatment as RWITM.

4.7.9 TLB Invalidate

TLB invalidate is issued by a processor that executes a **tlbie** instruction. This operation sends at least certain bits of an effective address (EA) across the bus. Receiving processors invalidate the entire congruence class in any TLBs associated with that effective address.

The address transmitted with the **tlbie** instruction contains EA[12–19] in their correct respective bit positions (see Figure 4-8).

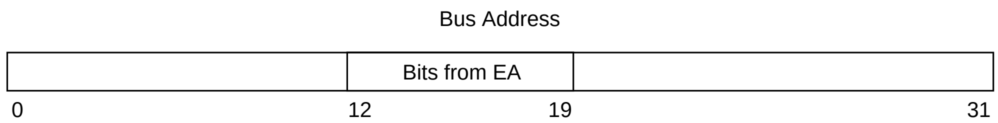

Figure 4-8. Effective Address Bits in Bus Address

When the TLBIE appears on the bus, attached 60x processors invalidate the congruence class of the TLB that corresponds to the transmitted bits of the effective address.

4.7.10 SYNC

SYNC is an address-only transaction that a 60x processor places onto the bus as the result of execution of a **sync** instruction. If a processor has other snooped cache operations pending when it detects a SYNC on the bus, it asserts $\overline{\text{ARTRY}}$. A 601 detecting a SYNC on the bus also asserts $\overline{\text{ARTRY}}$ for any pending operations based on an invalidated TLB.

The 603 does not broadcast or snoop SYNC.

4.7.11 TLBSYNC

TLBSYNC is an address-only transaction placed on the bus by execution of a **tlbsync** instruction or a pending TLBIE bus operation. A 604 seeing **tlbsync**, asserts $\overline{\text{ARTRY}}$ if any pending operations are based on an invalidated TLB.

The 603 does not broadcast or snoop TLBSYNC operations. The 601 does not implement the **tlbsync** instruction and does not generate this bus operation.

4.7.12 EIEIO

The EIEIO bus operation is generated by executing an **eieio** instruction, which acts as a fence in the instruction flow to enforce ordered execution of accesses to noncacheable memory. The 60x processors internally enforce ordering of such accesses with respect to the **eieio**, in the sense that noncacheable accesses due to instructions that occur before the **eieio** in the program order are placed on the bus before any noncacheable accesses that result from instructions that occur after the **eieio**, with the EIEIO bus operation separating the two sets of bus operations.

If the system implements any mechanism that allows reordering of noncacheable requests, then the appearance of an EIEIO should cause it to force ordering between accesses that occurred before and those that occur later.

The 603 does not broadcast or snoop EIEIO operations.

4.7.13 ICBI

This operation is issued by a processor that executes an instruction cache block invalidate (**icbi**) instruction. All copies of the addressed block in bus-attached instruction caches are invalidated.

The 603 does not broadcast or snoop ICBI operations.

The **icbi** causes the 601 to broadcast a kill operation to the bus.

4.7.14 Read with No Intent to Cache (RWNITC)

Read with no intent to cache (RWNITC) operations are issued by a bus-attached device as TT[0–4] = 0b01011 (like a read, but with TT4 = 1). The 603 and 604 snoop this and, if they get a cache hit on a block marked M, push the block and mark it E (the ordinary response would be to push and mark it S in 604 and push and invalidate for 603).

For a graphics adapter that reads display data from memory, this data may be in the processor's cache and the subject of frequent updates. Because the adapter does not cache the data, there is no reason for the processor to leave the block in the S state, requiring a bus operation to regain E access. Because the 603 has no S state, it must also reread the data.

4.7.15 XFERDATA

XFERDATA read and write bus transactions result from execution of the **eciwx** or **ecowx** instructions, respectively. These instructions help certain adapter types (especially displays) make high-speed data transfers with memory by calculating an effective address, translating it, and presenting the resulting physical address to the adapter.

The XFERDATA read and write transfer a word of data to or from the processor, respectively. They also present the 4-bit resource ID (RID) field, which is stored in the processor's transfer control register (TCR) to the bus, using the concatenation of the bits $\overline{\text{TBST}}$ || TSIZ[0–2]. These transactions are unique in the sense that the address that is transferred does not select the slave device; it is simply being passed to the slave device for use in a subsequent transaction. Rather, the RID field is used to select the appropriate slave device. Although it is intended that the slave device selected by the RID bits use the address transferred in a subsequent data transfer, the exact nature of this data transfer is not defined by 60x bus architecture. It is a private transfer that can be defined by the system like any other direct-memory access.

4.8 External WIM Bit Settings

The write-through ($\overline{\text{WT}}$), cache-inhibit ($\overline{\text{CI}}$), and global ($\overline{\text{GBL}}$) signals generally correspond to the W, I, and M bits supplied by the translation mechanism (page table or BAT); however, there are exceptions:

- In real mode, load and store operations bypass the translation mechanism and are implicitly WIM = 001 or WIM = 011 if the cache is disabled or locked.
- Write-back and snoop push operations do not involve the translation mechanism and are sent out as WIM = 000.
- TLB reloads are placed onto the bus as WIM = 001 or WIM = 011 if the cache is disabled or locked.
- The **dcbst** and **dcbf** instructions to non–write-through memory are placed on the bus indicating write-through (write-with-kill bus operation) if the cache block is in the M state.
- The XFERDATA bus operations are always placed on the bus with WIM = 010, regardless of the state of the WIM bits supplied by the translation mechanism.
- For SYNC, TLBSYNC, TLBIE, and EIEIO, WIM = xx1, where x is not defined.
- For ICBI, WIM is as provided by the translation mechanism.

4.9 Direct-Memory Access and Memory Coherency

When system devices perform direct-memory accesses, they may choose to assert or negate $\overline{\text{GBL}}$. 60x processors never snoop a request for which this bit is negated. It is therefore a system design decision whether or not a device that accesses memory should work from memory that is guaranteed to be coherent or not. The trade-off is that snooped accesses, while convenient, generally reduce system performance.

Another option available to system designers is to define different burst length transfers, using the reserved code points defined in the TSIZ[0–2] field. 60x processors snoop regardless of the state of the TSIZ signals, provided $\overline{\text{GBL}}$ is asserted. Note that coherency cannot be maintained if the system defines a transfer that crosses a cache block boundary.

4.10 Overview of Implementation Differences

Table 4-7 summarizes the basic differences in how the various PowerPC processors implement the bus operations defined in this chapter. This is a brief overview of those differences and do not describe the more subtle differences in the logic that is used to ensure cache coherency which are described in Appendix E, “Coherency Action Tables.”

Table 4-7. Differences in Implementation of Bus Operations

Bus Operation	Differences
Clean block	The 603 does not broadcast or snoop clean block operations.
Flush block	The 603 does not broadcast or snoop flush block or implement $\overline{\text{SHD}}$.
Write with flush, Write with flush atomic	In the 601, HID0[31] controls whether $\overline{\text{HP\_SNP\_REQ}}$ specifies high-priority operations.
Kill block	The 603 does not broadcast or snoop kill.
Write with kill	—
Read, read atomic	The 603 does not implement $\overline{\text{SHD}}$ and S state.
Read with Intent to modify (RWITM)	—
TLB invalidate	A snooping 604 also asserts $\overline{\text{ARTRY}}$ when it has a pending TLB invalidate operation and a second TLB invalidate operation is detected. The 601 uses the **sync** instruction to synchronize TLBIE operations; the 603 and 604 use **tlbsync**.
SYNC	The PowerPC architecture permits data accesses from more than one instruction to be combined for cache-inhibited operations, except when the accesses are separated by a **sync** instruction, or by an **eieio** instruction when the page or block is also designated as guarded. This combined access capability is not implemented on the 603e. The 603 does not broadcast or snoop SYNC. A SYNC operation is also generated by the **eieio** instruction on the 601.
TLBSYNC	A 604 seeing **tlbsync**, asserts $\overline{\text{ARTRY}}$ if any pending operations are based on an invalidated TLB. The 603 does not broadcast or snoop **tlbsync**. The 601 does not implement the **tlbsync** instruction and does not generate this bus operation.
EIEIO	The 603 does not broadcast or snoop EIEIO. The **eieio** is treated as a no-op by the 603e.
ICBI	The 603 does not broadcast or snoop ICBI. The **icbi** causes the 601 to broadcast a kill operation to the bus.
Read with no intent to cache (RWNITC)	Read with no intent to cache (RWNITC) operations are issued by a bus-attached device as TT[0–4] = 0b01011. The 603 and 604 snoop this and, if they get a cache hit on a block marked M, push the block and mark it E (the ordinary response would be to push and mark it S in 604 and push and invalidate for 603). For a graphics adapter that reads display data from memory, this data may be in the processor's cache and the subject of frequent updates. Because it has no S state, the 603 handles some operations differently.
XFERDATA	—
I/O reply	The I/O reply operation serves as the final bus operation in the series of bus operations that service direct-store interface operation. The 603e processors do not perform these operations because they do not implement the direct-store facility.

Chapter 5
System Status Signals

This chapter further describes the operation of the system status signals (interrupt, checkstop, and reset signals) which are described in Section 2.9, “System Status Signals.” Most of these are input signals that are used to generate asynchronous exceptions either as a function of normal system operations or as the result of an error. This chapter also briefly discusses asynchronous exceptions described in *The Programming Environments Manual*, with particular attention given to differences in how 60x processors implement those exceptions.

5.1 Overview

The PowerPC 601, 603, and 604 processors implement asynchronous exceptions that are triggered by signals. Asynchronous exceptions can be either maskable or nonmaskable.

Table 5-1 lists the signal-triggered operations implemented in the 60x processors. (The only other architecture-defined asynchronous exception, the decrementer exception, is triggered internally.) The table shows the event priority and indicates whether a related exception is maskable and precise.

Table 5-1. Resets, Interrupts, and Their Sources

Resets/ Interrupts	Maskable/ Nonmaskable	Precise/ Imprecise	Priority	Source		
				601	603	604
Hard reset	Nonmaskable	Imprecise	Highest priority	$\overline{\text{HRESET}}$	$\overline{\text{HRESET}}$	$\overline{\text{HRESET}}$
Machine check	Nonmaskable	Imprecise	Second-highest priority	$\overline{\text{TEA}}$	$\overline{\text{TEA}}$, $\overline{\text{MCP}}$, $\overline{\text{APE}}$, $\overline{\text{DPE}}$	$\overline{\text{TEA}}$, $\overline{\text{MCP}}$, $\overline{\text{APE}}$, $\overline{\text{DPE}}$
Soft reset	Nonmaskable	Imprecise	Third-highest priority	$\overline{\text{SRESET}}$	$\overline{\text{SRESET}}$	$\overline{\text{SRESET}}$
System management[*]	Maskable	Precise	Lower priority than synchronous exceptions; higher than external interrupt.	—	$\overline{\text{SMI}}$	$\overline{\text{SMI}}$
External interrupt	Maskable	Precise	Lower priority than a system management exception; higher than decrementer exception.	$\overline{\text{INT}}$	$\overline{\text{INT}}$	$\overline{\text{INT}}$

[*] The system management exception is not defined by the PowerPC architecture, but is implemented similarly in several PowerPC processors.

Table 5-2 describes general differences in how the 601, 603, and 604 implement the system status signals.

Table 5-2. Processor Bus Signal Differences

Signal(s)	Related Exception	Difference
Interrupt (INT)	External interrupt (0x00500)	For the 601, this signal may be negated after the minimum pulse width of three processor clock cycles.
System management interrupt (SMI)	System management interrupt (0x01400)	The system management interrupt exception is not defined by the PowerPC architecture and not implemented on the 601.
Machine check (MCP)	Machine check (0x00200)	This signal is not defined for the 601.
Checkstop input (CKSTP_IN)	Machine check (0x00200)	Early versions of the 603 identified this signal as CKSTP.
Checkstop output (CKSTP_OUT)	Machine check (0x00200)	Early versions of the 603 identified this signal as CHECKSTOP.
Hard reset (HRESET)	System reset (0x00100)	After assertion, output drivers are released to high impedance within five SYSCLK pulses (three for the 601) after the assertion of HRESET.
Soft reset (SRESET)	System reset (0x00100)	Negation may occur any time after the minimum soft reset pulse width of 2 (10 for the 601) bus cycles has been met.

5.2 Resets

There are two types of resets:

- Hard resets—These occur with the proper assertion of the HRESET signal, as part of a system's power-on reset, or due to other system-dependent occurrences. After a hard reset occurs, registers and other resources are initialized and instruction fetching begins at the system reset exception vector at 0xFFF00100.
- Soft resets—These occur with the proper assertion of the SRESET signal. When the SRESET is detected, the machine state is saved in the SRR0 and SRR1 registers, the MSR is reset, and exception processing continues from the system reset exception handler which resides at vector offset 0x00100.

5.2.1 Hard Reset and Power-On Reset

The 60x processors are reset by asserting the $\overline{\text{HRESET}}$ input for a minimum period of time after V_{dd} is stable. On the 603 and 604, this includes time for the phase-locked loop to lock. Refer to the respective hardware specifications for the duration requirement.

While $\overline{\text{HRESET}}$ is asserted, all 60x outputs are placed in the high-impedance state and bus content has no relevance. This state may continue after the $\overline{\text{HRESET}}$ signal has been negated as the processor performs various internal initializations and tests. The system must not perform any activity based on interpretation of floating lines.

Most control lines require pull-up resistors to be negated because they are multidrop. The $\overline{\text{BR}}$ signal must also be pulled up so it is not recognized as asserted during processor initialization.

The following is also true when a hard reset occurs:

- 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).
- Because MSR[IP] is set by a hard reset, the first instruction is fetched from address 0xFFF0_0100.

5.2.1.1 Hard Reset Settings

Note that a hard reset operation should be performed on power-on to appropriately reset the processor. Table 5-3 shows the state of the machine just before it fetches the first instruction after a hard reset.

Table 5-3. Hard Reset Settings

Resource	601[1]	603	603e	604/604e [2]
BATs	All 0s	Unknown	Unknown	Undefined
Cache	All 0s	All cache blocks invalidated	All cache blocks invalidated	Undefined and disabled
CR	All 0s	All 0s	All 0s	Undefined
CTR	All 0s	All 0s	All 0s	Undefined
DABR	—	—	—	Breakpoint disabled; Address undefined.
DAR	All 0s	All 0s	All 0s	Undefined
DCMP/ICMP	—	All 0s	All 0s	—
DEC	All 0s	FFFF_FFFF	FFFF_FFFF	Undefined
DMISS/IMISS	—	All 0s	All 0s	—

Table 5-3. Hard Reset Settings (Continued)

Resource	601[1]	603	603e	604/604e [2]
DSISR	All 0s	All 0s	All 0s	Undefined
EAR	All 0s	All 0s	—	E cleared; RID undefined.
FPRs	All 0s	All 0s	Unknown	Undefined
FPSCR	All 0s	All 0s	All 0s	Cleared
GPRs	All 0s	All 0s	Unknown	Undefined
HASH1	—	—	All 0s	—
HASH2	—	—	All 0s	—
HID0	8001_0080	All 0s	All 0s	All 0s
HID1	All 0s	—	All 0s	—
HID2	See IABR	—	—	—
HID5	All 0s	—	—	—
HID15	All 0s	—	—	—
IABR	All 0s	All 0s	All 0s	Breakpoint is disabled. Address is undefined.
LR	All 0s	All 0s	All 0s	Undefined
MQ	All 0s	—	—	—
MSR	0000_1040	0000_0040	0000_0040	0000_0040 (only IP set)
PIR	—	—	—	Undefined
PVR	Version-dependent			
RPA	—	All 0s	All 0s	—
RTCL	All 0s	—	—	—
RTCU	All 0s	—	—	—
SDR1	All 0s	All 0s	All 0s	Undefined
SPRGs	All 0s	All 0s	All 0s	Undefined
SRR0	All 0s	All 0s	All 0s	Undefined
SRR1	All 0s	All 0s	All 0s	Undefined
SRs	All 0s	Unknown	Unknown	Undefined
Tag directory	All 0s. (However, LRU bits are initialized so each side of the cache has a unique LRU value.)			
TBL	—	All 0s	All 0s	Undefined

Table 5-3. Hard Reset Settings (Continued)

Resource	601[1]	603	603e	604/604e [2]
TBU	—	All 0s	All 0s	Undefined
TLBs	All 0s	Unknown	Unknown	Undefined
XER	All 0s	All 0s	All 0s	Undefined

[1] 601 notes: Master checkstop enabled; internal power-on reset checkstops enabled. Note that if external clock is connected to RTC for the 601, the RTCL, RTCU, and DEC registers can change from their initial value of 0s without receiving instructions to load those registers. All internal arrays and registers are cleared during the hard reset process.

[2] 604/604e notes: Both HRESET and TRST signals should be asserted during power up and must remain asserted according to the values provided in the *PowerPC 604 RISC Microprocessor Hardware Specifications*. The 604 internal state after the hard reset interval is defined below. If HRESET is asserted for less than this amount of time, results are not predictable. If HRESET is asserted during normal operation, all operations stop and the machine state is lost. The processor automatically begins operations by issuing an instruction fetch. Because caching is inhibited at start-up, this generates a single-beat load operation on the bus.

The following output signals are placed in high impedance during hard reset: ABB, TS, XATS, A[0–31], AP[0–3], TT[0–4], TSIZ[0–2], TBST, TC*n*, CI, WT, GBL, CSE*n*, ARTRY, SHD, DBB, DH[0–31], DL[0–31], and DP[0–7].

The following output signals are negated during hard reset: BR, APE, DPE, RSRV, and CHKSTP_OUT.

The 604e's bus interface can be configured into one of two modes during a hard reset, as described in Table 5-4.

Table 5-4. PowerPC 604e Processor Modes Configurable during HRESET

604e Mode	Input Signal	Timing Requirements	Notes
Normal bus mode	DRTRY	Must be negated throughout HRESET assertion. After HRESET negation, DRTRY can be used normally.	—
Fast-L2 mode	DRTRY	Must be asserted and negated coincidentally with HRESET and remain negated during normal operation.	Can be done by tying DRTRY to HRESET
No-DRTRY Mode (604 only)	DRTRY	Must be asserted coincidentally with HRESET and remain asserted during normal operation.	Can be done by tying DRTRY asserted.

5.2.2 Soft Reset

A soft reset is generated by the proper assertion of the SRESET signal. When the signal is recognized as asserted, the system reset exception is generated as described in the following section.

5.2.2.1 System Reset Exception (0x00100)

The system reset exception is defined by the PowerPC architecture (operating environment architecture, or OEA) as a nonmaskable, asynchronous exception signaled to the processor typically through the assertion of a system-defined signal.

Table 5-5 shows how the machine state is saved and the MSR settings after the system reset exception is invoked.

Table 5-5. 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 bits from the MSR (cleared in the 601) 1–4 Cleared 5–9 Loaded with equivalent bits from the MSR (cleared in the 601) 10–15 Cleared 16–29 Loaded with equivalent bits from the MSR 30 Loaded from the equivalent MSR bit, MSR[RI][1], if the exception is recoverable; otherwise cleared. 31 Loaded with equivalent bit from the MSR Note that depending on the implementation, reserved bits in the MSR may not be copied to SRR1. If the processor state is corrupted to the extent that execution cannot resume reliably, the bit corresponding to MSR[RI][1], (SRR1[30]), is cleared.
MSR	POW[1] 0 PR 0 SE 0 IR [4] 0 TGPR [2] 0 FP 0 BE 0 DR [5] 0 ILE[1] — ME — FE1 0 RI[1] 0 EE 0 FE0 0 IP [3] — LE [6] Set to value of ILE

[1] Not implemented on the 601
[2] 603e only
[3] Identified as EP on the 601
[4] Identified as IT on the 601
[5] Identified as DT on the 601
[6] Not implemented on the 601. Control of little-endian mode on the 601 is provided by HID0[28], the LM bit.

When a system reset exception is taken, instruction execution continues at offset 0x00100 from the physical base address indicated by MSR[IP].

If the exception is recoverable, the value of the MSR[RI] bit is copied to the corresponding SRR1 bit. The exception functions as a context synchronizing operation. The exception is not recoverable if a reset exception causes the loss of any of the following:

- An asynchronous precise exception (interrupt, system management, or decrementer)
- Direct-store error type DSI
- Floating-point enabled type program exception

If the SRR1 bit corresponding to MSR[RI] is cleared, the exception is context synchronizing only with respect to subsequent instructions. Note that each implementation provides a means for software to distinguish between power-on reset and other types of system resets (such as soft reset).

5.2.2.2 Soft Reset on the PowerPC 601 Microprocessor

Because the 601 does not implement the MSR[RI] bit, it does not support restarting the interrupted process; however, to perform diagnostic operations it attempts to save the processor state.

5.2.2.3 Soft Reset on the PowerPC 603 Microprocessor

When $\overline{\text{SRESET}}$ is asserted, the processor attempts to reach a recoverable state by allowing the next instruction to either complete or cause an exception, blocking the completion of subsequent instructions and allowing the completed store queue to drain. A soft reset is recoverable provided that attaining the recoverable state does not cause a machine check exception.

5.2.2.4 Soft Reset on the PowerPC 604 Microprocessor

Unlike hard reset, soft reset does not directly affect the states of output signals. Attempts to use system reset during a hard reset sequence or while the JTAG logic is nonidle causes unpredictable results. Processing interrupted by a system reset can be restarted.

5.3 Machine Check and Checkstops

The PowerPC architecture defines a machine check exception which is used for diagnostics. Generally when a condition that generates a machine check is present, whether the exception is taken is determined by the value of the machine check enable bit, MSR[ME]. If it is cleared, the machine check exception is disabled and the processor instead enters checkstop state, which is described in the following section.

For a detailed discussion of the machine check exception, see Section 5.3.2, “Machine Check Exception (0x00200).”

5.3.1 Checkstop State (MSR[ME] = 0)

When a processor is in the checkstop state, instruction processing is suspended and generally cannot be restarted without resetting the processor. The contents of all latches (except any associated with the bus clock) are frozen within two cycles upon entering checkstop state so that the state of the processor can be analyzed as an aid in problem determination.

A machine check exception may result from referencing a nonexistent physical address, either directly (with MSR[DR] = 0), or through an invalid translation. On such a system, for example, execution of a Data Cache Block Set to Zero (**dcbz**) instruction that introduces a block into the cache associated with a nonexistent physical address may delay the machine check exception until an attempt is made to store that block to main memory.

Note that not all PowerPC processors provide the same level of error checking. The reasons a processor can enter the checkstop state are implementation-dependent.

5.3.2 Machine Check Exception (0x00200)

If no higher-priority exception is pending (namely, a hard reset), the processor initiates a machine check exception when the appropriate condition is detected. Note that the causes of machine check exceptions are implementation- and system-dependent, and are typically signalled to the processor by the assertion of a specified signal on the processor interface.

When a machine check condition occurs and MSR[ME] = 1, the exception is recognized and handled. If MSR[ME] = 0 and a machine check occurs, the processor generates an internal checkstop condition. When a processor is in checkstop state, instruction processing is suspended and generally cannot continue without resetting the processor. Some implementations may preserve some or all of the internal state of the processor when entering the checkstop state, so that the state can be analyzed as an aid in problem determination.

In general, it is expected that a bus error signal would be used by a memory controller to indicate a memory parity error or an uncorrectable memory ECC error. Note that the resulting machine check exception has priority over any exceptions caused by the instruction that generated the bus operation.

If a machine check exception causes an exception that is not context synchronizing, the exception is not recoverable. Also, if a machine check exception causes the loss of one of the following exceptions, the exception is not recoverable:

- An external exception (interrupt or decrementer)
- Direct-store error type DSI exception
- Floating-point enabled type program exception, If the SRR1 bit corresponding to MSR[RI] is cleared, the exception is context synchronizing only with respect to subsequent instructions. If the exception is recoverable, the SRR1 bit corresponding to MSR[RI] is set and the exception is context synchronizing.

On some implementations, a machine check exception may be caused by referring to a nonexistent physical (real) address, either because translation is disabled (MSR[IR] or MSR[DR] = 0) or through an invalid translation. On such a system, execution of the **dcbz** instruction can cause a delayed machine check exception by introducing a block into the data cache that is associated with an invalid physical (real) address. A machine check exception could eventually occur when and if a subsequent attempt is made to store that block to memory.

When a machine check exception is taken, registers are updated as shown in Table 5-6.

Table 5-6. Machine Check Exception—Register Settings

Register	Setting Description
SRR0	On a best-effort basis, implementations can set this to an EA of some instruction that was executing or about to be executing when the machine check condition occurred.
SRR1	Bit 30 is loaded from MSR[RI][1] if the processor is in a recoverable state. Otherwise cleared. The setting of all other SRR1 bits is implementation-dependent.
MSR	POW[1] 0 PR 0 SE 0 IR [5] 0 TGPR [2] 0 FP 0 BE 0 DR [6] 0 ILE[1] — ME[3] — FE1 0 RI[1] 0 EE 0 FE0 0 IP [4] — LE [7] Set to value of ILE

[1] Not implemented on the 601
[3] 603 only
[3] Note that when a machine check exception is taken, the exception handler should set MSR[ME] as soon as it is practical to handle another machine check exception. Otherwise, subsequent machine check exceptions cause the processor to automatically enter the checkstop state.
[4] Identified as EP on the 601
[5] Identified as IT on the 601
[6] Identified as DT on the 601
[7] Not implemented on the 601. Control of little-endian mode on the 601 is provided by HID0[28], the LM bit.

If MSR[RI] is set, the machine check exception may still be unrecoverable in the sense that execution cannot resume in the same context that existed before the exception.

When a machine check exception is taken, instruction execution resumes at offset 0x00200.

5.3.2.1 Machine Check Exception (0x00200)— PowerPC 601 Processor

The 601 conditionally initiates a machine check exception after detecting the assertion of the $\overline{\text{TEA}}$ signal, which indicates that a bus error occurred and the system terminates the current transaction. One clock cycle after $\overline{\text{TEA}}$ is asserted, the data bus signals go to the high-impedance state; however, data entering the GPR or the cache is not invalidated.

If the MSR[ME] bit is set, the exception is recognized and handled; otherwise, the 601 attempts to enter an internal checkstop condition. This may not lead to a checkstop depending upon the state of the various checkstop enable control bits in the HID0 register. These are described in Section 5.3.2.2.1, “Checkstop Sources and Enables Register—HID0.”

If MSR[ME], HID0[CE], and HID0[EM] bits are cleared (that is, when both the master checkstop and the machine check checkstops are disabled), the machine check exception is taken.

In general, it is expected that the $\overline{\text{TEA}}$ signal would be used by a memory controller to indicate a memory parity error or an uncorrectable memory ECC error. Note that the resulting machine check exception is imprecise and has priority over any exceptions caused by the instruction that generated the bus operation.

5.3.2.2 Checkstop State (MSR[ME] = 0)—PowerPC 601 Processor

When a processor is in checkstop state, instruction processing is suspended and generally cannot be restarted without resetting the processor. The contents of all latches are frozen within two cycles upon entering checkstop state so that the state of the processor can be analyzed as an aid in problem determination.

A machine check exception may result from referring to a nonexistent physical address. In some implementations, for example, execution of a Data Cache Block Set to Zero (**dcbz**) instruction that introduces a block into the cache associated with a nonexistent physical address may delay the machine check exception until an attempt is made to store that block to main memory.

Checkstop sources and enables for the 601 are described in the following section.

5.3.2.2.1 Checkstop Sources and Enables Register—HID0

The checkstop sources and enables register (HID0), shown in Figure 5-1, is a supervisor-level register that defines enable and monitor bits for each of the checkstop sources in the 601. The SPR number for HID0 is 1008.

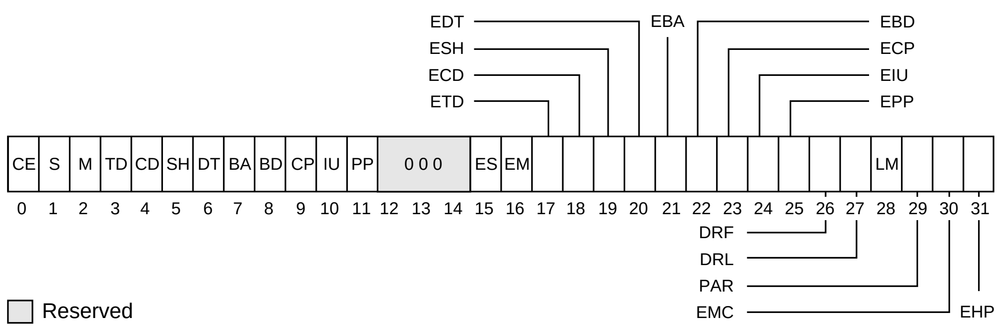

Figure 5-1. HID0—Checkstop Sources and Enables Register (601)

Table 5-7 defines the bits in HID0. The enable bits (bits 15–31) can be used to mask individual checkstop sources, although these are provided primarily to mask off any false reports of such conditions for debugging purposes. Bit 0 (HID0[CE]) is a master checkstop enable; if it is cleared, all checkstop conditions are disabled; if it is set, individual conditions can be enabled separately. HID0[EM] (bit 16) enables and disables machine check checkstops; clearing this bit masks machine check checkstop conditions that occur when MSR[ME] is cleared. Bits 1–11 are the checkstop source bits, and can be used to determine the specific cause of a checkstop condition.

All enable bits except 15 and 24 are disabled at start up. The operating system should enable these checkstop conditions before the power-on reset sequence is complete.

Table 5-7. HID0—Checkstop Sources and Enables Register (601)

Bit	Name	Description
0	CE	Master checkstop enable. Enabled if set. If this bit is cleared and the $\overline{\text{TEA}}$ signal is asserted, a machine check exception is taken, regardless of the setting of MSR[ME].
1	S	Microcode checkstop detected if set.
2	M	Double machine check detected if set.
3	TD	Multiple TLB hit checkstop if set.
4	CD	Multiple cache hit checkstop if set.
5	SH	Sequencer time out checkstop if set.
6	DT	Dispatch time out checkstop if set.
7	BA	Bus address parity error if set.
8	BD	Bus data parity error if set.
9	CP	Cache parity error if set.
10	IU	Invalid microcode instruction if set.
11	PP	Direct-store interface access protocol error if set.
12–14	—	Reserved
15	ES	Enable microcode checkstop. Enabled by hard reset. Enabled if set.
16	EM	Enable machine check checkstop. Disabled by hard reset. Enabled if set. If this bit is cleared and the $\overline{\text{TEA}}$ signal is asserted, a machine check exception is taken, regardless of the setting of MSR[ME].
17	ETD	Enable TLB checkstop. Disabled by hard reset. Enabled if set.
18	ECD	Enable cache checkstop. Disabled by hard reset. Enabled if set.
19	ESH	Enable sequencer time out checkstop. Disabled by hard reset. Enabled if set.
20	EDT	Enable dispatch time out checkstop. Disabled by hard reset. Enabled if set.
21	EBA	Enable bus address parity checkstop. Disabled by hard reset. Enabled if set.
22	EBD	Enable bus data parity checkstop. Disabled by hard reset. Enabled if set.
23	ECP	Enable cache parity checkstop. Disabled by hard reset. Enabled if set.
24	EIU	Enable for invalid ucode instruction checkstop. Enabled by hard reset. Enabled if set.
25	EPP	Enable for direct-store protocol checkstop. Disabled by hard reset. Enabled if set.
26	DRF	0 Optional reload of alternate sector on instruction fetch miss is enabled. 1 Optional reload of alternate sector on instruction fetch miss is disabled.
27	DRL	0 Optional reload of alternate sector on load/store miss is enabled. 1 Optional reload of alternate sector on load/store miss is disabled.
28	LM	0 Big-endian mode is enabled. 1 Little-endian mode is enabled.
29	PAR	0 Precharge of the $\overline{\text{ARTRY}}$ and $\overline{\text{SHD}}$ signals is enabled. 1 Precharge of the $\overline{\text{ARTRY}}$ and $\overline{\text{SHD}}$ signals is disabled.

Table 5-7. HID0—Checkstop Sources and Enables Register (601) (Continued)

Bit	Name	Description
30	EMC	0 No error detected in main cache during array initialization. 1 Error detected in main cache during array initialization.
31	EHP	0 The $\overline{\text{HP\_SNP\_REQ}}$ signal is disabled. Use of the associated queue position is restricted to a snoop hit that occurs when a read is pending. That is, its address tenure is complete but the data tenure has not begun. 1 The $\overline{\text{HP\_SNP\_REQ}}$ signal is enabled. Use of the associated queue position is restricted to a snoop hit on an address tenure that had $\overline{\text{HP\_SNP\_REQ}}$ asserted.

Checkstop enable bits can be set or cleared without restriction. If a checkstop source bit is set, it can be cleared; however, if the corresponding checkstop condition is still present on the next clock, the bit will be set again. A checkstop source bit can only be set when the corresponding checkstop condition occurs and the checkstop enable bit is set; it cannot be set via an **mtspr** instruction. That is, you cannot manually cause a checkstop.

The HID0 register is set to 0x80010080 by the hard reset operation. However, the state of the EMC bit depends on the results of the power-on diagnostics for the main cache array. This bit is set if the cache fails the built-in self test during the power-on sequence.

5.3.2.3 Machine Check Exception—PowerPC 603 Processor

The 603 conditionally initiates a machine check exception after detecting the assertion of the $\overline{\text{TEA}}$ or $\overline{\text{MCP}}$ signals on the 603 bus (assuming the machine check is enabled, MSR[ME] = 1). The assertion of one of these signals indicates that a bus error occurred and the system terminates the current transaction. One clock cycle after the signal is asserted, the data bus signals go to the high-impedance state; however, data entering the GPR or the cache is not invalidated. Note that if HID0[EMCP] is cleared, the processor ignores the assertion of the $\overline{\text{MCP}}$ signal.

Register settings when the 603 takes a machine check exception are described in Table 5-6.

Note that the 603 makes no attempt to force recoverability; however, it does guarantee the machine check exception is always taken immediately upon request, with a nonpredicted address saved in SRR0, regardless of the current machine state. Any pending stores in the completed store queue are cancelled when the exception is taken. Software can use the machine check exception in a recoverable mode for checking bus configuration. For this case, a **sync**, load, **sync** instruction sequence is used. A subsequent machine check exception at the load address indicates a bus configuration problem and the processor is in a recoverable state.

If MSR[ME] is set, the exception is recognized and handled; otherwise, the 603e attempts to enter an internal checkstop. Note that the resulting machine check exception has priority over any exceptions caused by the instruction that generated the bus operation.

5.3.2.4 Checkstop State (MSR[ME] = 0)—PowerPC 603 Processor

When the 603 enters checkstop state, it asserts the checkstop output signal, $\overline{\text{CKSTP\_OUT}}$. The following events will cause the 603e to enter the checkstop state:

- Machine check exception occurs with MSR[ME] cleared.
- External checkstop input, $\overline{\text{CKSTP\_IN}}$, is asserted.
- A direct-store protocol error occurs.

When a processor is in checkstop state, instruction processing is suspended and generally cannot be restarted without resetting the processor. The contents of all latches are frozen within two cycles upon entering the checkstop state so that the state of the processor can be analyzed as an aid in problem determination.

Note that not all PowerPC processors provide the same level of error checking. The reasons a processor can enter checkstop state are implementation-dependent.

5.3.2.5 Machine Check Exception—PowerPC 604 Processor

The 604 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 a cache, after receiving a qualified transfer error acknowledge ($\overline{\text{TEA}}$) indication on the 604 bus, or after the machine check interrupt ($\overline{\text{MCP}}$) signal had been asserted. As defined in the OEA, the exception is not taken if the MSR[ME] is cleared.

Machine check conditions can be enabled and disabled using bits in the HID0 register described in Table 5-8.

Table 5-8. Machine Check Enable Bits

HID0 Bit	Description
0	Enable machine check input pin
1	Enable cache parity checking
2	Enable machine check on address bus parity error
3	Enable machine check on data bus parity error

A $\overline{\text{TEA}}$ indication on the bus can result from any load or store operation initiated by the processor. In general, the $\overline{\text{TEA}}$ signal 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 the MSR[ME] bit and the appropriate bits in HID0 are set, the exception is recognized and handled; otherwise, the processor generates an internal checkstop condition. When a processor is in checkstop state, instruction processing is suspended and generally cannot continue without restarting the processor. Note that many conditions may lead to the checkstop condition; the disabled machine check exception is only one of these.

Machine check exceptions are enabled when MSR[ME] = 1; this is described in Section 5.3.2.5.1, “Machine Check Exception Enabled (MSR[ME] = 1).” If MSR[ME] = 0 and a machine check occurs, the processor enters the checkstop state. Checkstop state is described in Section 5.3.1, “Checkstop State (MSR[ME] = 0).”

5.3.2.5.1 Machine Check Exception Enabled (MSR[ME] = 1)

When a machine check exception is taken, registers are updated as shown in Table 5-6.

The machine check exception is usually unrecoverable in the sense that execution cannot resume in the same context that existed before the exception. 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. Typically earlier processes cannot resume; however, the operating systems can then use the machine check exception handler to try to identify and log the cause of the machine check condition.

5.3.2.5.2 Checkstop State (MSR[ME] = 0)

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.

A machine check exception may result from referencing a nonexistent physical address, either directly (with MSR[DR] = 0), or through an invalid translation. On such a system, for example, execution of a Data Cache Block Set to Zero (**dcbz**) instruction that introduces a block into the cache associated with a nonexistent physical address may delay the machine check exception until an attempt is made to store that block to main memory.

5.4 External Interrupt Exception (0x00500)

The PowerPC architecture defines an external interrupt exception, which in the 60x processors is signaled to the processor by the assertion of the external interrupt signal, $\overline{\text{INT}}$. The exception may be delayed by other higher-priority exceptions or if the MSR[EE] bit is zero when the exception is detected. Note that the occurrence of this exception does not cancel the external request.

The register settings for the external interrupt exception are shown in Table 5-9.

Table 5-9. External Interrupt—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 interrupt conditions were present. On the 603, note that in the rare case when the next instruction is not in the completion queue, the 603 searches elsewhere to provide the appropriate restart instruction address to SRR0.
SRR1	0 Loaded with equivalent bits from the MSR (cleared in the 601 and 603) 1–4 Cleared 5–9 Loaded with equivalent bits from the MSR (cleared in the 601 and 603) 10–15 Cleared 16–31 Loaded with equivalent bits from the MSR Note that depending on the implementation, reserved bits in the MSR may not be copied to SRR1.
MSR	POW[1] 0 PR 0 SE 0 IR[4] 0 TGPR[2] 0 FP 0 BE 0 DR[5] 0 ILE[1] — ME — FE1 0 RI[1] 0 EE 0 FE0 0 IP[3] — LE[6] Set to value of ILE

[1] Not implemented on the 601
[2] 603e only
[3] Identified as EP on the 601
[4] Identified as IT on the 601
[5] Identified as DT on the 601
[6] Not implemented on the 601. Control of little-endian mode on the 601 is provided by HID0[28], the LM bit.

Note that the processor recognizes the interrupt condition ($\overline{\text{INT}}$ asserted) only if MSR[EE] is set. To guarantee that the external interrupt is taken, $\overline{\text{INT}}$ must remain asserted until the processor takes the interrupt; otherwise, the processor is not guaranteed to take an external interrupt.

After the $\overline{\text{INT}}$ is detected asserted, the processor stops dispatching instructions and waits for executing instructions to complete. Therefore, exceptions caused by instructions in progress are taken before the external interrupt exception is taken. After all instructions complete, the processor takes the external interrupt exception.

The interrupt handler must send a command to the device that asserted $\overline{\text{INT}}$, acknowledging the interrupt and instructing the device to negate $\overline{\text{INT}}$.

When an external interrupt exception is taken, instruction execution resumes at offset 0x00500 from the physical base address indicated by MSR[IP].

5.4.1 External Interrupt—PowerPC 601 Processor

In early versions of the 601 (processor revision level 0x0000), the external interrupt is a level-sensitive signal and should be held active until reset by the interrupt service routine. Phantom interrupts due to phenomena such as crosstalk and bus noise should be avoided.

5.4.2 External Interrupt—PowerPC 603 Processor

On the 603, note that in the rare case when the next instruction is not in the completion queue, the 603 searches elsewhere to provide the appropriate restart instruction address to SRR0.

5.5 System Management Interrupt Exception (0x01400)

The system management interrupt, which is implemented on the 603 and 604, but not on the 601, behaves like an external interrupt except for the signal asserted and the vector taken.

A system management interrupt is signaled to the processor by the assertion of the $\overline{\text{SMI}}$ signal. The interrupt may not be recognized if a higher-priority exception occurs simultaneously or if the MSR[EE] bit is cleared when $\overline{\text{SMI}}$ is asserted. Note that $\overline{\text{SMI}}$ takes priority over $\overline{\text{INT}}$ if they are recognized simultaneously.

After the assertion of $\overline{\text{SMI}}$ is detected (and provided that MSR[EE] is set), the processor waits for the next instruction (and any exceptions associated with that instruction) to complete before taking the system management interrupt. Note that in the rare case when the next instruction is not in the completion queue, the processor searches elsewhere to provide the appropriate restart instruction address to SRR0.

The register settings for the system management interrupt exception are the same as those for the external interrupt, as shown in Table 5-10.

Table 5-10. System Management Interrupt—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 interrupt conditions were present.
SRR1	0 Loaded with equivalent bits from the MSR (cleared in the 601) 1–4 Cleared 5–9 Loaded with equivalent bits from the MSR (cleared in the 601) 10–15 Cleared 16–31 Loaded with equivalent bits from the MSR Note that depending on the implementation, reserved bits in the MSR may not be copied to SRR1.
MSR	POW 0 PR 0 SE 0 IR 0 TGPR [1] 0 FP 0 BE 0 DR 0 ILE — ME — FE1 0 RI 0 EE 0 FE0 0 IP — LE Set to value of ILE

[1]603e only

When a system management interrupt is taken, instruction execution for the handler begins at offset 0x01400 from the physical base address indicated by MSR[IP].

The processor recognizes the interrupt condition ($\overline{\text{SMI}}$ asserted) only if MSR[EE] is set; it ignores the interrupt condition if the MSR[EE] bit is cleared. To guarantee that the external interrupt is taken, the $\overline{\text{SMI}}$ signal must be held active until the processor takes the exception. If the $\overline{\text{SMI}}$ signal is negated before the interrupt is taken, the processor is not guaranteed to take a system management interrupt. The interrupt handler must send a command to the device that asserted $\overline{\text{SMI}}$, acknowledging the interrupt and instructing the device to negate $\overline{\text{SMI}}$.

Chapter 6
Additional Bus Configurations

Chapters 2 through 5 describe basic 60x bus operations. However some processors support additional bus functionality, including the following:

- No-data retry mode (referred to as no-DRTRY mode).—This mode allows DRTRY to be disabled in the 603 and 604e, which in turn allows data to be forwarded one bus cycle sooner than if DRTRY is enabled. (No-DRTRY mode is implemented on the 604e, but not on the 604; see data streaming mode below.)
- Data streaming mode—Data streaming is the ability to begin data tenure after a previous data tenure with no dead cycles between. Data streaming is implemented on 604s. (Note that in 604 documentation, this was called fast-L2/data streaming mode and no-DRTRY/data streaming mode, although there is no relation to the no-DRTRY mode described above.)
- 32-bit data bus mode—The 603 supports an optional 32-bit data bus mode, in which the processor uses only byte lanes 0–3 for a data transfer, therefore allowing a maximum of 32 bits of data to be transferred per bus clock.
- Reduced pinout mode—The 603 provides an optional reduced-pinout mode that disables DL[0–31], DP[0–7], AP[0–3], APE, DPE, and RSRV for reduced power consumption. The 32-bit data bus mode is implicitly selected when reduced-pinout mode is enabled.
- Direct-store mode, which provides an alternative method for I/O bus operations, is described in Chapter 7, “Direct-Store Interface.”

6.1 No-DRTRY Mode (603 and 604e)

The 603 family and 604e processors provides a way to disable the use of the data retry function. No-DRTRY mode allows data to be forwarded during load operations to the internal processor one bus cycle sooner than with normal bus protocol.

The 60x bus protocol specifies that, during load operations, the memory system normally can cancel data that the master read on the bus cycle after TA was asserted. On 603 and 604e processors, this late cancellation requires any data loaded at the bus interface to be held one additional bus clock to verify that the it is valid before forwarding it to the internal CPU. For systems that do not use the DRTRY function, no-DRTRY mode eliminates this one-cycle stall and allows data to pass to the internal CPU immediately when TA is recognized.

When the processor is in no-$\overline{\text{DRTRY}}$ mode, data can no longer be cancelled the cycle after it is acknowledged by an assertion of $\overline{\text{TA}}$. Data is immediately forwarded to the CPU internally, and any attempt at late cancellation by the system may cause improper operation by the processor.

When the 603e uses normal bus protocol, data can be cancelled the bus cycle after $\overline{\text{TA}}$ by either late cancellation by $\overline{\text{DRTRY}}$ or by $\overline{\text{ARTRY}}$. When no-$\overline{\text{DRTRY}}$ mode is selected, both cancellation cases must be disallowed in the system design for the bus protocol.

No-$\overline{\text{DRTRY}}$ mode requires the system to ensure that $\overline{\text{DRTRY}}$ not be asserted to the processor, which may cause improper operation of the bus interface. The system must also ensure that a snooping device does not assert $\overline{\text{ARTRY}}$ later than the first assertion of $\overline{\text{TA}}$ to the processor, but not on the cycle after the first assertion of $\overline{\text{TA}}$.

Apart from the inability to cancel data that was read by the master on the bus cycle after $\overline{\text{TA}}$ was asserted, the 603 bus protocol is identical to that for the basic transfer bus protocols, as well as for 32-bit data bus mode.

The processor selects the desired $\overline{\text{DRTRY}}$ mode at start-up by sampling $\overline{\text{DRTRY}}$ at the negation of $\overline{\text{HRESET}}$. If $\overline{\text{DRTRY}}$ is negated, normal operation is selected; if it is asserted, no-$\overline{\text{DRTRY}}$ mode is selected.

6.1.1 No-$\overline{\text{DRTRY}}$ Mode in PowerPC 604e Processor

In no-$\overline{\text{DRTRY}}$ mode, the system must define the beginning of the window in which the snoop response is valid and ensure that no data is transferred before the same cycle as the beginning of that window. For example, if the system defines a snoop response window that begins the second cycle after $\overline{\text{TS}}$, $\overline{\text{TA}}$ can be asserted no sooner than the second cycle after $\overline{\text{TS}}$. This timing constraint on the earliest allowable assertion of $\overline{\text{TA}}$ with respect to $\overline{\text{ARTRY}}$ is identical to that constraint in data streaming mode.

To upgrade a 604-based system to the 604e and use no-$\overline{\text{DRTRY}}$, the following should be observed:

- The system uses the 604 in normal bus mode, described earlier in this section.
- $\overline{\text{DRTRY}}$ must be tied negated and never used.
- The system must never assert $\overline{\text{TA}}$ before the first cycle of the system’s snoop response window.

This system would then see a performance improvement due to the shorter effective latency seen by the 604e on read operations. This improvement is equal to one bus cycle (three processor cycles in 3:1 bus mode).

6.2 Data Streaming Mode (604)

Data streaming is the ability to start a data tenure after a previous data tenure with no dead cycles between. (Note that in 604 documentation, this was called fast-L2/data streaming mode and no-$\overline{\text{DRTRY}}$/data streaming mode, although there is no connection to the no-$\overline{\text{DRTRY}}$ mode described above.) The 604 supports data streaming only for consecutive burst-read data transfers. This does include support for data streaming consecutive burst read data transfers between two separate masters. For instance, in a multiple-604 system, data streaming is allowed on consecutive burst read data transfers from different 604s.

To cause data streaming, the system asserts $\overline{\text{DBG}}$ during the last data transfer of the first data tenure as shown in Figure 6-1. To fully realize the performance gain of data streaming, the system should be prepared to, but is not required to, supply an uninterrupted sequence of $\overline{\text{TA}}$ assertions.

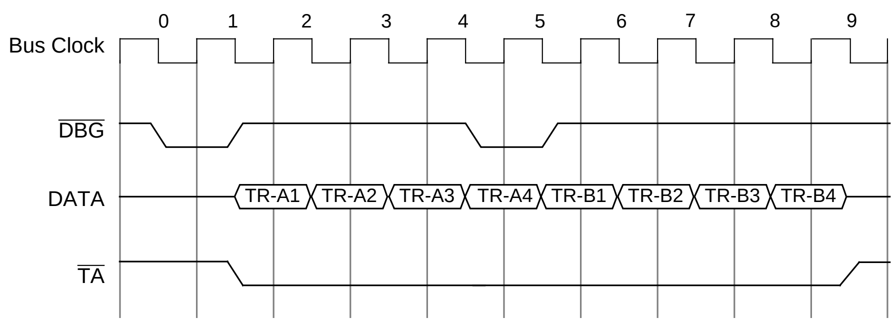

Figure 6-1. Data Transfer in Data Streaming Mode

6.2.1 Data Valid Window in the Data Streaming Mode

Standard bus mode operations allow data to be transferred no earlier than the cycle before the $\overline{\text{ARTRY}}$ window that the system defines. In some cases, an asserted $\overline{\text{ARTRY}}$ invalidates the data that was transferred the previous cycle, in the same way $\overline{\text{DRTRY}}$ cancels data from the previous cycle.

In data streaming mode, the data buffering that allows late cancellation of a data transfer does not exist, so late cancellation with $\overline{\text{ARTRY}}$ is also impossible. Therefore, the earliest that data can be transferred in data streaming mode is the first cycle of the $\overline{\text{ARTRY}}$ window, not the cycle before that.

6.2.2 Data Valid Window in the Data Streaming Mode

Standard bus mode operations allow data to be transferred no earlier than the cycle before the $\overline{\text{ARTRY}}$ window that the system defines. In some cases, an asserted $\overline{\text{ARTRY}}$ invalidates the data that was transferred the previous cycle, in the same way $\overline{\text{DRTRY}}$ cancels data from the previous cycle.

In data streaming mode, the data buffering that allows late cancellation of a data transfer does not exist, so late cancellation with $\overline{\text{ARTRY}}$ is also impossible. Therefore, the earliest that data can be transferred in data streaming mode is the first cycle of the $\overline{\text{ARTRY}}$ window, not the cycle before that.

6.2.3 Design Practices for Data Streaming Mode

It is recommended that use of data streaming mode be accompanied by two other system design practices:

- Do not use $\overline{\text{ABB}}$. If the system is designed so an address tenure is defined by $\overline{\text{TS}}$ and $\overline{\text{AACK}}$ assertion, (which the 604 is designed to support), $\overline{\text{ABB}}$ is unnecessary and should be pulled high at the 604. Because $\overline{\text{ABB}}$ has a short restore-high time, $\overline{\text{ABB}}$ should not be used in systems that try to achieve a short cycle time.
- Do not use $\overline{\text{DBB}}$, which is restored high in the same way as $\overline{\text{ABB}}$ and therefore has the same problems in a system with short cycle times. To avoid using $\overline{\text{DBB}}$, the system arbiter must assert $\overline{\text{DBG}}$ for a single cycle, one cycle before the 604 is supposed to begin its data tenure. The $\overline{\text{DBB}}$ signal should be pulled high. The additional system cost of operating in this manner is that data transfers must be counted and $\overline{\text{DBG}}$ can be asserted only on the last cycle in a data tenure.

6.3 32-Bit Data Bus Mode (603)

The 603 supports an optional 32-bit data bus mode, which operates like the 64-bit data bus mode but uses only byte lanes 0–3, corresponding to DH[0–31] and DP[0–3]. Byte lanes 4–7 (DL[0–31] and DP[4–7]) are never used in this mode. Unused bus signals are ignored during read operations and are driven low for write operations.

In 32-bit bus mode, data tenures can be 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. Note that two-beat burst transactions do not assert $\overline{\text{TBST}}$ (having the same $\overline{\text{TBST}}$ and TSIZ[0–2] encodings as the 64-bit data bus mode).

Single-beat data transactions transfer four bytes or less, and two-beat data transactions are performed for eight-byte operations only. The 603 generates an eight-byte operation only for a double-word–aligned load double or store double operation to or from the floating-point registers (FPRs).

Eight-beat burst data transactions load data into or store data from the 603's internal caches. These transactions transfer 32 bytes in the same way as in 64-bit data bus mode, asserting $\overline{\text{TBST}}$ and signalling 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 for both 32- and 64-bit bus modes. For word or smaller transactions, late $\overline{\text{ARTRY}}$ cancellation of the data tenure applies on the bus clock after the first data beat is acknowledged (after the first $\overline{\text{TA}}$); for double-word or burst operations, this may occur on the bus clock after the second data beat is acknowledged (after the second $\overline{\text{TA}}$ or coincident with respective $\overline{\text{TA}}$ if no-$\overline{\text{DRTRY}}$ mode is selected).

An example of an eight-beat data transfer while the 603 is in 32-bit data bus mode is shown in Figure 6-2. In this example, $\overline{\text{TA}}$ remains asserted for the entire burst transaction.

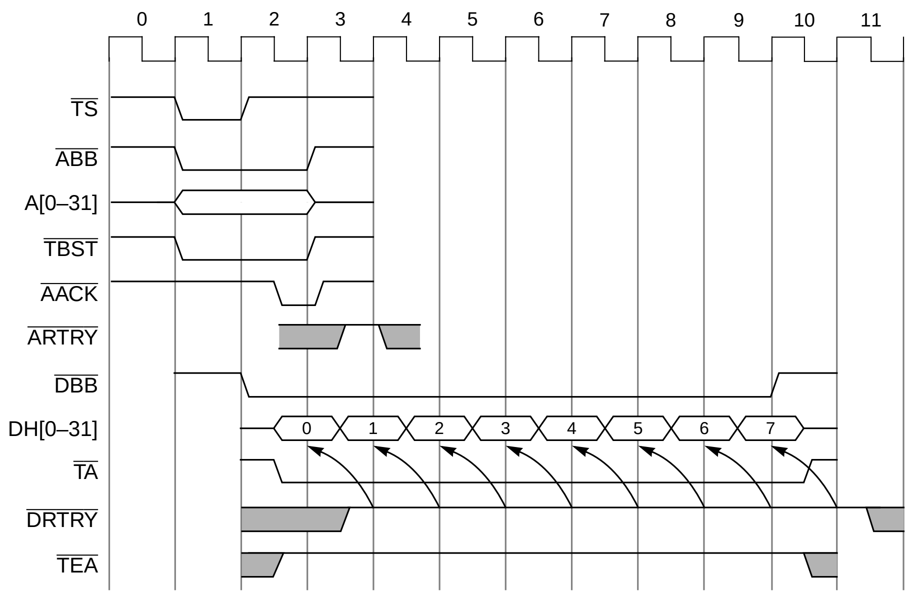

Figure 6-2. 32-Bit Data Bus Transfer (Eight-Beat Burst)

Figure 6-3 shows an example of a two-beat data transfer (with $\overline{\text{DRTRY}}$ asserted during each data tenure).

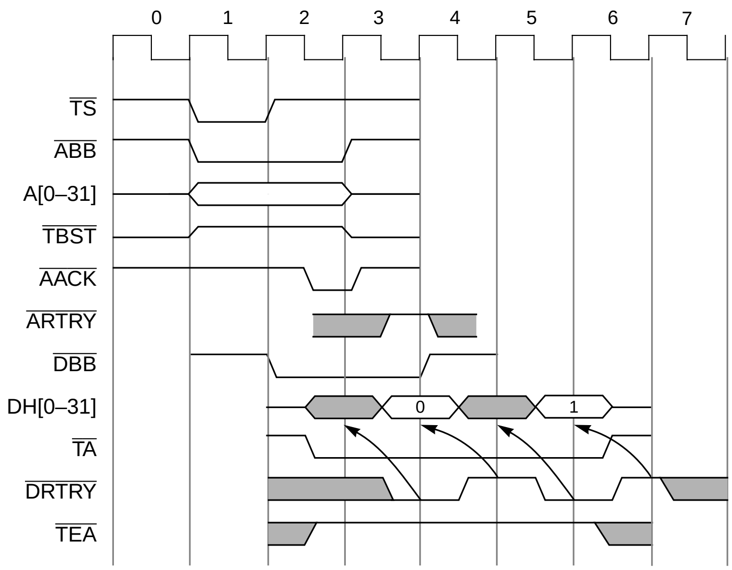

Figure 6-3. 32-Bit Data Bus Transfer (Two-Beat Burst with $\overline{\mathrm{DRTRY}}$)

The 603 selects the data bus mode at start-up by sampling $\overline{\mathrm{TLBISYNC}}$ at the negation of $\overline{\mathrm{HRESET}}$. If $\overline{\mathrm{TLBISYNC}}$ is asserted, the bus runs in 32-bit data mode; otherwise, it runs in 64-bit mode. If the $\overline{\mathrm{TLBISYNC}}$ input function is not used, it can be connected to $\overline{\mathrm{HRESET}}$ to place the processor in 32-bit bus mode. Otherwise, it should be connected to a pull-up resistor to select 64-bit mode. For systems using the $\overline{\mathrm{TLBISYNC}}$ input function, $\overline{\mathrm{HRESET}}$ must be logically combined with $\overline{\mathrm{TLBISYNC}}$ to select a data bus mode.

6.4 Reduced-Pinout Mode (603)

The 603 has an optional reduced-pinout mode. This mode idles the switching of numerous signals for reduced power consumption. The DL[0–31], DP[0–7], AP[0–3], $\overline{\mathrm{APE}}$, $\overline{\mathrm{DPE}}$, and $\overline{\mathrm{RSRV}}$ signals are disabled when the reduced-pinout mode is selected. Note that the 32-bit data bus mode is implicitly selected when the reduced-pinout mode is enabled.

When the 603 is in reduced-pinout mode, the bidirectional and output pins disabled are always driven low during the periods when normally they would have been driven by the 603. The open-drain outputs ($\overline{\mathrm{APE}}$ and $\overline{\mathrm{DPE}}$) are always three-stated. Bidirectional inputs are always turned off at the input receivers of the 603 and are not sampled.

The 603 selects either full-pinout or reduced-pinout mode at start-up by sampling the state of $\overline{\mathrm{QACK}}$ at the negation of $\overline{\mathrm{HRESET}}$. If $\overline{\mathrm{QACK}}$ is low at the negation of $\overline{\mathrm{HRESET}}$, full-pinout mode is selected by the 603. If $\overline{\mathrm{QACK}}$ is high at the negation of $\overline{\mathrm{HRESET}}$, reduced-pinout mode is selected.

Chapter 7
Direct-Store Interface

Accesses to direct-store segments, as defined in the PowerPC architecture, are executed on the bus using the extended transfer protocol (ETP), an extension to the basic transfer protocol described in previous chapters. Except for one signal, $\overline{\text{XATS}}$, this protocol uses the same signal set as the basic transfer protocol, although some signals are redefined. The PowerPC 601 processor documentation refers to the direct-store interface as the I/O controller interface.

Direct-store operations are no longer required by the PowerPC architecture. Some processors, such as the PowerPC 603e processor, do not support this feature.

PowerPC architecture defines the following characteristics for direct-store accesses:

- The extended address delivered to the I/O system includes a bus unit ID (BUID) to address one of several bus devices and a 32-bit address to be delivered to each.
- A transaction error can be detected and associated with the original instruction.

To satisfy the requirements of PowerPC architecture for direct-store segments, the following extensions are implemented:

- A new set of bus operations is provided that redefines how the transfer type (TT*n*), transfer burst ($\overline{\text{TBST}}$), and transfer size (TSIZ*n*) signals are used. These signals together generate the extended address transfer code (XATC), as shown in Table 7-4.
- Each direct-store address transfer takes two beats. The first transmits the BUID and several control bits from the segment register, and the second transfers a complete 32-bit address to the slave device.
- Explicit sender/receiver tags are provided.
- A split-response protocol is enforced; that is, the sender must wait for a reply from the receiver before considering a transaction complete.
- The 60x does not burst direct-store transactions, but a type of streaming is permitted. Streaming (in this context) allows multiple single-beat transactions to occur before a reply from the direct-store receiver is required.

Direct-store transactions are like memory-mapped accesses, as shown in Figure 7-1. They use most of the same signals. They use separate arbitration for the split address and data buses, and also define address-only and single-beat transactions. The address retry vehicle is identical, although there is no hardware coherency support.

A given 60x processor processes one direct-store transaction at a time, but may perform other bus transactions for the duration of the transfer. A direct-store cycle does not inhibit other bus traffic within its envelope.

In addition to the extensions noted above, there are fundamental differences between the basic transfer protocol and the extensions. For example, use of $\overline{\text{DRTRY}}$ is undefined. Also, only four bytes of the eight-byte data path are available (transmitted on DH[0–31]. This facilitates lower pin-count direct-store interfaces but also offer substantially less bandwidth than memory accesses. Additionally, load/store instructions to direct-store addresses cannot retire until an error-free reply is received, which likely further degrades performance, compared to access to normal segments.

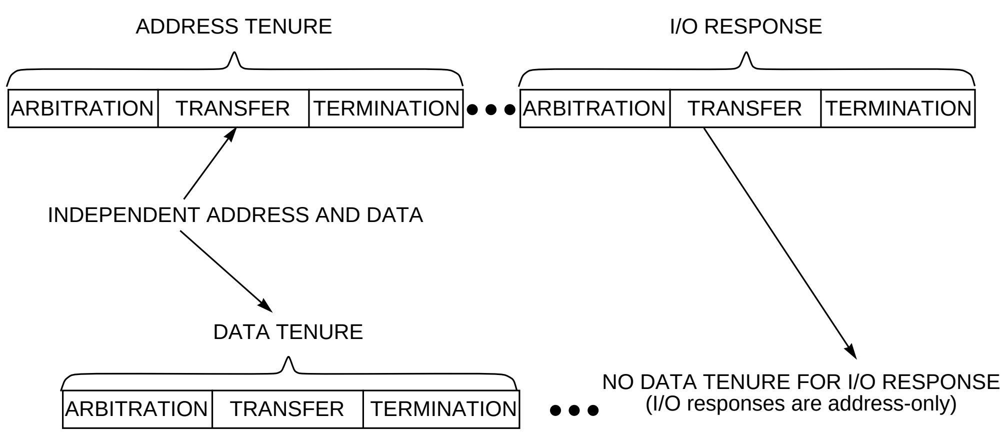

Figure 7-1. Direct-Store Interface Protocol Tenures

The 601 supports an additional mode, memory-forced direct-store mode, that is not defined by the PowerPC architecture and dependent on the value of BUID. This is described in Section 7.6, “Memory-Forced Direct-Store Interface (PowerPC 601 Processor Only).”

7.1 Direct-Store Transaction Protocol Details

As mentioned previously, there are two address-bus beats corresponding to two packets of information about the address. The two packets contain the sender and receiver tags, the address and extended address bits, and extra control and status bits. The two beats of the address bus (plus attributes) are shown at the top of Figure 7-2 as two packets. Packet 0 is then expanded to reflect the XATC and address bus information in detail.

7.1.1 Packet 0

Figure 7-2 shows the organization of the first packet in a direct-store transaction. The XATC contains the transfer code. The address bus contains the following:

Key bit || segment register || sender tag

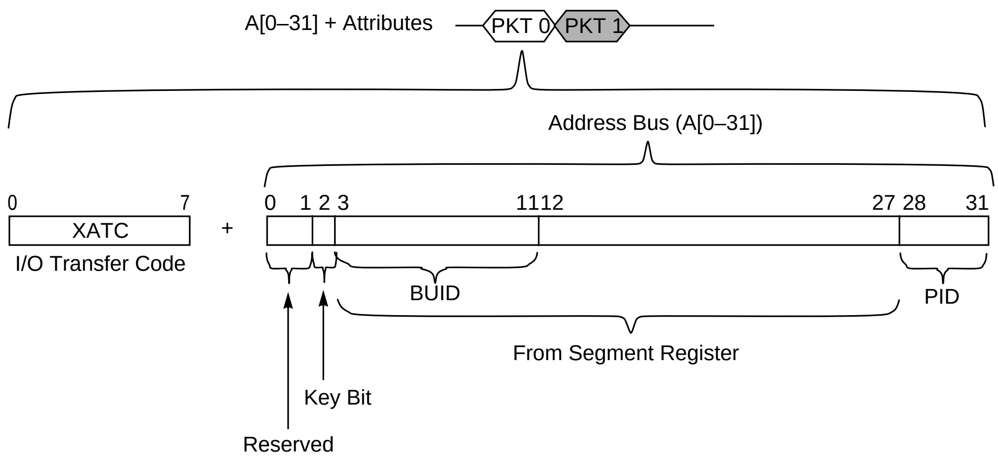

Figure 7-2. Direct-Store Operation—Packet 0

The contents of the address bus are described in Table 7-1.

Table 7-1. Address Bits for Packet 0

Bits	Description
0–1	Reserved. These bits should be cleared for compatibility with future PowerPC microprocessors.
2	Key bit—either SR[Kp] or SR[Ks]. Kp indicates user-level access and Ks indicate supervisor-level access. The processor multiplexes the correct key bit into this position according to the operating context.
3–27	Address bits 3–27 correspond to bits 3–27 of the selected segment register. A[3–11] form the receiver tag (BUID). Software must initialize these SR bits to the ID of the BUC to be addressed. The 601 supports an additional mode, memory-forced direct-store mode, not defined by the PowerPC architecture, and dependent on the value of BUID. See Section 7.6, “Memory-Forced Direct-Store Interface (PowerPC 601 Processor Only).”
28–31	PID (sender tag)—Allows a maximum of 16 processor IDs to be defined for a given system. If more bits are needed for a very large multiprocessor system, the L2 cache (or equivalent logic) can append a larger processor tag. The BUC addressed by the receiver tag should latch the sender address required by the subsequent I/O reply operation. The 601 and 604 PID comes from PID [28–31]. The 603 PID is always driven as 0b0000.

7.1.2 Packet 1

The second address beat, packet 1, transfers byte counts and the physical address for the transaction, as shown in Figure 7-3.

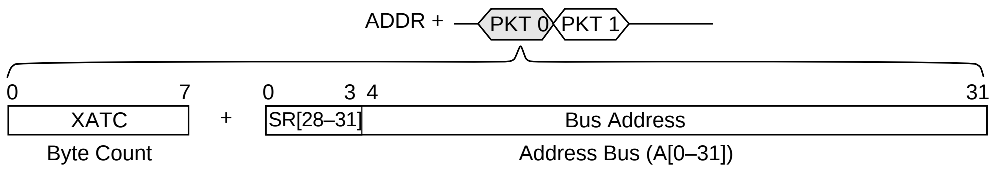

Figure 7-3. Direct-Store Operation—Packet 1

For packet 1, the XATC is defined as follows:

- Load request operations—XATC contains the total number of bytes to be transferred (128 bytes maximum for the 601, 603, and 604).
- Immediate/last (load or store) operations—XATC contains the current transfer byte count (1 to 4 bytes).

The processor gives the physical address, A[0–31], a concatenation of SR[28–31] with EA[4–31], to the BUC, which must keep a valid address pointer for the reply.

7.1.3 I/O Reply Operations

BUCs respond to direct-store transactions with an I/O reply operation, shown in Figure 7-4, which informs the processor of the success or failure of the operation. This requires a system to have bus mastership capability—a substantially more complex design task than bus slave implementations that use memory-mapped I/O access. Replies from the BUC to the processor are address-only transactions. As with packet 0 of the address bus on direct-store operations, the XATC has the transfer code (see Table 7-4). Additionally, an I/O reply operation transfers the sender/receiver tags in the first beat.

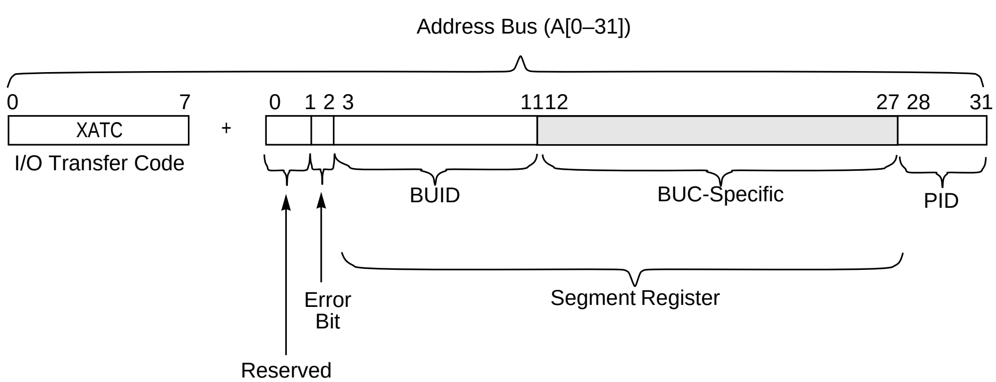

Figure 7-4. I/O Reply Operation

The address bits are described in Table 7-2.

Table 7-2. Address Bits for I/O Reply Operations

Bits	Description
0–1	Reserved. These bits should be cleared for compatibility with future PowerPC microprocessors.
2	Error bit. It is set if the BUC records an error in the access.
3–11	BUID. Sender tag of a reply operation. Corresponds with bits 3–11 of one of the segment registers.
12–27	Address bits 12–27 are BUC-specific and are ignored by the processor.
28–31	PID (receiver tag). The processor effectively snoops operations on the bus and, on reply operations, compares this field to PID[28–31] (601 and 604) to determine if it should recognize this I/O reply.

The second beat of the address bus is reserved; the XATC and address buses should be driven to zero to preserve compatibility with future protocol enhancements.

The following sequence occurs when a processor detects an error bit set on an I/O reply:

1. The processor completes the instruction that initiated the access.
2. If the instruction is a load, the data is forwarded onto the register file(s)/sequencer.
3. A direct-store error exception is generated, which transfers processor control to the direct-store error exception handler to recover from the error.

If the error bit is not set, the instruction that caused the access completes and instruction execution resumes. System designers should note the following:

- On the 601 and 603, reply operations that match the processor tag but arrive unexpectedly cause a checkstop condition. The 604 ignores these operations.
- External logic must assert $\overline{\text{AACK}}$ input for the processor, even though it is the receiver of the reply operation.
- The processor monitors address parity when enabled by software and $\overline{\text{XATS}}$ and reply operations (load or store).

7.2 Direct-Store Operations

Table 7-3 lists the type of bus operations supported by the direct-store interface.

Table 7-3. Direct-Store Bus Operations

Operation	Type	Direction
Load request	Address only	60x --> I/O
Load immediate	Address/data	60x --> I/O
Load last	Address/data	60x --> I/O
Store immediate	Address/data	60x --> I/O
Store last	Address/data	60x --> I/O
Load reply	Address only	I/O --> 60x
Store reply	Address only	I/O --> 60x

Table 7-3 shows the seven direct-store operations defined by the 60x. A single load or store instruction to a direct-store segment generates one or more direct-store operations (two or more direct-store operations for loads) from the 60x and one reply operation from the addressed device. For the first address beat, the XATC contains the direct-store transfer code shown in Table 7-4. The XATC is formed as follows:

XATC = TT0–TT3 || $\overline{\text{TBST}}$ || TSIZ0–TSIZ2

TT4 is not used. Definitions for these signals are irrelevant to direct-store transfers.

Table 7-4. Extended Address Transfer Code Definitions

Operation	TT[0–3]	$\overline{\text{TBST}}$	TSIZ[0–2]
Load request	0 1 0 0	0	0 0 0
Load immediate	0 1 0 1	0	0 0 0
Load last	0 1 1 1	0	0 0 0
Store immediate	0 0 0 1	0	0 0 0
Store last	0 0 1 1	0	0 0 0
Load reply	1 1 0 0	0	0 0 0
Store reply	1 0 0 0	0	0 0 0

Note: The values in the $\overline{\text{TBST}}$ column are the logical values seen on the signal.

7.3 Store Operations

Store operations are defined for the 60x as follows:

- Store immediate and store last operations transfer up to 32 bits of data to the device.
- A store reply from the slave device indicates the success or failure of that access.

A direct-store access consists of one or more data transfer operations followed by a store reply operation from the slave device. If the data can be transferred in one 32-bit data transaction, it is marked as a store last operation followed by the store reply operation; no store immediate operation is involved in the transfer, shown in the following:

STORE LAST (from 60x).....STORE REPLY (from slave device)

If more data is involved, there is one or more store immediate operations. The slave device detects the last transfer by looking for the store last transfer code, shown in the following:

STORE IMMEDIATE(s).....STORE LAST....STORE REPLY

7.4 Load Operations

Direct-store load accesses are like stores, except that the 60x receives instead of transmits data. As with basic transfer protocol, the 60x is master on both load and store operations. The system must grant the data bus to the 60x when the device is ready to provide data. Direct-store load requests have no analogous store operation; these address-only operations inform the addressed device of the number of bytes required on the subsequent load immediate/load last operations. The simplest, 32-bit or less, direct-store load is as follows:

LOAD REQUEST.....LOAD LAST.....LOAD REPLY (from slave device)

If more data is involved, there is one or more load immediate operations. The device detects the last data transfer by looking for the load last transfer code, shown in the following:

LOAD REQUEST.....LOAD IMMEDIATE(s).....LOAD LAST.....LOAD REPLY

Three of the seven defined operations are address-only transactions, which like basic transfer protocol, do not use the data bus. Unlike the basic transfer protocol, however, these transactions are not broadcast from one master to all snooping devices; rather, they pass control information between the processor and a specific slave device.

7.5 Direct-Store Operation Timing

The figures in this section show timings for typical load and store accesses to direct-store segments. All arbitration signals except for $\overline{\text{ABB}}$ and $\overline{\text{DBB}}$ have been omitted for clarity. Note that for either case, the number of immediate operations depends on the amount of data to be transferred. If fewer than four bytes of data are transferred and the data does not straddle a double-word address, there is no immediate operation. The 60x can transfer up to 128 bytes of data with a load or store instruction.

Figure 7-5 shows $\overline{\text{XATS}}$ asserted with the same timing as $\overline{\text{TS}}$ in basic transfer protocol. However, the address bus (and XATC) change on the next bus cycle. The first beat of the two-beat address bus operation is valid for one bus cycle window only, as defined by the assertion of $\overline{\text{XATS}}$ and cannot be extended. Address bus beat two can be extended by delaying assertion of $\overline{\text{AACK}}$ until the system latches the address.

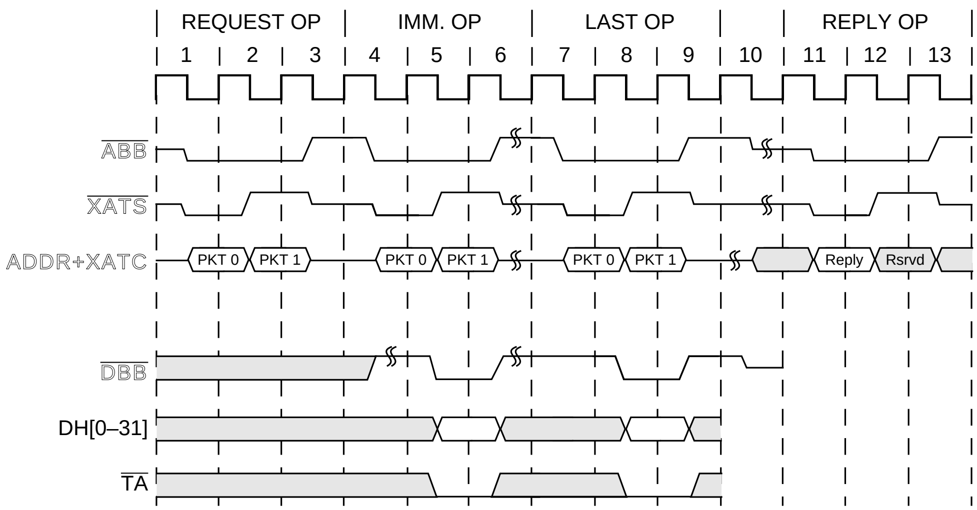

Figure 7-5. Direct-Store Interface Load Access Example

The load request and load reply operations in Figure 7-5 are address-only. Other types of bus operations can occur between individual direct-store operations on the bus. In this best-case example (no wait states), up to eight bytes of data are transferred in 13 bus cycles.

Figure 7-6 shows a store operation to a direct-store segment, consisting of a store immediate, a store last, and a store reply. Data is transferred on DH[0–31]. Unlike the load case, there is no request operation because the 60x has the data ready for the slave device.

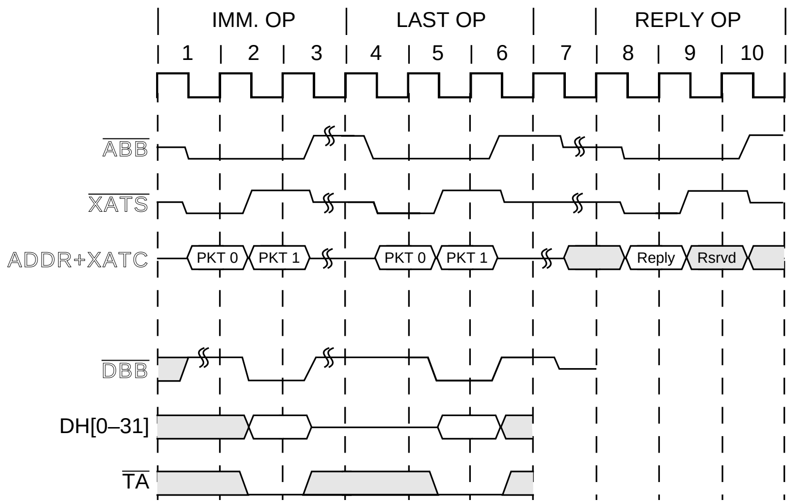

Figure 7-6. Direct-Store Interface Store Access Example

If $\overline{\text{TEA}}$ is asserted during a direct-store access, the resulting action is delayed until all data transfers from the direct-store access complete. The device asserting $\overline{\text{TEA}}$ must keep it asserted until the last direct-store data tenure is complete. The direct-store reply, in cases of $\overline{\text{TEA}}$ assertion, is not required and is ignored by the processor. The processor does not recognize the assertion of $\overline{\text{TEA}}$ until the last direct-store data tenure completes.

7.6 Memory-Forced Direct-Store Interface (PowerPC 601 Processor Only)

The 601 defines two types of direct-store segments (segment register T bit set) based on the value of the BUID, as follows:

- Direct-store interface (BUID ≠ 0x07F)—Normal direct-store accesses include all transactions between the 601 and BUCs mapped through direct-store address space.
- Memory-forced direct-store interface (BUID = 0x07F)—Memory-forced direct-store interface operations access memory space. They do not use the extensions to the memory protocol described for direct-store accesses, and they bypass the page- and block-translation and protection mechanisms. The physical address is found by concatenating bits 28–31 of the respective segment register with bits 4–31 of the effective address. This address is marked noncacheable, write-through, and global.

  Because memory-forced direct-store accesses address memory space, they are subject to the same coherency control as other memory reference operations. More generally, accesses to memory-forced direct-store segments are considered to be cache-inhibited, write-through, and memory-coherent operations with respect to the 601 cache and bus interface.

Chapter 8
System Considerations

This chapter describes general considerations for system design with the 60x bus. The following topics are included:

- Arbitration
- Write data reordering
- $\overline{\text{AACK}}$ generation
- Use of **sync** and **tlbsync**
- Pull-up resistors
- Features for improved bus performance
- IEEE 1149.1-compliant interface
- Using $\overline{\text{DBWO}}$
- **lwarx**/**stwcx.** considerations

8.1 Arbitration

Depending on the system implementation, the system arbiter may have various functions. As a minimum, it performs arbitration for access to the address bus and grants access to the data bus. It connects to each bus master with at least three unique signals; two for address bus control, bus request ($\overline{\text{BR}}$) and bus grant ($\overline{\text{BG}}$), and one for data bus granting, $\overline{\text{DBG}}$.

Apart from negating bus requests the cycle after $\overline{\text{ARTRY}}$ is asserted, 60x bus protocol offers no inherent fairness in determining bus mastership. Therefore system designers must consider system needs as a whole when choosing an arbitration strategy.

8.2 Using the Data Bus Write-Only Mechanism

Some processors support a limited out-of-order capability for its own pipelined transactions through the data bus write only ($\overline{\text{DBWO}}$) signal. When the assertion of $\overline{\text{DBWO}}$ is recognized on the clock of a qualified data bus grant, the processor is directed to perform the next pending data write tenure (if any) even if a pending read tenure would have normally been performed. The $\overline{\text{DBWO}}$ signal only allows a write tenure to be performed ahead of a pending read tenure from the same processor, not another write tenure.

In general, an address tenure is followed immediately by its associated data tenure. Transactions pipelined by a processor complete in strict order except when the system uses DBWO to allow a processor to perform a snoop push-out operation (or other write transaction pending in the write queues) between the address and data tenures of a read operation. This effectively envelopes the write operation within the read operation. Figure 8-1 shows how DBWO supports enveloped write transactions.

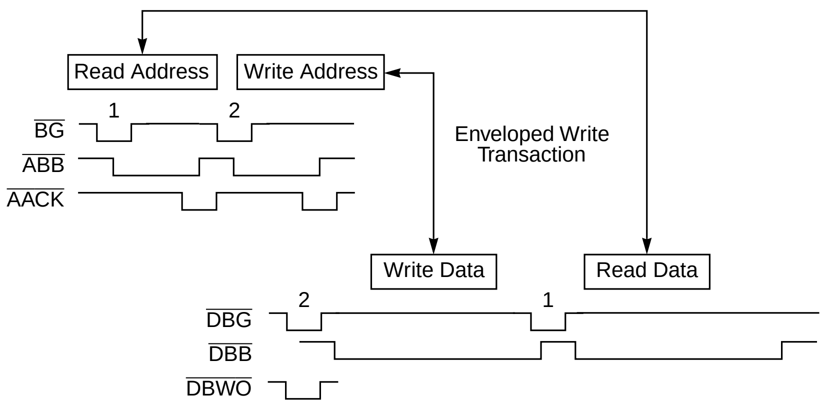

Figure 8-1. Data Bus Write Only Transaction

Care should be used when using the enveloped write feature. For systems that do not implement this capability, $\overline{\text{DBWO}}$ should remain negated. In systems where this capability is needed, $\overline{\text{DBWO}}$ should be asserted under the following scenario:

1. The processor initiates a read transaction (either single-beat or burst) by completing the read address tenure with no address retry ($\overline{\text{ARTRY}}$ negated).
2. Then, the processor initiates a write transaction by completing the write address tenure with no $\overline{\text{ARTRY}}$.
3. At this point, if $\overline{\text{DBWO}}$ is asserted with a qualified data bus grant to the processor, the processor 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 ends with the processor negating $\overline{\text{DBB}}$.
4. The next qualified data bus grant signals the processor 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 tried between any of these steps. Note the following regarding $\overline{\mathrm{DBWO}}$:

- The $\overline{\mathrm{DBWO}}$ signal can be asserted if no read operation is pending; it does not affect write ordering.
- Ordering and presence of data bus writes is determined by the writes in the write queues when $\overline{\mathrm{BG}}$ is asserted for the write address (not $\overline{\mathrm{DBG}}$). A snoop push-out operation has highest priority over other queued write operations.
- Because more than one write can be in the write queue when $\overline{\mathrm{DBG}}$ is asserted for the write address, more than one data bus write can be enveloped by a pending read.

The arbiter must monitor bus operations and coordinate masters and slaves with respect to the use of the data bus when $\overline{\mathrm{DBWO}}$ is used. Individual $\overline{\mathrm{DBG}}$ signals associated with each bus device should allow the arbiter to synchronize both pipelined and split-transaction bus organizations. Individual $\overline{\mathrm{DBG}}$ and $\overline{\mathrm{DBWO}}$ signals provide a primitive form of source-level tagging for the granting of the data bus.

The ability to perform a snoop push before completion of a read transaction that has been started by the processor prevents certain deadlock conditions. Consider a case where a 60x processor shares a bus with a memory controller and a bus converter. Assume that the bus converter produces an XYZ bus and that the following two requests appear simultaneously:

- A request from the processor on the 60x bus that requires an XYZ bus transaction.
- A request on the XYZ bus that should cause a data transfer with memory on the 60x.

The bus converter queues the processor request until the XYZ bus transaction completes. The XYZ bus transaction causes a request on the 60x bus, and, unfortunately, a snoop hit that requires a push. To avoid deadlock, this enveloped push must complete before the data transfer for the processor request. This is a problem with bus converters using certain styles of buses. In such cases, the system should assert $\overline{\mathrm{DBWO}}$ and $\overline{\mathrm{DBG}}$ together for write operations identified as snoop pushes, which may be a difficult determination because a system might assert this signal for all write data transfers, effectively reordering all write data ahead of outstanding reads.

The arbiter must monitor all bus operations in progress and synchronize masters and slaves with respect to the use of the data bus. Each master's $\overline{\mathrm{DBG}}$ allows the arbiter to synchronize pipelining and supports split transaction bus organizations.

8.3 $\overline{\text{AACK}}$ Generation

Systems can use the signals provided by 60x processors to implement a simple, single-envelope bus in which data and address tenures are always together. It can also implement a bus that provides limited pipelining, in which subsequent addresses are sent out before the completion of the current data transfer. It even allows creation of a bus that provides split address and data transfers. The degree to which each processor may support such operations depends on processor design, namely the depth and logic associated with the read and write buffers in the processor's bus interface unit (BIU).

The system designer must determine how $\overline{\text{AACK}}$ signals the completion of an address transfer and allows other address transfers to occur. Following are some possibilities:

- The system arbiter may assert $\overline{\text{AACK}}$ the cycle after it sees an asserted $\overline{\text{TS}}$. This allows requests to be placed onto the bus at the maximum rate of one every three cycles. System design must ensure that these requests do not exceed the rate at which slave devices can process them. One alternative is for the arbiter to limit the number of outstanding requests, using the bus grant mechanism.

  Another possibility would be to collect busy status from individual bus devices, and use this to pace the arbitration mechanism or to delay the $\overline{\text{AACK}}$ response.
- Individual devices might generate $\overline{\text{AACK}}$, based on their decode of the address. This approach is somewhat limited in performance, however. If the bus clock is slow, it might be permissible to latch the address, decode it, and then drive $\overline{\text{AACK}}$ on the next cycle. Note that it would be necessary to prevent false transitions of $\overline{\text{AACK}}$. For a system with a fast clock rate, devices would need to latch the address, take a cycle to decode it, and then issue $\overline{\text{AACK}}$ on the second cycle following $\overline{\text{TS}}$, or later.

The 60x processors do not provide a graceful way to recover from an operation that receives no $\overline{\text{AACK}}$; however, they perform all address checking that they are to perform before placing addresses on the bus. In general, a processor considers all requests complete when they are placed on the bus, and there is no recoverable error reporting on the bus.

8.4 SYNC vs. TLBSYNC and System Design

The 601 and 604 handle TLBIE, SYNC, and TLBSYNC bus operations differently. In a 601 system, TLBIE operations are followed by a SYNC operation. In a 604 system, TLBIE operations are followed by TLBSYNC operations, which may affect devices that maintain TLBs.

If processors perform the TLBIE operation immediately, or if no pending operations are queued, they may require no extra steps to ensure compatibility. However, if they maintain queues of pending operations, and these queues contain translated addresses, they may need to participate in the synchronization operation, and they may need to implement different modes for 601 and 604.

8.5 Pull-Up Resistors

The following signals are driven by various bus devices: $\overline{\text{TS}}$, $\overline{\text{XATS}}$, $\overline{\text{ABB}}$, $\overline{\text{TA}}$, $\overline{\text{DBB}}$, $\overline{\text{ARTRY}}$, $\overline{\text{SHD}}$, and $\overline{\text{DRTRY}}$. When control of these signals is passed from one device to another, the device that is releasing control always deasserts them before release. The signals are then left in high-impedance state before being driven by another device. Pull-up resistors are required on these signals to keep them in the negated state for this interval. Note that this can be a fairly high value resistor because it does not cause a transition and only retains a value.

8.6 Features for Improved Bus Performance

The following 60x processor features help improve bus performance:

- Disabling the $\overline{\text{DRTRY}}$ feature (603 and 604e) decreases read latency by one cycle.
- The use of $\overline{\text{ABB}}$ and $\overline{\text{DBB}}$ are optional on 60x processors. Because of the fractional-cycle restoration to the high state, this helps achieve shorter cycle time.
- Consecutive read data transfers can be sent without a dead cycle (604). This increases maximum bandwidth by 25%.

8.7 IEEE 1149.1-Compliant Interface

The 604 boundary-scan interface is a fully-compliant implementation of the IEEE 1149.1 standard. This section describes the 604 IEEE 1149.1(JTAG) interface.

8.7.1 IEEE 1149.1 Interface Description

Table 8-1 describes the 604's five dedicated JTAG signals. The TDI and TDO scan ports are used to scan instructions and 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 TMS input sequence. Scan data is latched at the rising edge of TCK.

Table 8-1. IEEE Interface Signal Descriptions

Signal Name	Input/Output	Weak Pullup Provided	IEEE 1149.1 Function
TDI	Input	Yes	Serial scan input signal
TDO	Output	No	Serial scan output signal
TMS	Input	Yes	TAP controller mode signal
TCK	Input	Yes	Scan clock
$\overline{\text{TRST}}$	Input	Yes	TAP controller reset

$\overline{\text{TRST}}$ is a JTAG-optional signal used to reset the TAP controller asynchronously; it ensures that JTAG logic does not interfere with normal chip operation. It can be asserted coincident with $\overline{\text{HRESET}}$.

8.8 lwarx/stwcx. Considerations

The **lwarx** and **stwcx.** instructions are used to synchronize multiple processors. Operation of these instructions is described in the following sections.

8.8.1 Coherency Participation

This section describes the 604 MESI coherency mechanism. There are three legal WIM encodings that define coherency-required regions:

- x11—Noncacheable
- 001—Write-back
- 101—Write-through

This discussion assumes that any semaphore (the address used for an **lwarx/stwcx.** operation) addressed by different processors, has the same WIM encodings regardless of which processor accesses it. Additionally, some of the discussion of write-back cacheable and write-through cacheable are combined as they have similar requirements.

8.8.1.1 Noncacheable Reservations

Regardless of whether they are associated with reservations, load and store operations to noncacheable semaphores must access main memory. Loads for noncacheable semaphores occur as read atomic bus operations. Typically, noncacheable writes (write-with-flush operations) can be buffered at various stages. However, these writes must be broadcast to all processors holding reservations so they can be compared against reservation addresses.

Note that this is not strictly true. If a memory system implemented a directory of reservations (entry per processor), it would need only direct noncacheable writes to the appropriate processor when a match is detected. It could directly reset the reservation if another input signal existed, although there is not one present on the 604.

Because it appears to be required for memory coherence for these writes to be broadcast (rather than for reservation reasons), it can be assumed that **lwarx/stwcx.** will follow this requirement. Noncacheable writes would not be required to be broadcast, if no processor could have a cached copy of the data. This is not specified by the PowerPC architecture.

Snooping by the processor for write-with-flush (normal and atomic) operations to the reservation address must begin as soon as the **lwarx** address is acknowledged. This is because a write to that address can occur between the address and data phase of an **lwarx** instruction. If a snooped write operation matches, the reservation is cleared. Snooping for writes by an L2 cache must begin as soon as the **lwarx** address is acknowledged on its system bus. L2 snoop filtering for reservations may be simple or complex (see Section 8.8.2, “Filtering Options for Reservations,” for alternatives). In either case, snooping must be able to start as soon as the L2 system bus side sees an acknowledgment, which may constrain when the processor can assert $\overline{\text{RSRV}}$.

The **stwcx.** instruction cannot be allowed to complete until the operation (write with flush atomic) gains access to main memory. This requires any L2 cache to delay acknowledgment of completion of the operation until it is globally performed. Buffering cannot be provided unless it is after the completion point with respect to main memory.

8.8.1.2 Cacheable Reservations

If a read to a cacheable semaphore misses, it is fetched with a read atomic bus operation. This places the data in the cache as S or E, depending upon the state of the $\overline{\text{SHD}}$ signal. The read may hit in the cache with states M, E, or S for write-back cacheable space, or E or S for write-through cacheable space. It is recommended that the processor notify the external world of the address of the reservation when setting a reservation on an address in the cache. See Section 8.8.2, “Filtering Options for Reservations.”

8.8.1.3 Read Snooping Requirements

A processor with a reservation on a cacheable semaphore must ensure that any subsequent reads (both read and read atomic) by any other processor do not take the address into their cache in the exclusive state (E). This prevents semaphores that are write-back, cacheable from being modified by a write that is invisible to the processor holding the reservation (that is, going from the E state to M state within the other cache). For the MESI protocol used by the 604, this involves asserting $\overline{\text{SHD}}$ whenever another processor executes a read to the reservation address. This assertion of $\overline{\text{SHD}}$ for reservation purposes is independent of whether the data associated with the address is in the cache.

This requirement also extends to an L2 cache. If a reservation is held, it must selectively or freely assert $\overline{\text{SYS\_SHD}}$ for read operations that may occur to the semaphore address. See Section 8.8.2, “Filtering Options for Reservations.”

8.8.1.4 Write-Back Reservation-Canceling Snoops

In addition to snooping to ensure that reads do not take exclusive ownership of a reservation address, the processor must also snoop for operations that would cancel the reservation. This snooping is in addition to that required to maintain cache coherency. The following operations cancel a reservation held on a semaphore that is write-back cacheable as they involve transfer of ownership of the address to another processor:

- RWITM—another processor gains ownership before completing a store
- RWITM atomic—another processor gains ownership before completing an **stwcx.**
- Kill block—another processor stores into a shared block or a **dcbz** instruction is executed

A write-with-kill operation cannot occur since it would imply that another processor has gained ownership, in which case a reservation would have been lost.

8.8.1.5 Write-Through Reservation-Canceling Snoops

The following operations cancel a reservation held on a semaphore that is write-through cacheable as they involve transfer of ownership of the address to some other processor:

- RWITM—another processor gains ownership before completing a store.
- RWITM atomic—another processor gains ownership before completing an **stwcx.**.
- Write and flush—another processor stores into a shared block.

Because an address can be treated as both write-through and write-back by different processors, both of the previous sets of operations should be snooped for clearing reservations.

8.8.1.6 Noncanceling Bus Operations

Because the following bus operations do not transfer ownership, they do not cancel the reservation to another processor regardless of the effect they may have on the state of the data in the cache:

- Clean block—another processor executing **dcbst**
- Flush block—another processor executing **dcbf**

These operations can be viewed as transferring ownership back to main memory. They are often followed by an attempt to gain ownership, but these operations in themselves do not transfer ownership to another processor or cancel the reservation:

8.8.2 Filtering Options for Reservations

An L2 cache must also participate in bus operations to ensure correct operation of reservations. There is a range of options for filtering reservations. The following sections describe two much different approaches and the hardware required for each. This section assumes that bus operations are passed on to support reservations; clearly an operation may be passed either for supporting reservations, coherency, or both.

8.8.2.1 Minimal Reservation Support

The simplest approach to reservation filtering relies only on an indication that a reservation exists; for example assertion of the $\overline{\text{RSRV}}$ signal. In this case, when no reservation is indicated by the processor ($\overline{\text{RSRV}}$ negated) no reservation-influencing operation (or read-influenced operation, for example, read/read atomic operations that might need to have $\overline{\text{SYS\_SHD}}$ asserted) need to be passed on to the processor. When a reservation is held by the processor ($\overline{\text{RSRV}}$ asserted) all reservation-influencing operations are passed on to the processor. All reservation-influenced operations are responded to with $\overline{\text{SYS\_SHD}}$ asserted.

This approach requires no state in the L2 (or higher level cache) and relies upon simply gathering the various reservation indications at any level and passing them down as a unified signal to lower levels to allow reservation influencing operations to propagate back up the tree where they can be selected by a branch that is interested in such operations. While the hardware requirements are simple in this approach, performance is affected in the following ways:

- The available system bus bandwidth is reduced while operations are retried pending a response from the top of the tree.
- Intermediate buses are tied up so other processors cannot have higher level misses serviced.
- Many read operations are cached as shared instead of exclusive, which generates unnecessary bus traffic later when stores are performed to those addresses. For a large multiprocessor system, this could cause significant loss in total bandwidth.

Implementation of this scheme only requires timely assertion of the $\overline{\text{RSRV}}$ signal by a processor. If $\overline{\text{RSRV}}$ were asserted by the end of the cycle after $\overline{\text{AACK}}$ assertion for operations where a read-atomic operation is required, or the next three-state after setting the reservation for a cache hit, then there is adequate time for the L2 controller to prepare for future system bus operations.

8.8.2.2 Improved Reservation Snooping

A more hardware-intensive approach to filtering is to require L2 caches to contain registers and comparators for the address associated with a specific processor’s reservation. The controller only passes on reservation-modifying cycles from the system bus side to the processor bus side and can participate directly in reservation-influenced cycles. Thus, only those addresses with actual outstanding reservations causes accesses to be retried on the system bus, intermediate buses being unavailable, and placed in other caches as shared only when necessary to maintain a reservation.

To provide this level of support, a processor must always ensure lower levels can snoop addresses on which a reservation is placed. In the case of either noncacheable or cacheable miss operations, the address is transmitted during the read-atomic operation that acquires the data. For cacheable snoop hits, an address-only bus operation should be performed, to allow the reservation address to be passed cleanly from the processor to any L2 caches.

This additional bus operation type is proposed since there are problems in using the current read-atomic operation in the face of a cache hit. While there are many ways of trying to use the current data transferring read-atomic operation, there are problems with both the L1 and higher level caches dealing with the case of modified data already resident in the cache. For these reasons it is cleaner to require a new bus operation type which would transmit a reservation address down from one level in the hierarchy to the next below.

Additionally, the reservation address needs to be cleared so higher levels of the memory hierarchy can stop snooping for reservations. This **stwcx.** address-only operation is an optimization, and is not required. The cost of not clearing the reservation address is that a small amount of unnecessary snoop operations is sent up the memory hierarchy to the processor assumed to be holding the reservation and a small amount of system bus bandwidth is lost to unnecessary retries.

8.8.2.3 lwarx/stwcx. Address-Only Operation

An **lwarx/stwcx.** address-only operation should meet several criteria. Most importantly, it should not cause abnormal system behavior in systems designed around the 601 and only sampling TT[0–4]. For this reason they have been mapped to operations such as clean block and flush block that are innocuous from a system perspective. This yields the TT[0–4] encodings shown in Table 8-2.

Table 8-2. Transfer Type Settings for lwarx/stwcx. Address-Only Operation

TT[0–4]	Cycle
00001	Set **lwarx** address.
00010	Clear reservation address.

8.8.2.4 Software Implications

Bus traffic should be considered when system software deals with semaphores. Noncacheable semaphores incur no additional overhead because all **lwarx/stwcx.** operations are broadcast anyway. However, if the semaphore was in the cache, cacheable semaphores may cause additional address-only bus cycles for each **lwarx** instruction executed. Likewise, write-back, cacheable semaphores may cause additional address-only bus cycles for each **stwcx.** operation. This small overhead may dictate some software considerations if **lwarx/stwcx.** are used frequently. For example, to reduce bus bandwidth for heavily-used semaphores, something like the following test and test and set operation may be needed:

```
loop:   ld      rn,S
        cmpi    rn, VAL
        bcc     loop
        lwarx   rn,S
                ops as required
        stwcx.  rm,S
        bne     loop.
```

The preceding operation may be more useful than the following test and set operation:

```
loop:   lwarx   rn,S
        ops as required
        stwcx.  rm,S
        bne     loop.
```

Appendix A
Processor Summary

This section provides an overview of the different functionality of the PowerPC 601, 603 and 604 processors. The 603 supports coherent memory, but does not explicitly support multiprocessors or L2 caches. The 601 and 604 support both multiprocessing configurations and L2 caches. Table A-1 summarizes differences in bus and memory coherency behavior between the 60x processors.

Table A-1. Bus and Memory Coherency Behavior Summary

Functionality	601	603	604
Cache set element (bits)	3	1	2
Linefill strategy	Critical quad word	Critical double word	Critical double word
Cache coherency protocol	MESI	MEI	MESI
Broadcast cache operations	Yes	No	Yes
TLBI on bus	Yes	No	Yes
TLBISYNC	SYNC bus operation	$\overline{\text{TLBISYNC}}$ input signal	TLBSYNC bus operation
ICBI on bus	N/A	No	Yes (extra TT operation)
EIEIO on bus	SYNC bus operation	No	EIEIO bus operation
No-$\overline{\text{DRTRY}}$ mode	No	No-$\overline{\text{DRTRY}}$ mode	No-$\overline{\text{DRTRY}}$/data streaming mode
Clocking	Direct, with phase inputs (PCLK_EN, BCLK_EN)	PLL	PLL
Window of opportunity usage	Varies	Snoop push only	Snoop push only
Snoop push buffering	Varies	Dedicated	Dedicated
Fast push after $\overline{\text{ARTRY}}$ if parked	Yes	No	No
High priority push (input signal)	Yes	No	No
Read with no intent to cache	No	Yes	Yes
Timing of $\overline{\text{ARTRY}}$/$\overline{\text{SHD}}$ restore (see notes at end of signal tables)	A	B	B
Broadcast **lwarx** indicator on cache hit	No	No	Yes

Table A-1. Bus and Memory Coherency Behavior Summary (Continued)

Functionality	601	603	604
Slight differences in TC encodings	—	—	—
Data bus disable signal	No	Yes	Yes
$\overline{\text{TA}}$ required during last $\overline{\text{DRTRY}}$	No	Yes	Yes
Snoop response signals	$\overline{\text{ARTRY}}$, $\overline{\text{SHD}}$	$\overline{\text{ARTRY}}$	$\overline{\text{ARTRY}}$, $\overline{\text{SHD}}$
Misaligned within double word that crosses word causes two accesses	No	Yes	Yes
Time base source	RTC input	System clock	System clock
Time base enable	RTC	TBEN	TBEN
Power management	—	Several modes	Nap mode
Power management signals	—	$\overline{\text{QREQ}}$, $\overline{\text{QACK}}$	RUN, HALTED
$\overline{\text{DBWO}}$ only with pending read	Nothing	Transfer read	Transfer read
Data mirroring on $\overline{\text{CI}}$ writes	Yes	No	Unspecified
Early $\overline{\text{ARTRY}}$ data tenure termination	No	Yes	Yes
Processor ID value in PIO	From PID register	Zero	From PID register
Snoop for misplaced PIO reply	Yes	No	No
32-bit data transfer mode	No	Yes	No
Reduced pinout mode	No	Yes	No
$\overline{\text{CKSTP\_OUT}}$ asserted, outputs high impedance for $\overline{\text{CKSTP\_IN}}$ assertion	No	Yes	Yes
Optional machine check for checkstop condition	No	Yes	Yes
Cancel reservation on snooped RWITM	Yes	No	Yes
Snoop nonglobal transactions for reservation cancellation	No	Yes	No
stwcx. treated as write-through	No	Yes	No
$\overline{\text{WT}}$ state on snoop push	See Section 2.4.7, “Transfer Code (TCn)—Output”	See Section 2.4.7, “Transfer Code (TCn)—Output”	See Section 2.4.7, “Transfer Code (TCn)—Output”

Differences in programming model (for example, implementation-specific special-purpose registers) are described in the user’s manual for each device.

Appendix B
Processor Clocking Overview

This appendix provides a short overview of clocking on the PowerPC 60x processors. Detailed information is provided in each processor's user's manual and hardware specifications.

B.1 PowerPC 601 Microprocessor Clocking

The 601 requires an input clock, 2X_PCLK, which operates at twice the processor rate. In addition, it requires the $\overline{\text{PCLK\_EN}}$ signal, which defines the phase of the internal processor clock, and the $\overline{\text{BCLK\_EN}}$ signal, which likewise determines the phase of the internal bus clock, both relative to positive edges of the input clock. Figure B-1 illustrates the clocks for the 601, with the bus clock enable selected to run at half the processor frequency.

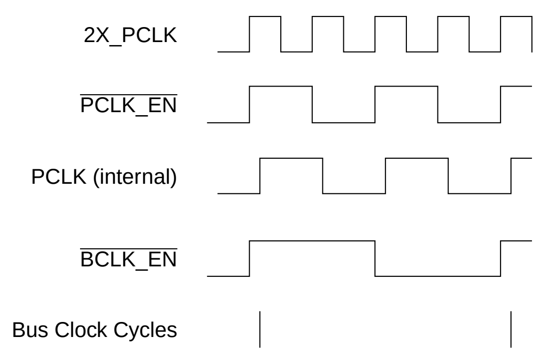

Figure B-1. PowerPC 601 Processor Clocking

See the *PowerPC 601 RISC Microprocessor User's Manual* and *PowerPC 601 RISC Microprocessor Hardware Specifications* for more information.

B.2 PowerPC 603 and PowerPC 604 Microprocessor Clocking

The 603 and 604 clocks are derived by internal phase-locked loops (PLL) which lock onto the positive edge of the bus clock input. In a given system, it is required that the bus clock operate at a constant frequency, so the PLL can maintain its lock.

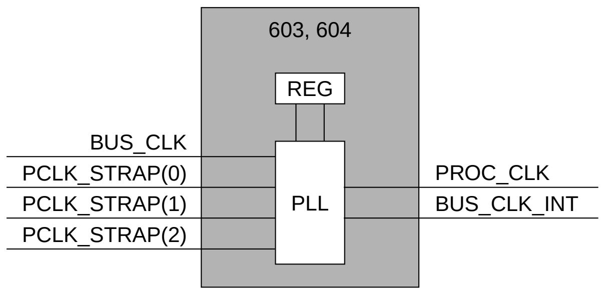

Figure B-2. PowerPC 603 and PowerPC 604 Processor Clock Generation

Clock selection inputs on the chips are sampled at reset to determine the clock ratio at which the part operates. The condition of these inputs also programs the VCO to operate within its proper range. The 603 and 604 can operate with a variety of bus-and-processor clock frequency ratios. This functionality is described generally in the user’s manuals and more specifically in the hardware specifications.

Appendix C
Processor Upgrade Suggestions

This appendix provides upgrade suggestions for the PowerPC 601, PowerPC 603, and PowerPC 604 processors.

C.1 PowerPC 601 Processor Upgrade to 60x

The recommended approach to disable the 601 when a 60x processor is plugged into an upgrade socket is as follows:

- Apply $\overline{\text{HRESET}}$ for at least 300 processor clock cycles. This causes all outputs to be placed in high-impedance state and put 0s in all internal latches and registers. The $\overline{\text{HRESET}}$ signal can be held active longer if desired, but the minimum is 300 cycles.
- Deactivate the 2X_PCLK or $\overline{\text{PCLK\_EN}}$. This stops the processor and minimize any dynamic power.
- Hold TST16 low to ensure that all OCDs remain in a high-impedance state.

All other test signals should be connected as specified in the *PowerPC 601 RISC Microprocessor Hardware Specifications*.

C.2 PowerPC 603 Processor Upgrade to 604 or 60x

Figure C-1 illustrates the recommended connection that allows a 603 to upgrade to a 604 (and potentially other future 60x chips).

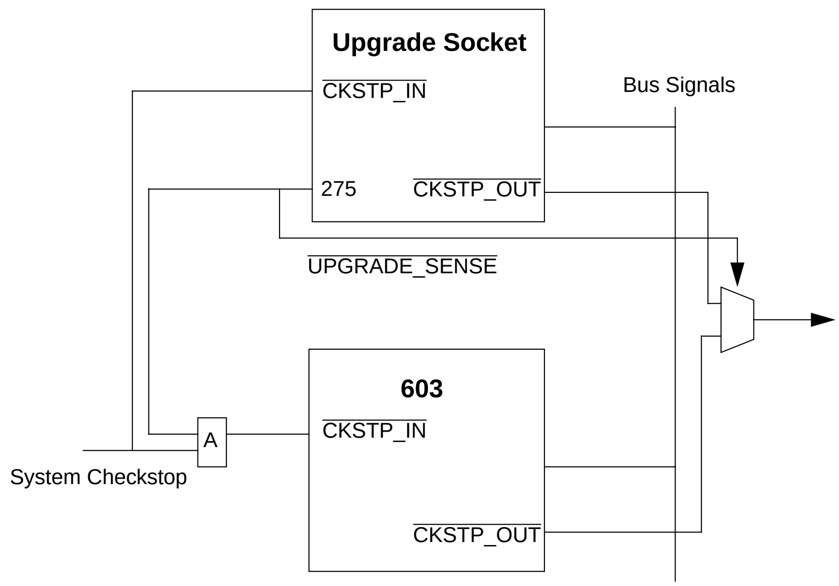

Figure C-1. PowerPC 603 to PowerPC 604 Processor Upgrade Option

A designer should consider the following:

- The upgrade socket has 604 pinout.
- Pin 275 is ordinarily an OGND pin. Instead of connecting it to ground, use it as negative true output $\overline{\text{UPGRADE\_SENSE}}$ to indicate presence of upgrade processor.
- Put a pull-up resistor on $\overline{\text{UPGRADE\_SENSE}}$.
- On BGA module, use any ground as upgrade sense.
- If the system does not use $\overline{\text{CKSTP\_IN}}$, use a pull-up resister. A gate is not required.
- If the system does not sample $\overline{\text{CKSTP\_OUT}}$, multiplexing is unnecessary.
- Connect in parallel: $\overline{\text{BR}}$, $\overline{\text{BG}}$, $\overline{\text{TS}}$, $\overline{\text{XATS}}$, $\overline{\text{ABB}}$, A[0–31], $\overline{\text{AACK}}$, AP*n*, $\overline{\text{APE}}$, TT*n*, TC*n*, TSIZ*n*, $\overline{\text{TBST}}$, $\overline{\text{CI}}$, $\overline{\text{WT}}$, $\overline{\text{GBL}}$, $\overline{\text{SHD}}$, $\overline{\text{ARTRY}}$, $\overline{\text{DBG}}$, $\overline{\text{DBWO}}$, $\overline{\text{DBB}}$, DH, DL, $\overline{\text{DBDIS}}$, DP*n*, $\overline{\text{DPE}}$, $\overline{\text{TA}}$, $\overline{\text{DRTRY}}$, $\overline{\text{TEA}}$, $\overline{\text{INT}}$, $\overline{\text{SMI}}$, $\overline{\text{MCP}}$, $\overline{\text{SRESET}}$, $\overline{\text{HRESET}}$, $\overline{\text{RSRV}}$, TBEN, and SYSCLK.
- Upgrade socket provides RUN input. The 603 has no equivalent function.
- Upgrade socket provides HALTED output. The 603 has no equivalent function.
- 603 has $\overline{\text{QREQ}}$ and $\overline{\text{QACK}}$. There are no corresponding signals on upgrade.
- No connection necessary on CLKOUT (test only).

- CKSTP_IN is referred to as CKSTP on earlier versions of the 603. Likewise, CKSTP_OUT is referred to as CHECKSTOP.
- Processors may have different strappings on PLL_CFG. Programmability on the upgrade socket is recommended.
- ANALOG VDD inputs on the two processors should each have dedicated filter network.
- Connection of TRST, TDI, TDO, TMS, and TCK is a function of system JTAG testing requirements.
- Pull-up L1_TEST_CLK, L2_TEST_CLK, and LSSD_MODE on the 603 and L1_TEST_CLK, L2_TEST_CLK, LSSD_MODE, and ARRAY_WR on the upgrade socket.

C.3 PowerPC 604 Processor Upgrade to 60x

The following describes the recommended connection to provide for upgrades from 604 to future processors:

- Pin 275 is ordinarily an OGND signal. Instead of connecting it to ground, use it as negative true output UPGRADE_SENSE to indicate presence of upgrade processor.
- Put a pullup resistor on UPGRADE_SENSE.
- On BGA module, use any ground as upgrade sense
- If the system does not use the CKSTP_IN input, use a pull-up resister. A gate is not required.
- Connect the following signals in parallel: BR, BG, TS, XATS, ABB, A[0–31], AACK, AP*n*, APE, TT*n*, TC*n*, TSIZ*n*, TBST, CI, WT, GBL, SHD, ARTRY, DBG, DBWO, DBB, DH, DL, DBDIS, DP*n*, DPE, TA, DRTRY, TEA, INT, SMI, MCP, SRESET, HRESET, RSRV, TBEN, and SYSCLK.
- No connection necessary for CLKOUT (test only).
- Processors may have different strappings on PLL_CFG. Programmability on upgrade socket is recommended.
- ANALOG VDD inputs on the two processors should each have dedicated filter network.
- Connection of TRST, TDI, TDO, TMS, and TCK is a function of system JTAG testing requirements.
- Pull up L1_TEST_CLK, L2_TEST_CLK, LSSD_MODE, and ARRAY_WR on both processors.

Appendix D
L2 Considerations for the PowerPC 604 Processor

This L2 cache reduces the average memory access time for each processor and partitions bus traffic between the various buses. In addition to keeping most of the individual processor bus traffic off of the system bus, this arrangement can screen memory coherency snoop traffic, keeping it off of individual processor buses. This section discusses the use of an L2 cache controller in a system configuration shown in Figure D-1.

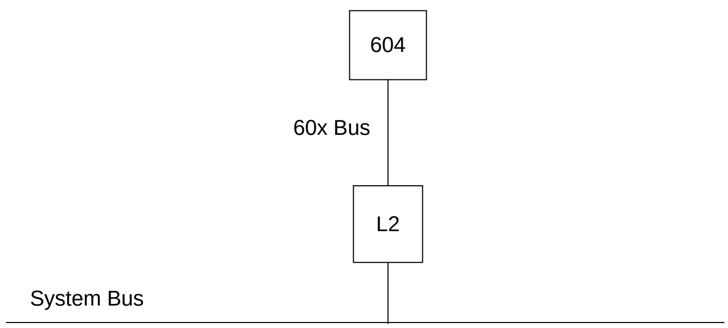

Figure D-1. L2 Cache Controller Organization

The system bus may use a 60x bus or a bus of some other design. Methods of designing a system with an L2 cache are as follows:

- No snoop filtering—The simplest approach to an L2 system design is to not filter snoop activity. This is not practical for multiprocessor systems.
- Keeping a copy of L1 tags—Keeping a copy of the L1 tags in the L2 cache allows a system address to be compared against the L1 tags and the L2 tags in parallel. If neither directory matches, the processor/L2 cache complex is not involved in the current bus transaction and does not need to intervene in the operation. Typically, intervention implies assertion of either $\overline{\text{SYS-ARTRY}}$ or $\overline{\text{SYS-SHD}}$.
- Maintaining L1 state and tags—Keeping a copy of the L1 tags and recording whether the cache block is in the S or E state allows the L2 cache to filter more snoop traffic from the processor than by saving the L1 state alone.

- Simple L1 inclusion—L1 inclusion requires that an address cannot be in the L1 cache unless it is in the L2 cache. Ensure that the contents of the L1 cache are a subset of the contents of the L2 cache.
- Marked L1 inclusion—In addition to guaranteeing inclusion, marked inclusion keeps more information about when an L2 cache entry is also in the L1 cache. The advantages of marked L1 inclusion over simple L1 inclusion include being able to do a better job of snoop filtering and reducing the amount of back invalidations.

For each of these approaches (except the simplest case of performing no snoop filtering), each description includes the following:

- Requirements for saving state information—Information about the kind and amount of state information that must be maintained.
- Operations required for processor bus operations—Information regarding operations that are necessary to maintain consistency between the L1 and L2 caches.
- System bus operation forwarding to the processor—A description of the system bus operations must be passed to the processor for each configuration.

Note that the prefix SYS distinguishes system bus signals from 60x signals with the same name. For example, the system bus counterpart to the 60x signal $\overline{\text{SHD}}$ is $\overline{\text{SYS-SHD}}$.

D.1 Unfiltered Snooping

The simplest way to design in an L2 cache is to not filter snoop operations. The L2 cache responds to snoop requests from the system bus after first passing the snoop request through to the L1 cache.This following design issues should be considered:

- If the processor's L1 cache has a second tag port dedicated to snooping, the processor is not stalled for unnecessary snoops. This is true for the 601 and 604 but not for the 603.
- The time the external address bus is busy with unnecessary snoops is not a significant portion of the address bandwidth required by the processor. A multiprocessor system cannot meet this condition practically; however, a single-processor system can meet the condition if DMA address bandwidth is low.

D.2 Keeping a Copy of L1 Tags

Keeping a copy of the L1 tags in the L2 cache allows a system address to be compared against the L1 tags and the L2 tags in parallel. If neither directory matches, the processor/L2 cache complex is not involved in the current bus transaction and does not need to intervene. Typically, intervention implies assertion of either $\overline{\text{SYS-ARTRY}}$ or $\overline{\text{SYS-SHD}}$.

If only the L2 tag matches, the L2 cache must intervene. If an L1 tag matches, the system address must be passed to the processor so it can respond. After the processor responds and completes any necessary snoop response, the system bus operation can be rerun against the possibly-changed state of the L2 cache.

D.2.1 Requirements for Saving State Information

This approach requires the implementation of a set of cache tags and comparators to maintain the addresses in the primary cache. For separate 16-Kbyte instruction and data caches, each cache directory must have 4 by 128 entries of a valid bit and 20 tag bits, for a combined total of 21 Kbits and eight 21-bit comparators. The valid bit is assumed to be implemented as a seven-transistor cell, as it should be cleared at power-up for repeatability and testing. Because this resource is a small fraction of the total tag required for a 1-Mbyte L2 (approximately 256 Kbits), it easily can be placed in the same controller. This tag array must be able to read a tag and valid bit, to write a tag and valid bit, and compare a tag and valid bit against an address (and assumed one valid bit).

A second state requirement is a set of registers with associated comparators per register (termed the copy-back address registers) to hold addresses displaced from the L1 tags that need snooping. One such register is needed for each copy-back buffer on the processor. These registers need only be able to write and compare. Reading them is unnecessary.

D.2.2 Operations Required for Processor Bus Operations

Apart from the memory required, the L2 cache must determine when a replacement operation has been performed by the L1 cache. This determination together with the cache set information allows the L2 to update the L1 directory copy so as to insert the new address information. Therefore, the following processor bus operations must be decoded and dealt with as indicated.

- Read, RWITM, read atomic, RWITM atomic—Provided the cache inhibit ($\overline{\text{CI}}$) signal is not asserted, the tag is allocated in the L1 directory as indicated by the CSE*n* signals and address. Additionally, in the 604 these allocations must indicate whether they have caused a data and address pair to be transferred into a copy-back buffer.

  Note that the 601 position is that the cache directory model must be increased in associativity by as many buffers as exist. For example, the 604 would require a four-way instruction cache model and '4+1' data cache model. It is referred to as '4+1' because it is not truly a five-way model since groups 0–3 are selected by CSE, group 4 is replaced by the evicted tag.

- Kill block—TC0 asserted distinguishes kill block operations that deallocate cache entries (caused by DCBI cache operations) from kill block operations that allocate entries in the L1 cache (caused by DCBZ cache operations) or retain a cache entry (a store to a shared entry). Likewise, kill-block operations as a result of a DCBZ operation must also indicate (through TC2) whether an entry has been placed in a copy-back buffer.

  Whenever an allocation is generated by the processor that uses a copy-back buffer, the previous L1 directory entry must be saved into a copy-back address register. The use of these registers is simple first-in/first-out. It is unnecessary to copy the valid bit from the tag directory into the address register, but for repeatability and testing, it is useful to have in the copy-back address registers a valid bit that is initialized to invalid at power-up and loaded with the valid bit from the tag array when an address is copied into one of these registers.

The copy-back address registers could strictly have their valid bit reset whenever a write-with-kill operation matches (which indicates the castout operation is occurring), but this is an optimization that is probably unnecessary. The copy-back address register is more likely to be reloaded with another displaced tag from the L1 than it is to detect a match on the system bus side.

D.2.3 Forwarding System Bus Operations to the Processor

When a system bus operation (with $\overline{\text{SYS-GBL}}$ asserted) occurs on the system bus, it is compared against both the copy of the L1 directory and the copy-back address registers. If there is a match, the system bus operation must be forwarded onto the processor to determine the final outcome. If no match occurs, the addressed data does not reside in the processor and so can complete. Note that this does not address the match in L2 case, which is a separate issue.

However, instead of simply loading the valid bit of the L1 directory shadow with a one when an allocation is detected, it could be loaded with the value of the $\overline{\text{GBL}}$ signal. Comparisons against system bus operations (which were marked as $\overline{\text{SYS-GBL}}$) would still compare the valid bit read from the tag arrays against one. This automatically maximizes use of the information supplied on the $\overline{\text{GBL}}$ signal.

Note the following discussion of system bus operations is concerned with snoop filtering, and hence memory-accessing operations. Clearly operations such as TLBIE and SYNC that do not involve memory accesses are not filtered and are passed to the processor unchanged.

D.3 Maintaining L1 State and Tags

An alternative to simply keeping the tags of an L1 cache is to keep state information about the L1 cache, namely whether the cache line is in the S or E state. Keeping this information allows the L2 cache to filter even more snoop traffic from the processor.

Because some transitions, such as E to M, are invisible outside the processor, maintaining an identical copy of the L1 cache block state is impossible. Thus, the L1 directory copy is restricted to keeping the following range of states, I, S, and EX. The EX state describes both cases of the processor having the data exclusively (M and E). EX implies simply valid but not shared and does not distinguish whether the data has been modified with respect to main memory or to the L2 cache.

D.3.1 Requirements for Saving State Information

The only state additional to that for the simple copy of tags structure required is a single state bit. The tag entry for the data cache now looks like 4 by 128 entries of a valid bit, shared/exclusive bit, and 20 tag bits, which requires only an extra 1/2 Kbits. However, it is likely that the increase would be 1 Kbits since probably the same macrocell would be used in the instruction and data halves of the L1 copy.

D.3.2 Operations Required for Processor Bus Operations

Logic must detect not only whether a tag matches in the L1 directory copy for a system bus operation, but also the type of intervention required and whether it must be passed to the processor before it can complete. This logic is on the critical path for the snoop access, because it must be determined whether $\overline{\text{SYS-ARTRY}}$ or $\overline{\text{SYS-SHD}}$ needs to be asserted in response to the system bus operation.

Along with operations monitored for L1 tag maintenance, the cases in Table D-1 need to be distinguished.

Table D-1. Operations Required for Processor Bus Operations

Bus Operation	Allocate/Deallocate	Action
Read, read atomic	Allocate as per discussion above	State loaded as S if $\overline{\text{SYS-SHD}}$ was asserted or EX if $\overline{\text{SYS-SHD}}$ was negated. It is assumed that the value of $\overline{\text{SHD}}$ reflects the value of $\overline{\text{SYS-SHD}}$ sampled, which is practical in a single-processor system. In a multiprocessor system, it may be desirable to always assert $\overline{\text{SHD}}$ on a read regardless of the state of $\overline{\text{SYS-SHD}}$.
RWITM, RWITM atomic	Allocate as per discussion above	State loaded as EX.
Write with kill	Allocate	State goes to EX (or S, see below).
Write with kill	Deallocate	State goes to INV. It may not match in L1 tag, and the address may already be transferred into the copy-back address register.
Kill block	Store into S cache block or allocate	Allocate tag if necessary and state goes to EX.
Kill block	Deallocate	State goes to I.
ICBI	—	State goes to I.
Flush block	—	State goes to I.
Write with kill	—	Distinguished as with kill block. If TC0 is asserted, the address is deallocated from the cache; if TC0 is negated, then it is retained in the cache (there are no actual allocations associated with a write with kill). As a further optimization, TC1 can be used to determine the final L1 cache state for a write with kil (allocate). If TC1 is asserted, the cache state is S and if it is negated, the cache state is E; the L1 state should be set to S or EX, respectively.

D.3.3 Forwarding System Bus Operations to the Processor

If an L1 tag entry was marked as S, for system read operations (a fairly common occurrence) the L2 controller can directly respond with the $\overline{\text{SYS-SHD}}$ signal without requiring an access to the processor's cache. This not only reduces processor-to-L2 address bus interference, it also improves the system bus bandwidth, as the system bus operation would not need to be retried during interrogation of the processor's L1 cache. Whenever the L1 tag state is E or whenever something other than a simple read operation is performed on the system bus, the operation passes to the processor to determine the final outcome.

D.4 Simple L1 Inclusion

L1 inclusion requires that when an address is not in the L2 cache, it is also not in the L1 cache. Although this functionally is the same as saying that an address cannot be in the L1 cache unless it is in the L2 cache, the first definition more closely reflects how L1 inclusion is implemented.

The simplest approach to L1 inclusion in an L2 cache is to require that whenever something is discarded from the L2 cache, to ensure that it is also discarded from the L1 cache through a back invalidation. In this discussion, the L2 cache is assumed to use four-state MESI protocol. Simplifications to a three-state protocol are trivial.

D.4.1 Requirements for Saving State Information

Simple L1 inclusion requires the same tags and state as is needed to implement the L2. Note that the state information can be kept across the L2 cache block or per-coherency granule.

D.4.2 Operations Required for Processor Bus Operations

The operations performed and monitored to maintain L1 inclusion are like those required for maintaining L1 tags (See Section D.3.2, "Operations Required for Processor Bus Operations.") However, as is discussed below, there is no need to be concerned with the indication that a copy-back buffer is being used.

When an allocation is performed by the L1 cache, the L2 cache must also ensure that a tag is allocated. Before allocation of a tag in the L2, a back invalidation (flush block or RWITM) for each coherency granule removed must be sent to the processor. These back invalidations cause either a snoop miss or hit.

For a snoop miss, the L1 cache has replaced that entry (either previously or for the current allocation) and no further work is needed. Only when snoop push-backs required for all removed granules are completed can the old tag be removed from the L2 directory and the fetch for the new tag begin. The replacement of a tag from the L2 may itself require a copy-back to main memory. Whether the copy-back is buffered is independent from the maintenance of the inclusion.

Allocation operations for L2 inclusion are decoded like those required for maintaining L1 tag copies. With L1 inclusion however, there is no need to monitor the L1 cache's use of its copy-back buffers because the back invalidations force any modified data replaced to be copied back to the L2 level (if not all the way to main memory) before the fetch operations can proceed, thereby reducing the benefit of copy-back buffers in such an environment.

The data cache block allocated in an L2 cache can be larger than that in the L1 cache, in which case multiple back-invalidation operations to the L1 cache may be required whenever a tag is deallocated in the L2 cache, depending upon whether subblocking is implemented. This increases the latency of such operations and must be weighed against the hit rate advantages of such a configuration.

D.4.3 Forwarding System Bus Operations to the Processor

If L1 inclusion is assured, the following scenarios can be considered:

- The system bus operation does not match in the L2 directory. In this case, there can be no copy in the L1 cache, so the system bus operation requires no intervention.
- The address matches in the cache and is in E or M state. In this case, the system bus operation must be retried and the operation must be forwarded to the processor cache because it may have a more up-to-date copy of the data.
- The address matches in the cache and is in the S state. In this case, for the simple case of a read/read-atomic operation, $\overline{\text{SYS-SHD}}$ needs to be asserted only. Other operations may require retrying, even if the data is only state S, as it is reasonable to wait until the operation completes at the processor before letting it complete on the system bus.

D.5 Marked L1 Inclusion

In addition to guaranteeing inclusion, marked inclusion keeps more information about when an L2 cache entry is also in the L1 cache. Viewed narrowly, it is only necessary to require that when an entry is marked as not in the L1 cache, that it in fact not be present. It is acceptable to assume an entry is in the L1 when it is in fact not. Without maintaining a structure that mimics the L1 directory, it is hard to closely match entries marked as included in the L1 with those that actually are. Marked L1 inclusion offers reduction of back invalidations and more efficient snoop filtering than simple L1 inclusion.

D.5.1 Requirements for Saving State Information

The included state for a tag is typically independent of the coherency states supported by the L2 cache; for example, both a shared and an exclusive data entry can be present or not in the L1 cache. Inclusion information is most easily kept on a 32-byte coherency granule (doing otherwise may complicate some mechanisms with no large benefit). Consider the extra state required for a 1-Mbyte L2 cache; for such a configuration, the additional memory required is 32 Kbits.

D.5.2 Operations Required for Processor Bus Operations

The operations performed to maintain marked L1 inclusion are like those required for simple L1 inclusion. When an allocation is performed by the L1 cache, the L2 cache must also ensure that a tag is allocated and that the inclusion bit for the accessed 32-byte granule is set (that this bit might already be set if a processor discarded this block without being detected). Before an L2 cache tag can be allocated, it must be inspected. If the tag contains address granules for which the inclusion bit is set, a back invalidation for each granule must be sent to the processor. If they do not, the tag can be removed directly, assuming that the data is copied back to main memory as required by the state indicated for the cache blocks. The old tag can be removed from the L2 directory and the fetch for the new tag can begin only when the snoop push-backs required for all included granules are completed. As with simple inclusion, replacing a tag from the L2 may require a copy-back to main memory.

The inclusion bit is reset whenever a processor bus operation is performed, which visibly removes an entry from the L1 cache. These operations are as follows:

- Write with kill (deallocate)—A cache castout operation or a snoop response in which the L1 cache state goes to invalid, so the inclusion bit can be reset.
- Kill block (deallocate)—The result of **dcbi** instruction, L1 cache state goes to invalid, so the inclusion bit can be reset.
- ICBI—The result of an **icbi** instruction, L1 cache state goes to invalid, so the inclusion bit can be reset.

Other operations indicate an entry has been removed from the L1 cache; in particular read operations. However, because it is not easy to determine for which L2 cache block to reset the inclusion bit, L2 inclusion bits can only approximate the actual L1 contents.

The exact L2 set index to use can be determined only by creating a structure like the L1 directory that can track both the L1 set index and the L1 group entry information. This structure could be simpler than the L1 directory because it needs no comparator and because it requires only an L2 index entry rather than a tag. However, adding this structure to marked L1 inclusion requires more resources and is needlessly more complicated than simply implementing an L2 cache with a copy of the L1 tag and state information.

D.5.3 Forwarding System Bus Operations to the Processor

When a system bus operation is run, the cases of interest are as follows:

- Snoop miss—Because of L1 inclusion, there is no need to pass the operation onto the processor.
- Snoop hit with inclusion bit reset—There is also no need to pass the operation to the processor.
- Snoop hit with inclusion bit set—The operations are identical to simple L1 inclusion operations.

Appendix E
Coherency Action Tables

The tables in this appendix describe the behavior of the 60x bus when certain operations are presented to the bus. These tables describe the difference in how the bus operates depending upon such factors as the WIM bit settings, the current MESI state, the setting of the transfer type signals (TT[0–4]), and the signals that are presented as the result of the operation ($\overline{\text{ARTRY}}$ and $\overline{\text{SHD}}$) being snooped on the bus.

The tables in this appendix also indicate the difference in how specific 60x processors respond to certain operations.

Abbreviations used in these tables are described in Table E-1.

Table E-1. Guide to Abbreviations

Abbreviation	Meaning
LRS	**lwarx** reservation set
RdA	Read atomic
RWITM	Read-with-intent-to-modify
RWITMA	Read-with-intent-to-modify-atomic
SBR	Single-beat read
SBRA	Single-beat read atomic
SBW	Single-beat write
WWF	Write-with-flush
WWFA	Write-with-flush-atomic
WWK	Write-with-kill

For a description of these operations and others listed in these tables, refer to Section 4.7, “Descriptions of Bus Transactions and Snoop Responses.”

E.1 Load Operations

Table E-2 indicates the behavior and response of 60x processors when a load operation is presented to the system bus.

Table E-2. Coherency Actions—Load Operations

Cache		Processor	Bus			Snoop Response	Processor Response
WIM	MESI		Operation	WIM[3]	TT[0–4]		
000	I	60x	Read	000	01010	(None)	Load block into cache Forward data to perform load Mark cache block E
		603	RWITM		01110		
		60x	Read	000	01010	$\overline{SHD}$	Load block into cache Forward data to perform load Mark cache block S
		603	RWITM		01110		Load block into cache Load from cache Mark cache block E
		60x	Read	000	01010	$\overline{ARTRY}$ or $\overline{ARTRY}$&$\overline{SHD}$	Release bus Retry operation
		603	RWITM		01110		
	MES[1]	60x	(None)	(n/a)	(n/a)	(n/a)	Load from cache
001	I	60x	Read	001	01010	(None)	Load block into cache Forward data to perform load Mark cache block E
		603	RWITM		01110		Load block into cache Load from cache Mark cache block E
		60x	Read	001	01010	$\overline{SHD}$	Load block into cache Forward data to perform load Mark cache block S
		603	RWITM		01110		Load block into cache Load from cache Mark cache block E
		60x	Read	001	01010	$\overline{ARTRY}$ or $\overline{ARTRY}$&$\overline{SHD}$	Release the bus Retry the operation
		603	RWITM		01110		
	MES[1]	60x	(None)	(n/a)	(n/a)	(n/a)	Load from cache

Table E-2. Coherency Actions—Load Operations (Continued)

Cache		Processor	Bus			Snoop Response	Processor Response
WIM	MESI		Operation	WIM[3]	TT[0–4]		
x1x	I	60x	SBR	W1M	01010	(None) or $\overline{\text{SHD}}$	Load from main memory
		603	RWITM		01110		
		60x	SBR	W1M	01010	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		603	RWITM		01110		
	ES[1]	60x	SBR	W1M	01010	(None) or $\overline{\text{SHD}}$	Load from main memory
	E	603	RWITM	x1M	01110	(None) or $\overline{\text{SHD}}$	Load from main memory
		601	(None)	(n/a)	(n/a)	(n/a)	Mark cache block I Cache retry the operation
	ES[1]	60x	SBR	W1M	01010	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
	E	603	RWITM	x1M	01110	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		601	(n/a)	(n/a)	(n/a)	(n/a)	(n/a)
	M	60x	SBR	W1M	01010	(None) or $\overline{\text{SHD}}$	Paradox[2]—cache should be I Load from main memory
		603	RWITM		01110		
		601	WWK		00110	(n/a)	Flush the block Mark cache block I Cache retry the operation
		60x	SBR	W1M	01010	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Paradox[2]—cache should be I Release the bus Retry the operation
		603	RWITM		01110		
		601	WWK		00110		
100	I	60x	Read	100	01010	(None)	Load block into cache Load from cache Mark the cache block E
		603	RWITM		01110		
		60x	Read	100	01010	$\overline{\text{SHD}}$	Load block into cache Load from cache Mark cache block S
		603	RWITM		01110		Load block into cache Load from cache Mark cache block E
		60x	Read	100	01010	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		603	RWITM		01110		
	MES[1]		(None)	(n/a)	(n/a)	(n/a)	Load from cache

Table E-2. Coherency Actions—Load Operations (Continued)

Cache		Processor	Bus			Snoop Response	Processor Response
WIM	MESI		Operation	WIM[3]	TT[0–4]		
101	I	60x	Read	101	01010	(None)	Load block into cache Load from cache Mark cache E
		603	RWITM		01110		
		60x	Read	101	01010	$\overline{\text{SHD}}$	Load block into cache Load from cache Mark cache block S
		603	RWITM		01110		Load block into cache Load from cache Mark cache block E
		60x	Read	101	01010	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY\&SHD}}$	Release the bus Retry the operation
		603	RWITM		01110		
	MES[1]	60x	(None)	(n/a)	(n/a)	(n/a)	Load from cache

Notes:

[1]Because it does not implement shared state, these entries are not applicable to the 603.

[2]A coherency paradox to the processor may cause incoherent data to appear in the system. That is, there is a potential for data integrity errors in the system.

[3] The WIM bits in this column are active-high representations of the active-low $\overline{\text{WT}}$, $\overline{\text{CI}}$, and $\overline{\text{GBL}}$ 60x bus signals, respectively. Thus, a WIM = 101 value corresponds to 60x signal value of $\overline{\text{WT}}$, $\overline{\text{CI}}$, $\overline{\text{GBL}}$ = 010.

E.2 Store Operations

Table E-3 describes the behavior of the 60x bus in response to store operations.

Table E-3. Coherency Actions—Store Operations

Cache		Proc.	Bus			Snoop Response	Processor Response
WIM	MESI		Operation	WIM	TT[0–4]		
000	I	60x	RWITM	000	01110	(None) or $\overline{\text{SHD}}$	Load block into cache Store to cache Mark cache M
						$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
	S[1]	60x	Kill	000	01100	(None) or $\overline{\text{SHD}}$	After kill is successfully presented: Store to cache Mark cache block M
						$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
	E	60x	(None)	(n/a)	(n/a)	(n/a)	Store to cache Mark cache block M
	M	60x	(None)	(n/a)	(n/a)	(n/a)	Store to cache
001	I	60x	RWITM	001	01110	(None) or $\overline{\text{SHD}}$	Load block into cache Mark cache block E Store to cache Mark cache block M
						$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
	S[1]	60x	Kill	001	01100	(None) or $\overline{\text{SHD}}$	After kill is successfully presented: Mark cache block E Store to cache Mark cache block M
						$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
	E	60x	(None)	(n/a)	(n/a)	(n/a)	Store to cache Mark cache block M
	M	60x	(None)	(n/a)	(n/a)	(n/a)	Store to cache

Table E-3. Coherency Actions—Store Operations (Continued)

Cache WIM	Cache MESI	Proc.	Bus Operation	Bus WIM	Bus TT[0–4]	Snoop Response	Processor Response
x1x	I	60x	WWF	x1M	00010	(None) or $\overline{\text{SHD}}$	Store to main memory
		601				$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
	ES[1]	60x	WWF	x1M	00010	(None) or $\overline{\text{SHD}}$	Paradox[2]—cache should be I Store to main memory
		601	(None)	(n/a)	(n/a)	(n/a)	Mark cache block I Cache retry the operation
		60x	WWF	x1M	00010	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Paradox[2]—cache should be I Release the bus Retry the operation
		601	(n/a)	(n/a)	(n/a)	(n/a)	(n/a)
	M	60x	WWF	x1M	00010	(None) or $\overline{\text{SHD}}$	Paradox[2]—cache should be I Store to main memory
		601	WWK		00110		Flush the block Mark cache block I Cache retry the operation
		60x	WWF	x1M	00010	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Paradox[2]—cache should be I Release the bus Retry the operation
		601	WWK		00110		
100	I	60x	WWF	100	00010	(None) or $\overline{\text{SHD}}$	Store to main memory
						$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
	S[1]					(None) or $\overline{\text{SHD}}$	Store to cache Store to main memory
						$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
	E					(None) or $\overline{\text{SHD}}$	Store to cache Store to main memory
						$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
	M	60x	WWF		00010	(None) or $\overline{\text{SHD}}$	Store into cache Store into main memory
		601	WWK		00110		Push the block Mark cache block E Cache retry the operation
		60x	WWF		00010	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		601	WWK		00110		

Table E-3. Coherency Actions—Store Operations (Continued)

Cache		Proc.	Bus			Snoop Response	Processor Response
WIM	**MESI**		**Operation**	**WIM**	**TT[0–4]**		
101	I	60x	WWF	101	00010	(None) or $\overline{\text{SHD}}$	Write to main memory (**Note**: no reload on a store miss)
						$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
	S[1]					(None) or $\overline{\text{SHD}}$	Store to cache Store to main memory
						$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
	E					(None) or $\overline{\text{SHD}}$	Store to cache Store to main memory
						$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
	M	60x	WWF		00010	(None) or $\overline{\text{SHD}}$	Store to cache Store to main memory
		601	WWK		00110		Push block Mark cache block E Cache retry the operation
		60x	WWF		00110	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		601	WWK		00110		

Notes:

[1]Because it does not implement shared state, these entries are not applicable to the 603.

[2]A coherency paradox to the processor may cause incoherent data to appear in the system; that is, there is a potential for data integrity errors in the system.

E.3 LWARX Operations

Table E-4 describes the behavior of the 60x bus in response to LWARX operation generated by the execution of an **lwarx** instruction. Note that the reservation entry in this table refers to reservations associated with the **lwarx** instruction.

Table E-4. Coherency Actions—LWARX Operations

Cache		Proc.	Bus			Reservation	Snoop Response	Processor Response
WIM	MESI		Operation	WIM	TT[0–4]			
000	I	60x	RdA	000	11010	Set by this op	(None)	Load block into cache Set reservation Load from cache Mark cache block E
		603	RWITMA		11110			
		60x	RdA	000	11010	Set by this op	$\overline{\text{SHD}}$	Load block into cache Set reservation Load from cache Mark cache block S
		603	RWITMA		11110			Load block into cache Set reservation Load from cache Mark cache block E
		60x	RdA	000	11010	(n/a)	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		603	RWITMA		11110			
	MES[1]	60x	LRS	000	00001	Set by this op	(None) or $\overline{\text{SHD}}$	Set reservation Load from cache
		601/603	(n/a)	(n/a)	(n/a)		(n/a)	
		60x	LRS	000	00001	(n/a)	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		601/603	(n/a)	(n/a)	(n/a)		(n/a)	(n/a)

Table E-4. Coherency Actions—LWARX Operations (Continued)

Cache		Proc.	Bus			Reservation	Snoop Response	Processor Response
WIM	MESI		Operation	WIM	TT[0–4]			
001	I	60x	RdA	001	11010	Set by this op	(None)	Load block into cache Mark cache block E Set reservation Load from cache
		603	RWITMA		11110			
		60x	RdA	001	11010	Set by this op	$\overline{\text{SHD}}$	Load block into cache Set reservation Load from cache Mark cache block S
		603	RWITMA		11110			Load block into cache Set reservation Load from cache Mark cache block E
		60x	RdA	001	11010	(n/a)	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		603	RWITMA		11110			
	MES[1]	60x	LRS	001	00001	Set by this op	(None) or $\overline{\text{SHD}}$	Set reservation Load from cache
		601/603	(n/a)	(n/a)	(n/a)		(n/a)	
		60x	LRS	001	00001	(n/a)	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		601/603	(n/a)				(n/a)	(n/a)

Table E-4. Coherency Actions—LWARX Operations (Continued)

Cache		Proc.	Bus			Reservation	Snoop Response	Processor Response
WIM	MESI		Operation	WIM	TT[0–4]			
x1x	I	60x	SBRA	x1M	11010	Set by this op	(None) or $\overline{\text{SHD}}$	Set reservation Load from main memory
		603	RWITMA		11110			
		60x	SBRA	x1M	11010	(n/a)	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		603	RWITMA		11110			
	ES[1]	60x	RdA	x1M	11010	Set by this op	(None) or $\overline{\text{SHD}}$	Set the reservation Load from main memory
		603	RWITMA		11110			
		601	(None)	(n/a)	(n/a)	(n/a)	(n/a)	Mark cache block I Cache retry the operation
		60x	RdA	x1M	11010	(n/a)	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		603	RWITMA		11110			
		601	(None)	(n/a)	(n/a)		(n/a)	(n/a)
	M	60x	RdA	x1M	11010	Set by this op	(None) or $\overline{\text{SHD}}$	Paradox[2]—cache should be I Set the reservation Load from main memory
		603	RWITMA		11110			
		601	WWK		00110	(n/a)		Flush the block Mark cache block I Cache retry the operation
		60x	RdA	x1M	11010	(n/a)	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Paradox[2]—cache should be I Release the bus Retry the operation
		603	RWITMA		11110			
		601	WWK		00110			
100	I	601[3]	RdA	100	11010	Set by this op	(None)	Load block into cache Set reservation Load from cache Mark cache block E
							$\overline{\text{SHD}}$	Load block into cache Set reservation Load from cache Mark cache block S
						(n/a)	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
	MES	601[3]	RdA	(n/a)	(n/a)	Set by this op	(n/a)	Set reservation Load from cache

Table E-4. Coherency Actions—LWARX Operations (Continued)

Cache WIM	Cache MESI	Proc.	Bus Operation	Bus WIM	Bus TT[0–4]	Reservation	Snoop Response	Processor Response
101	I	601[3]	RdA	101	11010	Set by this op	(None)	Load block into cache Set reservation Load from cache Mark cache block E
							$\overline{\text{SHD}}$	Load block into cache Set reservation Load from cache Mark cache block S
						(n/a)	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
	MES	601[3]	(n/a)	(n/a)	(n/a)	Set by this op	(n/a)	Set reservation Load from cache

Notes:

[1]Because it does not implement shared state, these entries are not applicable to the 603.

[2]A coherency paradox to the processor may cause incoherent data to appear in the system. That is, there is a potential for data integrity errors in the system.

[3]An LWARX to a page marked write-through causes a DSI exception; therefore, this transaction does not occur on the bus.

E.4 STWCX Operations

Table E-5 describes the behavior of the 60x bus in response to STWCX operation generated by the execution of an **stwcx.** instruction. Note that the reservation entry in this table refers to reservations set by the **lwarx** instruction and cleared either by the **stwcx.** instruction or by a snoop operation.

Table E-5. Coherency Actions—STWCX Operations

Cache WIM	Cache MESI	Proc.	Bus Operation	Bus WIM	Bus TT[0–4]	Res.	Snoop Response	Processor Response
000	I	60x	(None)	(n/a)	(n/a)	None	(n/a)	Update CR
		60x	RWITMA	000	11110	Yes (and reset)	(None) or $\overline{\text{SHD}}$	Load block into cache Release the reservation Update CR Store to cache Mark cache M
		603	WWFA		10010			Issue WWF on the bus Release the reservation Update CR
		60x	RWITMA	000	11110	Yes	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		603	WWFA		10010			

Table E-5. Coherency Actions—STWCX Operations (Continued)

Cache		Proc.	Bus			Res.	Snoop Response	Processor Response
WIM	MESI		Operation	WIM	TT[0–4]			
000	S[1]	60x	(None)	(n/a)	(n/a)	None	(n/a)	Update CR
			Kill	000	01100	Yes (and reset)	(None) or $\overline{SHD}$	After kill is successfully presented: Release reservation Update CR Store to cache Mark cache block M
					01100	Yes	$\overline{ARTRY}$ or $\overline{ARTRY\&SHD}$	Release the bus Retry the operation
	E	60x	(None)	(n/a)	(n/a)	None	(n/a)	Update CR
		60x	(None)	(n/a)	(n/a)	Yes (and reset)	(n/a)	Release reservation Update CR Store to cache Mark cache block M
		603	WWFA	000	10010		(None) or $\overline{SHD}$	WWFA on the bus Wait for write to complete Release reservation Update CR Store to cache
		60x	(None)	(n/a)	(n/a)	Yes (and reset)	(n/a)	(n/a)
		603	WWFA	000	10010		$\overline{ARTRY}$ or $\overline{ARTRY\&SHD}$	Release the bus Retry the operation
	M	60x	(None)	(n/a)	(n/a)	None	(n/a)	Update CR
		60x	(None)	(n/a)	(n/a)	Yes (and reset)	(n/a)	Release reservation Update CR Store to cache
		603	WWFA	000	10010		(None) or $\overline{SHD}$	WWFA on the bus Wait for write to complete Release reservation Update condition register Store to cache
		60x	(None)	(n/a)	(n/a)	Yes(and reset)	(n/a)	(n/a)
		603	WWFA	000	10010		$\overline{ARTRY}$ or $\overline{ARTRY\&SHD}$	Release the bus Retry the operation

Table E-5. Coherency Actions—STWCX Operations (Continued)

Cache		Proc.	Bus			Res.	Snoop Response	Processor Response
WIM	MESI		Operation	WIM	TT[0–4]			
001	I	60x	(None)	(n/a)	(n/a)	None	(n/a)	Update CR
		60x	RWITMA	001	11110	Yes (and reset)	(None) or $\overline{\text{SHD}}$	Load block into cache Release the reservation Update the CR Store to cache Mark cache M
		603	WWFA		10010			Issue WWF on the bus Release the reservation Update the CR
		60x	RWITMA	001	11110	Yes	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		603	WWFA		10010			
	S[1]	60x	(None)	(n/a)	(n/a)	None	(n/a)	Update CR
			Kill	001	01100	Yes (and reset)	(None) or $\overline{\text{SHD}}$	Release reservation Update CR Mark cache block E Store to cache Mark cache block M
						Yes	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
	E	60x	(None)	(n/a)	(n/a)	None	(n/a)	Update CR
		60x	(None)	(n/a)	(n/a)	Yes (and reset)	(n/a)	Release reservation Update CR Store to cache Mark cache block M
		603	WWFA	001	10010		(None) or $\overline{\text{SHD}}$	WWFA on bus Wait for write to complete Release reservation Update CR Store to cache
		60x	(None)	(n/a)	(n/a)	Yes	(n/a)	(n/a)
		603	WWFA	001	10010		$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation

Table E-5. Coherency Actions—STWCX Operations (Continued)

Cache		Proc.	Bus			Res.	Snoop Response	Processor Response
WIM	MESI		Operation	WIM	TT[0–4]			
001	M	60x	(None)	(n/a)	(n/a)	None	(n/a)	Update CR
		60x	(None)	(n/a)	(n/a)	Yes (and reset)	(n/a)	Release reservation Update CR Store to cache
		603	WWFA	001	10010		(None) or $\overline{\text{SHD}}$	WWFA on bus Wait for write to complete Release reservation Update CR Store to cache
		60x	(None)	(n/a)	(n/a)	Yes	(n/a)	(n/a)
		603	WWFA	001	10010		$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
x1x	I	60x	(None)	(n/a)	(n/a)	None	(n/a)	Update CR
			WWFA	x1M	10010	Yes (and reset)	(None) or $\overline{\text{SHD}}$	Release reservation Update CR Store to main memory
						Yes	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
	ES [1]	60x	(None)	(n/a)	(n/a)	None	(n/a)	Paradox[2]—cache should be I Update CR
		601						Mark cache block I Cache retry the operation
		60x	WWFA	x1M	10010	Yes (and reset)	(None) or $\overline{\text{SHD}}$	Paradox[2]—cache should be I Release reservation Update CR Store to main memory
		601	(None)	(n/a)	(n/a)			Mark cache block I Cache retry the operation
		60x	WWFA	x1M	10010	Yes	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Paradox[2]—cache should be I Release the bus Retry the operation
		601	(n/a)	(n/a)	(n/a)			(n/a)

Table E-5. Coherency Actions—STWCX Operations (Continued)

Cache		Proc.	Bus			Res.	Snoop Response	Processor Response
WIM	MESI		Operation	WIM	TT[0–4]			
x1x	M	60x	(None)	(n/a)	(n/a)	None	(n/a)	Paradox[2]—cache should be I Update CR
		601	WWK	x1M	00110		(None) or $\overline{\text{SHD}}$	Flush the block Mark cache block I Cache retry the operation
		60x	(n/a)	(n/a)	(n/a)	None	(n/a)	(n/a)
		601	WWK	x1M	00110		$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		60x	WWFA	x1M	10010	Yes (and reset)	(None) or $\overline{\text{SHD}}$	Paradox[2]—cache should be I Release reservation Update CR Store to main memory
		601	WWK		00110			Flush the block Mark cache block I Cache retry the operation
		60x	WWFA	x1M	10010	Yes	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Paradox[2]—cache should be I Release the bus Retry the operation
		601	WWK		00110			
100	I	601[3]	(None)	(n/a)	(n/a)	None	(n/a)	Update CR
			RWITMA	100	11110	Yes (and reset)	(None) or $\overline{\text{SHD}}$	Load block of data into cache Release reservation Update the CR Store to cache Mark cache M
						(n/a)	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
	S	601[3]	(None)	(n/a)	(n/a)	(None)	(n/a)	Update CR
			Kill	100	01100	Yes (and reset)	(None) or $\overline{\text{SHD}}$	After kill is successfully presented: Release reservation Update CR Store to cache Mark cache block M
		60x[3]	(n/a)	(n/a)	(n/a)	Yes	(n/a)	See footnote 3
		601	Kill	100	01100	(n/a)	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation

Table E-5. Coherency Actions—STWCX Operations (Continued)

Cache		Proc.	Bus			Res.	Snoop Response	Processor Response
WIM	MESI		Operation	WIM	TT[0–4]			
100	E	601[3]	(None)	(n/a)	(n/a)	None	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Update CR
						Yes (and reset)	(n/a)	Release reservation Update CR Store to cache Mark cache block M
	M	601[3]	(None)	(n/a)	(n/a)	None	(n/a)	Update CR
						Yes (and reset)	(n/a)	Release reservation Update CR Store to cache
101	I	601[3]	(None)	(n/a)	(n/a)	None	(n/a)	Update CR
			RWITMA	101	11110	Yes (and reset)	(None) or $\overline{\text{SHD}}$	Load block of data into cache Release reservation Update CR Store to cache Mark cache M
						(n/a)	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
	S	601[3]	(None)	(n/a)	(n/a)	None	(n/a)	Update CR
			Kill	101	01100	Yes (and reset)	(None) or $\overline{\text{SHD}}$	After kill is successfully presented Release reservation Update CR Store to cache Mark cache block M
	(n/a)	60x[3]	(n/a)	(n/a)	(n/a)	Yes(and not reset)	(n/a)	See footnotes 2 and 3
	S	601	Kill	101	01100	(n/a)	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation

Table E-5. Coherency Actions—STWCX Operations (Continued)

Cache		Proc.	Bus			Res.	Snoop Response	Processor Response
WIM	MESI		Operation	WIM	TT[0–4]			
101	E	601	(None)	(n/a)	(n/a)	None	(n/a)	Update CR
						Yes (and reset)		Release reservation Update CR Store to cache Mark cache block M
	M	601	(None)	(n/a)	(n/a)	None	(n/a)	Update CR
						Yes (and reset)		Release reservation Update CR Store to cache

Notes:

[1]Because it does not implement shared state, these entries are not applicable to the 603.

[2]An **stwcx.** to a page marked write-though causes a DSI exception. Therefore this bus transaction cannot occur. The state of reservation is not changed due to an **stwcx.** to a page marked write-though.

[3]For all but 601s, an LWARX to a page marked write-through causes a DSI exception; therefore this transaction does not occur on the bus.

E.5 DCBT Operations

Table E-6 shows the coherency actions when a DCBT operation is generated by the execution of a **dcbt** instruction.

Table E-6. Coherency Actions—DCBT Operations

Cache		Processor	Bus			Snoop Response	Processor Response
WIM	MESI		Operation	WIM	TT[0–4]		
000	I	60x	Read	000	01010	(None)	Load block into cache Mark the cache E
		603	RWITM		01110		
		60x	Read	000	01010	$\overline{SHD}$	Load block into cache Mark the cache S
		603	RWITM		01110		Load block into cache Mark the cache E
		60x	Read	000	01010	$\overline{ARTRY}$ or $\overline{ARTRY}$&$\overline{SHD}$	Release the bus Retry the operation
		603	RWITM		01110		
	MES[1]	60x	(None)	(n/a)	(n/a)	(n/a)	No-op

Table E-6. Coherency Actions—DCBT Operations (Continued)

Cache		Processor	Bus			Snoop Response	Processor Response
WIM	**MESI**		**Operation**	**WIM**	**TT[0–4]**		
001	I	60x	Read	001	01010	(None)	Load block into cache Mark the cache E
		603	RWITM		01110		
		60x	Read	001	01010	$\overline{SHD}$	Load block into cache Mark the cache S
		603	RWITM		01110		Load block into cache Mark the cache E
		60x	Read	001	01010	$\overline{ARTRY}$ or $\overline{ARTRY}$&$\overline{SHD}$	Release the bus Retry the operation
		603	RWITM		01110		
	MES[1]		(None)	(n/a)	(n/a)	(n/a)	No-op
x1x	I	60x	(None)	x1M	(n/a)	(n/a)	No-op
		601	SBR				
	ES[1]	60x	(None)	(n/a)	(n/a)	(n/a)	No-op
		601					Mark cache block I Cache retry the operation
	M	60x	(None)	(n/a)	(n/a)	(n/a)	No-op
		601	WWK	x1M	00110	(None) or $\overline{SHD}$	Flush the block Mark cache block I Cache retry the operation
	M	60x	(n/a)	(n/a)	(n/a)	(n/a)	(n/a)
		601	WWK	x1M	00110	$\overline{ARTRY}$ or $\overline{ARTRY}$&$\overline{SHD}$	Release the bus Retry the operation
100	I	60x	Read	100	01010	(None)	Load block into cache Mark the cache E
		603	RWITM		01110		
		60x	Read	100	01010	$\overline{SHD}$	Load block into cache Mark the cache S
		603	RWITM		01110		Load block into cache Mark the cache E
		60x	Read	100	01010	$\overline{ARTRY}$ or $\overline{ARTRY}$&$\overline{SHD}$	Release the bus Retry the operation
		603	RWITM		01110		
	MES[1]	60x	(None)	(n/a)	(n/a)	(n/a)	No-op

Table E-6. Coherency Actions—DCBT Operations (Continued)

Cache		Processor	Bus			Snoop Response	Processor Response
WIM	**MESI**		**Operation**	**WIM**	**TT[0–4]**		
101	I	60x	Read	101	01010	(None)	Load block into cache Mark the cache E
		603	RWITM		01110		
		60x	Read	101	01010	$\overline{SHD}$	Load block into cache Mark the cache S
		603	RWITM		01110		Load block into cache Mark the cache E
		60x	Read	101	01010	$\overline{ARTRY}$ or $\overline{ARTRY}$&$\overline{SHD}$	Release the bus Retry the operation
		603	RWITM		01110		
	MES[1]		(None)	(n/a)	(n/a)	(n/a)	No-op

Note: [1]Because it does not implement shared state, these entries are not applicable to the 603.

E.6 DCBTST Operations

Table E-7 describes the behavior of the 60x bus interface in response to the execution of a **dcbtst** instruction.

Table E-7. Coherency Actions—DCBTST Operations

Cache		Processor	Bus			Snoop Response	Processor Response
WIM	MESI		Operation	WIM	TT[0–4]		
000	I	60x	Read	000	01010	(None)	Load the block of data into cache Mark the cache E
		603	RWITM		01110		
		60x	Read	000	01010	$\overline{\text{SHD}}$	Load the block of data into cache Mark the cache S
		603	RWITM		01110		Load the block of data into cache Mark the cache E
		60x	Read	000	01010	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		603	RWITM		01110		
	S[1]	60x	(None)	(n/a)	(n/a)	(n/a)	No-op
	ME			000			
001	I	60x	Read	001	01010	(None)	Load the block of data into cache Mark the cache E
		603	RWITM		01110		
		60x	Read	001	01010	$\overline{\text{SHD}}$	Load the block of data into cache Mark the cache S
		603	RWITM		01110		Load the block of data into cache Mark the cache E
		60x	Read	001	01010	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		603	RWITM		01110		
	MES[1]	60x	(None)	(n/a)	(n/a)	(n/a)	No-op
x1x	I	60x	(None)	x1M	(n/a)	(n/a)	No-op
		601	SBR				
	ES[1]	60x	(None)	(n/a)	(n/a)	(n/a)	No-op
		601					Mark cache block I Cache retry the operation
	M	60x	(None)	(n/a)	(n/a)	(n/a)	No-op
		601	WWK	x1M	00110	(None) or $\overline{\text{SHD}}$	Flush the block Mark cache block I Cache retry the operation
		60x	(None)	(n/a)	(n/a)	(n/a)	(n/a)
		601	WWK	x1M	00110	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation

Table E-7. Coherency Actions—DCBTST Operations (Continued)

Cache		Processor	Bus			Snoop Response	Processor Response
WIM	**MESI**		**Operation**	**WIM**	**TT[0–4]**		
100	I	60x	Read	100	01010	(None)	Load the block of data into cache Mark cache E
		603	RWITM		01110		
		60x	Read	100	01010	$\overline{SHD}$	Load the block of data into cache Mark cache as block S
		603	RWITM		01110		(Ignore the shared response) Load the block of data into cache Mark cache E
		60x	Read	100	01010	$\overline{ARTRY}$ or $\overline{ARTRY}$&$\overline{SHD}$	Release the bus Retry the operation
		603	RWITM		01110		
	MES[1]	60x	(None)	(n/a)	(n/a)	(n/a)	No-op
101	I	60x	Read	101	01010	(None)	Load the block of data into cache Mark cache block E
		603	RWITM		01110		
		60x	Read	101	01010	$\overline{SHD}$	Load the block of data into cache Mark cache block S
		603	RWITM		01110		Load the block of data into cache Mark cache block E
		60x	Read	101	01010	$\overline{ARTRY}$ or $\overline{ARTRY}$&$\overline{SHD}$	Release the bus Retry the operation
		603	RWITM		01110		
	MES[1]	60x	(None)	(n/a)	(n/a)	(n/a)	No-op

Note: [1]Because it does not implement shared state, these entries are not applicable to the 603.

E.7 DCBZ Operations

Table E-8 describes the behavior of the 60x bus interface in response to the execution of a **dcbz** instruction.

Table E-8. Coherency Actions—DCBZ Operations

Cache		Processor	Bus			Snoop Response	Processor Response
WIM	MESI		Operation	WIM	TT[0–4]		
000	I	60x	Kill	000	01100	(None) or $\overline{SHD}$	Establish the block in data cache without fetching the block from main memory Set all bytes to zero Mark cache block M
		603	RWITM				RWITM, then write zeros instead of data Mark cache block M
		60x	Kill	000	01100	$\overline{ARTRY}$ or $\overline{ARTRY}$&$\overline{SHD}$	Release the bus Retry the operation
		603	RWITM				
	S[1]	60x	Kill	000	01100	(None) or $\overline{SHD}$	Clear all bytes in the block Mark cache block M
						$\overline{ARTRY}$ or $\overline{ARTRY}$&$\overline{SHD}$	Release the bus Retry the operation
	E		(None)	000	(n/a)	(n/a)	Clear all bytes in the block Mark cache block M
	M		(None)	(n/a)	(n/a)	(n/a)	Write zeros to all bytes in the cache block
001	I	60x	Kill	001	01100	(None) or $\overline{SHD}$	Establish the block in data cache without fetching the block from main memory Set all bytes to zero Mark cache block M
		603	RWITM				RWITM, then write zeros instead of data Mark cache block M
		60x	Kill	001	01100	$\overline{ARTRY}$ or $\overline{ARTRY}$&$\overline{SHD}$	Release the bus Retry the operation
		603	RWITM				
	S[1]	60x	Kill	001	01100	(None) or $\overline{SHD}$	Mark cache block E Set all bytes of the block to zero Mark the cache block M
						$\overline{ARTRY}$ or $\overline{ARTRY}$&$\overline{SHD}$	Release the bus Retry the operation
	E		(None)	(n/a)	(n/a)	(n/a)	Write zeros to all bytes in the cache block Mark cache block M
	M		(None)	(n/a)	(n/a)	(n/a)	Write zeros to all bytes in the cache block
All others	MES[1]I	(n/a)	(n/a)	(n/a)	(n/a)	(n/a)	A **dcbz** to a cache-inhibited or write-through page causes an alignment exception; this bus transaction cannot occur.

Note: [1]Because it does not implement shared state, these entries are not applicable to the 603.

E.8 DCBST Operations

Table E-9 describes the behavior of the 60x bus interface in response to the execution of a **dcbst** instruction.

Table E-9. Coherency Actions—DCBST Operations

Cache		Processor	Bus			Snoop Response	Processor Response
WIM	MESI		Operation	WIM	TT[0–4]		
000	I	60x	Clean	000	00000	(None) or $\overline{\text{SHD}}$	No-op
		601		100			
		601		100	(n/a)	(n/a)	
		60x	Clean	000	00000	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus
		601		100			
		603	(None)	(n/a)	(n/a)	(n/a)	(n/a)
	S[1]	60x	Clean	000	00000	(None) or $\overline{\text{SHD}}$	No-op
		601		100			
		60x		000	00000	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		601		100			
	E	60x	Clean	000	00000	(None) or $\overline{\text{SHD}}$	No-op
		601		100			
		603	(None)	(n/a)	(n/a)	(n/a)	
		60x	Clean	000	00000	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		601		100			
		603	(None)	(n/a)	(n/a)	(n/a)	(n/a)
	M	60x	WWK	100	00110	(None) or $\overline{\text{SHD}}$	Write the block to main memory Mark cache block E
		603		000			
		60x	WWK	100	00110	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		603		000			

Table E-9. Coherency Actions—DCBST Operations (Continued)

Cache		Processor	Bus			Snoop Response	Processor Response
WIM	MESI		Operation	WIM	TT[0–4]		
001	I	60x	Clean	001	00000	(None) or $\overline{\text{SHD}}$	No-op
		601		101			
		603	(None)	(n/a)	(n/a)	(n/a)	
		60x	Clean	001	00000	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		601		101			
		603	(None)	(n/a)	(n/a)	(n/a)	(n/a)
	S[1]	60x	Clean	001	00000	(None) or $\overline{\text{SHD}}$	No-op
		601		101			
		60x	Clean	001	00000	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		601		101			
	E	60x	Clean	001	00000	(None) or $\overline{\text{SHD}}$	No-op
		601		101			
		603	(None)	(n/a)	(n/a)	(n/a)	
		60x	Clean	001	00000	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		601		101			
		603	(None)	(n/a)	(n/a)	(n/a)	
	M	60x	WWK	001	00110	(None) or $\overline{\text{SHD}}$	Write all bytes in the cache block to main memory Mark cache block E
		601		101			
		603		001			
		60x		001		$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		601		101			
		603		001			
x1x	I	60x	Clean	W1M	00000	(None) or $\overline{\text{SHD}}$	No-op
		601		11M			
		603	(None)	(n/a)	(n/a)	(n/a)	
		60x	Clean	W1M	00000	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		601		11M			
		603	(None)	(n/a)	(n/a)	(n/a)	

Table E-9. Coherency Actions—DCBST Operations (Continued)

Cache		Processor	Bus			Snoop Response	Processor Response
WIM	MESI		Operation	WIM	TT[0–4]		
x1x	ES[1]	60x	Clean	W1M	00000	(None) or $\overline{SHD}$	No-op
		603	(None)	(n/a)	(n/a)	(n/a)	
	M	60x	WWK	100	00110	(None) or $\overline{SHD}$	Write all bytes in the cache block to main memory Mark cache block E
		601		11M			Flush the block Mark cache block I
		60x		100		$\overline{ARTRY}$ or $\overline{ARTRY\&SHD}$	Release the bus Retry the operation
		601		11M			
100	I	60x	Clean	100	00000	(None) or $\overline{SHD}$	No-op
		603	(None)	(n/a)	(n/a)	(n/a)	
		60x	Clean	100	00000	$\overline{ARTRY}$ or $\overline{ARTRY\&SHD}$	Release the bus Retry the operation
		603	(None)	(n/a)	(n/a)	(n/a)	
	S[1]	60x	Clean	100	00000	(None) or $\overline{SHD}$	No-op
		60x		100	00000	$\overline{ARTRY}$ or $\overline{ARTRY\&SHD}$	Release the bus Retry the operation
	E	60x	Clean	100	00000	(None) or $\overline{SHD}$	No-op
		603	(None)	(n/a)	(n/a)	(n/a)	
		60x	Clean	100	00000	$\overline{ARTRY}$ or $\overline{ARTRY\&SHD}$	Release the bus Retry the operation
		603	(None)	(n/a)	(n/a)	(n/a)	(n/a)
	M	60x	WWK	100	00110	(None) or $\overline{SHD}$	Push the block Mark cache block E
						$\overline{ARTRY}$ or $\overline{ARTRY\&SHD}$	Release the bus Retry the operation
101	ES[1]I	60x	Clean	101	00000	(None) or $\overline{SHD}$	No-op
		603	(None)	(n/a)	(n/a)	(n/a)	
		60x	Clean	101	00000	$\overline{ARTRY}$ or $\overline{ARTRY\&SHD}$	Release the bus Retry the operation
		603	(None)	(n/a)	(n/a)	(n/a)	(n/a)

Table E-9. Coherency Actions—DCBST Operations (Continued)

Cache		Processor	Bus			Snoop Response	Processor Response
WIM	MESI		Operation	WIM	TT[0–4]		
101	M	60x	WWK	100	00110	(None) or $\overline{\text{SHD}}$	Push the block Mark cache block E
		601		101			
		60x		100	00110	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		601		101			

Note: [1]Because it does not implement shared state, these entries are not applicable to the 603.

E.9 DCBF Operations

Table E-10 describes the behavior of the 60x bus interface in response to the execution of a **dcbf** instruction.

Table E-10. Coherency Actions—DCBF Operations

Cache		Processor	Bus			Snoop Response	Processor Response
WIM	MESI		Operation	WIM	TT[0–4]		
000	I	60x	Flush	000	00100	(None) or $\overline{\text{SHD}}$	No-op
		601		100			
		603	(None)	(n/a)	(n/a)	(n/a)	
		60x	Flush	000	00100	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		601		100			
		603	(None)	(n/a)	(n/a)	(n/a)	(n/a)
	S[1]	60x	Flush	000	00100	(None) or $\overline{\text{SHD}}$	Mark cache block I
		601		100			
		603	(None)	(n/a)	(n/a)		(n/a)
		60x	Flush	000	00100	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		601		100			
		603	(None)	(n/a)	(n/a)		(n/a)
	E	60x	Flush	000	00100	(None) or $\overline{\text{SHD}}$	Mark cache block I
		601		100			
		603	(None)	(n/a)	(n/a)	(n/a)	
		60x	Flush	000	00100	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		601		100			
		603	(None)	(n/a)	(n/a)	(n/a)	(n/a)
	M	60x	WWK	100	00110	(None) or $\overline{\text{SHD}}$	Write the block of data back to main memory Mark the cache block I
		603		000			
		60x	WWK	100	00110	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		603		000			

Table E-10. Coherency Actions—DCBF Operations (Continued)

Cache		Processor	Bus			Snoop Response	Processor Response
WIM	MESI		Operation	WIM	TT[0–4]		
001	I	60x	Flush	001	00100	(None) or $\overline{SHD}$	No-op
		601		101			
		603	(None)	(n/a)	(n/a)	(n/a)	
		60x	Flush	001	00100	$\overline{ARTRY}$ or $\overline{ARTRY}$&$\overline{SHD}$	Release the bus Retry the operation
		601		101			
		603	(None)	(n/a)	(n/a)	(n/a)	(n/a)
	S[1]	60x	Flush	001	00100	(None) or $\overline{SHD}$	Mark cache block I
		601		101			
		60x		001		$\overline{ARTRY}$ or $\overline{ARTRY}$&$\overline{SHD}$	Release the bus Retry the operation
		601		101			
	E	60x	Flush	001	00100	(None) or $\overline{SHD}$	Mark cache block I
		601		101			
		603	(None)	(n/a)	(n/a)	(n/a)	
		60x	Flush	001	00100	$\overline{ARTRY}$ or $\overline{ARTRY}$&$\overline{SHD}$	Release the bus Retry the operation
		601		101			
		603	(None)	(n/a)	(n/a)	(n/a)	(n/a)
	M	60x	WWK	100	00110	(None) or $\overline{SHD}$	Write all bytes in the cache block to main memory Mark cache block I
		601		101			
		603		001			
		60x	WWK	100	00110	$\overline{ARTRY}$ or $\overline{ARTRY}$&$\overline{SHD}$	Release the bus Retry the operation
		601		101			
		603		001			
x1x	I	60x	Flush	W1M	00100	(None) or $\overline{SHD}$	No-op
		601		11M			
		603	(None)	(n/a)	(n/a)	(n/a)	
		60x	Flush	W1M	00100	$\overline{ARTRY}$ or $\overline{ARTRY}$&$\overline{SHD}$	Release the bus Retry the operation
		601		11M			
		603	(None)	(n/a)	(n/a)	(n/a)	

Table E-10. Coherency Actions—DCBF Operations (Continued)

Cache		Processor	Bus			Snoop Response	Processor Response
WIM	MESI		Operation	WIM	TT[0–4]		
x1x	ES[1]	60x	Flush	W1M	00100	(None) or $\overline{\text{SHD}}$	Mark cache block I
		601	(None)	(n/a)	(n/a)	(n/a)	Cache retry the operation
		603					Mark cache block I
		60x	Flush	W1M	00100	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Retry the operation
		601	(None)	(n/a)	(n/a)	(n/a)	(n/a)
		603					
	M	60x	WWK	100	00110	(None) or $\overline{\text{SHD}}$	Flush the block Mark cache block I
		601		11M			
		60x	WWK	100	00110	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		601		11M			
100	I	60x	Flush	100	00100	(None) or $\overline{\text{SHD}}$	No-op
		603	(None)	(n/a)	(n/a)	(n/a)	
		60x	Flush	100	00100	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		603	(None)	(n/a)	(n/a)	(n/a)	(n/a)
	S[1]	60x	Flush	100	00100	(None) or $\overline{\text{SHD}}$	Mark cache block I
						$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
	E	60x	Flush	100	00100	(None) or $\overline{\text{SHD}}$	Mark cache block I
		603	(None)	(n/a)	(n/a)	(n/a)	
		60x	Flush	100	00100	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		603	(None)	(n/a)	(n/a)	(n/a)	(n/a)
	M	60x	WWK	100	00110	(None) or $\overline{\text{SHD}}$	Push the block Mark cache block I
						$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation

Table E-10. Coherency Actions—DCBF Operations (Continued)

Cache		Processor	Bus			Snoop Response	Processor Response
WIM	MESI		Operation	WIM	TT[0–4]		
101	I	60x	Flush	101	00100	(None) or $\overline{\text{SHD}}$	No-op
		603	(None)	(n/a)	(n/a)	(n/a)	
		60x	Flush	101	00100	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		603	(None)	(n/a)	(n/a)	(n/a)	(n/a)
	S[1]	60x	Flush	101	00100	(None) or $\overline{\text{SHD}}$	Mark cache block I
						$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
	E	60x	Flush	101	00100	(None) or $\overline{\text{SHD}}$	Mark cache block I
		603	(None)	(n/a)	(n/a)	(n/a)	
		60x	Flush	101	00100	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		603	(None)	(n/a)	(n/a)	(n/a)	(n/a)
	M	60x	WWK	100	00110	(None) or $\overline{\text{SHD}}$	Flush the block Mark cache block I
		601		101			
		60x	WWK	100	00110	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		601		101			

Note: [1]Because it does not implement shared state, these entries are not applicable to the 603.

E.10 DCBI Operations

Table E-11 describes the behavior of the 60x bus interface in response to the execution of a **dcbi** instruction.

Table E-11. Coherency Action—DCBI Operations

Cache		Processor	Bus			Snoop Response	Processor Response
WIM	MESI		Operation	WIM	TT[0–4]		
000	I	60x	Kill	000	01100	(None) or $\overline{SHD}$	No-op
		603	(None)	(n/a)	(n/a)	(n/a)	
		60x	Kill	000	01100	$\overline{ARTRY}$ or $\overline{ARTRY}$&$\overline{SHD}$	Release the bus Retry the operation
		603	(None)	(n/a)	(n/a)	(n/a)	(n/a)
	S[1]	60x	Kill	000	01100	(None) or $\overline{SHD}$	Mark the cache block I
						$\overline{ARTRY}$ or $\overline{ARTRY}$&$\overline{SHD}$	Release the bus Retry the operation
	ME	60x	Kill	000	01100	(None) or $\overline{SHD}$	Mark cache block I
		603	(None)	(n/a)	(n/a)	(n/a)	
		60x	Kill	000	01100	$\overline{ARTRY}$ or $\overline{ARTRY}$&$\overline{SHD}$	Release the bus Retry the operation
		603	(None)	(n/a)	(n/a)	(n/a)	(n/a)
001	I	60x	Kill	001	01100	(None) or $\overline{SHD}$	No-op
		603	(None)	(n/a)	(n/a)	(n/a)	
		60x	Kill	001	01100	$\overline{ARTRY}$ or $\overline{ARTRY}$&$\overline{SHD}$	Release the bus Retry the operation
		603	(None)	(n/a)	(n/a)	(n/a)	(n/a)
	S[1]	60x	Kill	001	01100	(None) or $\overline{SHD}$	Mark cache block I
						$\overline{ARTRY}$ or $\overline{ARTRY}$&$\overline{SHD}$	Release the bus Retry the operation
	ME	60x	Kill	001	01100	(None) or $\overline{SHD}$	Mark cache block I
		603	(None)	(n/a)	(n/a)	(n/a)	
		60x	Kill	001	01100	$\overline{ARTRY}$ or $\overline{ARTRY}$&$\overline{SHD}$	Release the bus Retry the operation
		603	(None)	(n/a)	(n/a)	(n/a)	(n/a)

Table E-11. Coherency Action—DCBI Operations (Continued)

Cache WIM	Cache MESI	Processor	Bus Operation	Bus WIM	Bus TT[0–4]	Snoop Response	Processor Response
x1x	I	60x	Kill	W1M	01100	(None) or $\overline{\text{SHD}}$	No-op
		603	(None)	(n/a)	(n/a)	(n/a)	
		60x	Kill	W1M	01100	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		603	(None)	(n/a)	(n/a)	(n/a)	(n/a)
	S[1]E	60x	Kill	W1M	01100	(None) or $\overline{\text{SHD}}$	Mark cache block I
		603	(None)	(n/a)	(n/a)	(n/a)	
		601					Mark cache block I Cache retry the operation
		60x	Kill	W1M	01100	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		601/603	(None)	(n/a)	(n/a)	(n/a)	(n/a)
	M	60x	Kill	W1M	01100	(None) or $\overline{\text{SHD}}$	Mark cache block I
		603	(None)	(n/a)	(n/a)	(n/a)	
		601	WWK	W1M	01100	(None) or $\overline{\text{SHD}}$	Flush the block Mark cache block I Cache retry the operation
		60x	Kill	W1M	01100	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		603	(None)	(n/a)	(n/a)	(n/a)	(n/a)
		601	WWK	W1M	01100	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
100	I	60x	Kill	100	01100	(None) or $\overline{\text{SHD}}$	No-op
		603	(None)	(n/a)	(n/a)	(n/a)	
		60x	Kill	100	01100	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		603	(None)	(n/a)	(n/a)	(n/a)	(n/a)
	S[1]	60x	Kill	100	01100	(None) or $\overline{\text{SHD}}$	Mark cache block I
						$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
	ME	60x	Kill	100	01100	(None) or $\overline{\text{SHD}}$	Mark cache block I
		603	(None)	(n/a)	(n/a)	(n/a)	
		60x	Kill	100	01100	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		603	(None)	(n/a)	(n/a)	(n/a)	(n/a)

Table E-11. Coherency Action—DCBI Operations (Continued)

Cache		Processor	Bus			Snoop Response	Processor Response
WIM	MESI		Operation	WIM	TT[0–4]		
101	I	60x	Kill	101	01100	(None) or $\overline{\text{SHD}}$	No-op
		603	(None)	(n/a)	(n/a)	(n/a)	
		60x	Kill	101	01100	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		603	(None)	(n/a)	(n/a)	(n/a)	(n/a)
	S[1]	60x	Kill	101	01100	(None) or $\overline{\text{SHD}}$	Mark cache block I
		603	(None)	(n/a)	(n/a)	(n/a)	(n/a)
		60x	Kill	101	01100	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		603	(None)	(n/a)	(n/a)	(n/a)	(n/a)
	ME	60x	Kill	101	01100	(None) or $\overline{\text{SHD}}$	Mark cache block I
		603	(None)	(n/a)	(n/a)	(n/a)	
		60x	Kill	101	01100	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		603	(None)	(n/a)	(n/a)	(n/a)	(n/a)

Note: [1]Because it does not implement shared state, these entries are not applicable to the 603.

E.11 ICBI Operations

Table E-12 describes the behavior of the 60x bus interface in response to the execution of an **icbi** instruction.

Table E-12. Coherency Actions—ICBI Operations

Cache		Processor	Bus			Snoop Response	Processor Response
WIM	MESI		Operation	WIM	TT[0–4]		
000	I	60x	ICBI	000	01101	(None) or $\overline{SHD}$	No-op
		603	(None)	(n/a)	(n/a)	(n/a)	
		601	(n/a)				
		60x	ICBI	000	01101	$\overline{ARTRY}$ or $\overline{ARTRY}$&$\overline{SHD}$	Release the bus Retry the operation
		603	(None)	(n/a)	(n/a)	(n/a)	(n/a)
		601	(n/a)				
	VAL	60x	ICBI	000	01101	(None) or $\overline{SHD}$	Mark I-cache block I
		603	(None)	(n/a)	(n/a)	(n/a)	
		601	(n/a)				No-op (unified cache)
		60x	ICBI	000	01101	$\overline{ARTRY}$ or $\overline{ARTRY}$&$\overline{SHD}$	Release the bus Retry the operation
		603	(None)	(n/a)	(n/a)		(n/a)
		601	(n/a)			(n/a)	
001	I	60x	ICBI	001	01101	(None) or $\overline{SHD}$	No-op
		603	(None)	(n/a)	(n/a)	(n/a)	
		601	(n/a)				
		60x	ICBI	001	01101	$\overline{ARTRY}$ or $\overline{ARTRY}$&$\overline{SHD}$	Release the bus Retry the operation
		603	(None)	(n/a)	(n/a)	(n/a)	(n/a)
		601	(n/a)				
	VAL	60x	ICBI	001	01101	(None) or $\overline{SHD}$	Mark I-cache block I
		603	(None)	(n/a)	(n/a)	(n/a)	
		601	(n/a)				No-op (unified cache)
		60x	ICBI	001	01101	$\overline{ARTRY}$ or $\overline{ARTRY}$&$\overline{SHD}$	Release the bus Retry the operation
		603	(None)	(n/a)	(n/a)		(n/a)
		601	(n/a)			(n/a)	

Table E-12. Coherency Actions—ICBI Operations (Continued)

Cache		Processor	Bus			Snoop Response	Processor Response
WIM	MESI		Operation	WIM	TT[0–4]		
x1x	I	60x	ICBI	x1M	01101	(None) or $\overline{SHD}$	No-op
		603	(None)	(n/a)	(n/a)	(n/a)	
		601	(n/a)				No-op (unified cache)
		60x	ICBI	x1M	01101	$\overline{ARTRY}$ or $\overline{ARTRY\&SHD}$	Release the bus Retry the operation
		603	(None)	(n/a)	(n/a)	(n/a)	(n/a)
		601	(n/a)				
	VAL	60x	ICBI	x1M	01101	(None) or $\overline{SHD}$	Mark I-cache block I
		603	(None)	(n/a)	(n/a)	(n/a)	
		601	(n/a)				No-op (unified cache)
		60x	ICBI	x1M	01101	$\overline{ARTRY}$ or $\overline{ARTRY\&SHD}$	Release the bus Retry the operation
		603	(None)	(n/a)	(n/a)	(n/a)	(n/a)
		601	(n/a)				
100	I	60x	ICBI	100	01101	(None) or $\overline{SHD}$	No-op
		603	(None)	(n/a)	(n/a)	(n/a)	
		601	(n/a)				
		60x	ICBI	100	01101	$\overline{ARTRY}$ or $\overline{ARTRY\&SHD}$	Release the bus Retry the operation
		603	(None)	(n/a)	(n/a)	(n/a)	(n/a)
		601	(n/a)				
	VAL	60x	ICBI	100	01101	(None) or $\overline{SHD}$	Mark I-cache block I
		603	(None)	(n/a)	(n/a)	(n/a)	
		601	(n/a)				No-op (unified cache)
		60x	ICBI	100	01101	$\overline{ARTRY}$ or $\overline{ARTRY\&SHD}$	Release the bus Retry the operation
		603	(None)	(n/a)	(n/a)		(n/a)
		601	(n/a)			(n/a)	

Table E-12. Coherency Actions—ICBI Operations (Continued)

Cache		Processor	Bus			Snoop Response	Processor Response
WIM	MESI		Operation	WIM	TT[0–4]		
101	I	60x	ICBI	101	01101	(None) or $\overline{\text{SHD}}$	No-op
		603	(None)	(n/a)	(n/a)	(n/a)	
		601	(n/a)				
		60x	ICBI	101	01101	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		603	(None)	(n/a)	(n/a)	(n/a)	(n/a)
		601	(n/a)				
	VAL	60x	ICBI	101	01101	(None) or $\overline{\text{SHD}}$	Mark I-cache block I
		603	(None)	(n/a)	(n/a)	(n/a)	
		601	(n/a)				No-op (unified cache)
		60x	ICBI	101	01101	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation
		603	(None)	(n/a)	(n/a)		(n/a)
		601	(n/a)			(n/a)	

E.12 SYNC Operations

Table E-13 describes the behavior of the 60x bus interface in response to the execution of a **sync** instruction. This table does not fully describe the operation of the **sync** instruction.

Table E-13. Coherency Actions—SYNC Operations

Cache		Bus		Snoop Response	Processor Response
WIM	MESI	WIM	TT[0–4]		
(n/a)	(n/a)	xx1	01000	(None) or $\overline{\text{SHD}}$	The **sync** instruction has completed
				$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation

E.13 EIEIO Operations

Table E-14 describes the behavior of the 60x bus interface in response to the execution of an **eieio** instruction.This table does not fully describe the operation of the **eieio** instruction.

Table E-14. Coherency Actions—EIEIO Operations

Cache		Bus		Snoop Response	Processor Response
WIM	**MESI**	**WIM**	**TT[0–4]**		
(n/a)	(n/a)	xx1	10000	(None) or $\overline{\text{SHD}}$	The **eieio** instruction completed.
				$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation

E.14 TLBIE Operations

Table E-15 describes the behavior of the 60x bus interface in response to the execution of a **tlbie** instruction. This table does not fully describe the operation of the **tlbie** instruction.

Table E-15. Coherency Actions—TLBIE Operations

Cache		Bus		Snoop Response	Processor Response
WIM	**MESI**	**WIM**	**TT[0–4]**		
(n/a)	(n/a)	xx1	11000	(None) or $\overline{\text{SHD}}$	Hold off any new memory instructions Wait for completion of any outstanding memory instructions Invalidate the requested TLB entry
				$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation

E.15 TLBSYNC Operations

Table E-16 describes the behavior of the 60x bus interface in response to the execution of a **tlbsync** instruction. This table does not fully describe the operation of the **tlbsync** instruction.

Table E-16. Coherency Actions—TLBSYNC Operations

Cache		Bus		Snoop Response	Processor Response
WIM	**MESI**	**WIM**	**MESI**		
(n/a)	(n/a)	xx1	01001	(None) or $\overline{\text{SHD}}$	The **tlbsync** instruction has completed
			01001	$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Release the bus Retry the operation

E.16 Snoop-Kill Operations

Table E-17 describes the behavior of the 60x bus interface in response to a snoop-kill bus operation.

Table E-17. Coherency Actions—Snoop-Kill Operations

Cache		Processor	Bus		Reservation	Snoop Response	Processor Response
WIM	**MESI**		**WIM**	**TT[0–4]**			
(n/a)	I	60x	xx1	01100	None	(None)	No-op
		60x			Yes (and reset)		Release reservation
		603					No-op
	S[1]EM	60x			None		Mark cache block I
		603					No-op
		60x			Yes (and reset)		Mark cache block I Release reservation
		603					No-op
	M	604			Yes (and reset)	$\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Try to write cache block back to main memory If successful, mark cache block I

Note: [1]Because it does not implement shared state, these entries are not applicable to the 603.

E.17 Snoop-Read Operations

Table E-18 describes the behavior of the 60x bus interface in response to a snoop-read bus operation.

Table E-18. Coherency Actions—Snoop-Read Operations

Cache		Proc.	Bus		Reservation	Snoop Response	Processor Response
WIM	MESI		WIM	TT[0–4]			
(n/a)	I	60x	x11	01010	None	(None)	No-op
		60x			Yes	$\overline{\text{SHD}}$	
		603				(None)	
	S[1]	60x			(n/a)	$\overline{\text{SHD}}$	No-op
	E	60x				$\overline{\text{SHD}}$	Mark cache block S
		603				(None)	Mark cache block I
	M	60x	x01			$\overline{\text{ARTRY}}$ or $\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Try to write cache block back to main memory If successful, mark cache block S
		603					Try to write cache block back to main memory If successful, mark cache block I
		60x	x11				Try to write cache block back to main memory If successful, mark cache block S
		603					Try to write cache block back to main memory If successful, mark cache block E

Note: [1]Because it does not implement shared state, these entries are not applicable to the 603.

E.18 Snoop-Read-Atomic Operations

Table E-19 describes the behavior of the 60x bus interface in response to a snoop-read atomic bus operation.

Table E-19. Coherency Actions—Snoop-Read Atomic Operations

Cache		Proc.	Bus		Res.	Snoop Response	Processor Response
WIM	MESI		WIM	TT[0–4]			
(n/a)	I	60x	xx1	11010	None	(None)	No-op
		60x			Yes	$\overline{SHD}$	
		603				None	
	S[1]	60x			None	$\overline{SHD}$	No-op
	E	60x			(n/a)	$\overline{SHD}$	Mark cache block S
		603				None	Mark cache block I
	M	60x	x01			$\overline{ARTRY}$&$\overline{SHD}$	Try to write cache block back to main memory If successful, mark cache block S
		603					Try to write cache block back to main memory If successful, mark cache block I
		60x	x11				Try to write cache block back to main memory If successful, mark cache block S
		603					Try to write cache block back to main memory If successful, mark cache block E

Note: [1]Because it does not implement shared state, these entries are not applicable to the 603.

E.19 Snoop-RWITM Operations

Table E-20 describes the behavior of the 60x bus interface in response to a snoop-RWITM bus operation.

Table E-20. Coherency Actions—Snoop-RWITM Operations

Cache		Bus		Reservation	Snoop Response	Processor Response
WIM	MESI	WIM	TT[0–4]			
(n/a)	I	xx1	01110	None	(None)	No-op
				Yes (and reset)		Release reservation
	S[1]E			None		Mark cache block I
				Yes (and reset)		Mark cache block I Release reservation
	M			None	$\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Try to write cache block back to main memory If successful, mark cache block I
				Yes (and reset)		Try to write cache block back to main memory If successful: Mark cache block I Release reservation

Note: [1]Because it does not implement shared state, these entries are not applicable to the 603.

E.20 Snoop-RWITM-Atomic Operations

Table E-21 describes the behavior of the 60x bus interface in response to a snoop-snoop-RWITM atomic bus operation.

Table E-21. Coherency Actions—Snoop-RWITM Atomic Operations

Cache		Bus		Reservation	Snoop Response	Processor Response
WIM	MESI	WIM	TT[0–4]			
(n/a)	I	xx1	11110	None	(None)	No-op
				Yes (and reset)		Release reservation
	S[1]E			None		Mark cache block I
				Yes (and reset)		Mark cache block I Release reservation
	M			None	$\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Try to write cache block back to main memory If successful, mark cache block I
				Yes (and reset)		Try to write cache block back to main memory If successful: Mark cache block I Release reservation

Note: [1]Because it does not implement shared state, these entries are not applicable to the 603.

E.21 Snoop-Flush Operations

Table E-22 describes the behavior of the 60x bus interface in response to a snoop-snoop-flush bus operation.

Table E-22. Coherency Actions—Snoop-Flush Operations

Cache		Proc.	Bus		Reservation	Snoop Response	Processor Response
WIM	MESI		WIM	TT[0–4]			
(n/a)	I	60x	xx1	00100	None	(None)	No-op
					Yes	(None)	No-op. Snoop-flush operation cannot clear the reservation.
	S[1]E	60x			(n/a)	(None)	Mark cache block I
		603					No-op
	M	60x				$\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Try to write cache block back to main memory If successful, mark cache block I
		603				(None)	No-op

Note: [1]Because it does not implement shared state, these entries are not applicable to the 603.

E.22 Snoop-Clean Operations

Table E-23 describes the behavior of the 60x bus interface in response to a snoop-clean bus operation.

Table E-23. Coherency Actions—Snoop-Clean

Cache		Processor	Bus		Snoop Response	Processor Response
WIM	MESI		WIM	TT[0–4]		
(n/a)	ES[1]I	60x	xx1	00000	(None)	No-op
	M	60x	xx1		$\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Try to write cache block back to main memory If successful, mark cache block E
		603			(None)	No-op

Note: [1]Because it does not implement shared state, these entries are not applicable to the 603.

E.23 Snoop-Write-with-Flush Operations

Table E-24 describes the behavior of the 60x bus interface in response to a snoop-write with flush bus operation.

Table E-24. Coherency Actions—Snoop-Write-with-Flush Operations

Cache		Bus		Reservation	Snoop Response	Processor Response
WIM	MESI	WIM	TT[0–4]			
(n/a)	I	xx1	00010	None	(None)	No-op
				Yes (and reset)		Release reservation
	S[1]			None		Mark cache block I
				Yes (and reset)		Mark cache block I Release reservation
	E			None		Paradox[2]—no one else should be writing if this cache is E Mark cache block I
				Yes (and reset)		Paradox[2]—no one else should be writing if this cache is E Mark cache block I Release reservation
	M			None	$\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Paradox[2]—no one else should be writing if this cache is M Try to write cache block back to main memory If successful, mark cache block I
				Yes (and reset)		Paradox[2]—no one else should be writing if this cache is M Try to write cache block back to main memory If successful: Mark cache block I Release reservation

Notes:

[1]Because it does not implement shared state, these entries are not applicable to the 603.

[2]A coherency paradox to the processor may cause incoherent data to appear in the system. That is, there is a potential for data integrity errors in the system.

E.24 Snoop-Write-with-Kill Operations

Table E-25 describes the behavior of the 60x bus interface in response to a snoop-write-with-kill bus operation.

Table E-25. Coherency Actions—Snoop-Write-with-Kill Operations

Cache		Bus		Reservation	Snoop Response	Processor Response
WIM	MESI	WIM	TT[0–4]			
(n/a)	I	xx1	00110	None	(None)	No-op
				Yes (and reset)		Release reservation
	S[1]			None		Mark cache block I
				Yes (and reset)		Mark cache block I Release reservation
	E			None		Paradox[2]—no one else should be writing if this cache is E Mark cache block I
				Yes (and reset)		Paradox[2]—no one else should be writing if this cache is E Mark cache block I Release reservation
	M			None		Paradox[2]—no one else should be writing if this cache is M Mark cache block I
				Yes (and reset)		Paradox[2]—no one else should be writing if this cache is M Mark cache block I Release reservation

Notes:

[1]Because it does not implement shared state, these entries are not applicable to the 603.

[2]A coherency paradox to the processor may cause incoherent data to appear in the system. That is, there is a potential for data integrity errors in the system.

E.25 Snoop-Write-with-Flush-Atomic Operations

Table E-26 describes the behavior of the 60x bus interface in response to a snoop-write-with-flush-atomic bus operation.

Table E-26. Coherency Actions—Snoop-Write-with-Flush-Atomic Operations

Cache		Bus		Reservation	Snoop Response	Processor Response
WIM	MESI	WIM	TT[0–4]			
(n/a)	I	xx1	00110	None	(None)	No-op
				Yes (and reset)		Release reservation
	S[1]			None		Mark cache block I
				Yes (and reset)		Mark cache block I Release reservation
	E			None		Paradox[2]—no one else should be writing if this cache is E Mark cache block I
				Yes (and reset)		Paradox[2]—no one else should be writing if this cache is E Mark cache block I Release reservation
	M			None	$\overline{\text{ARTRY}}$&$\overline{\text{SHD}}$	Paradox[2]—no one else should be writing if this cache is M Try to write block back to main memory If successful, mark cache block I
				Yes (and reset)		Paradox[2]—no one else should be writing if this cache is M Try to write block back to main memory If successful: Mark cache block I Release reservation

Notes:

[1]Because it does not implement shared state, these entries are not applicable to the 603.

[2]A coherency paradox to the processor may cause incoherent data to appear in the system. That is, there is a potential for data integrity errors in the system.

E.26 Snoop-TLB-Invalidate Operations

Table E-27 describes the behavior of the 60x bus interface in response to a snoop-TLB-invalidate bus operation.

Table E-27. Coherency Actions—Snoop-TLB-Invalidate Operations

Cache		Bus		Snoop Response	Processor Response
WIM	MESI	WIM	TT[0–4]		
(n/a)	(n/a)	xx1	11000	(None)	Respond with (None) when all previous TLB invalidates have been performed
				(None) but $\overline{\text{ARTRY}}$ is activated on the bus from another processor	Do not perform the TLB invalidate—this is to prevent a deadlock condition from occurring
				$\overline{\text{ARTRY}}$	Respond with retry until TLB has been invalidated. Previous TLB invalidate is still in progress.

E.27 Snoop-SYNC Operations

Table E-28 describes the behavior of the 60x bus interface in response to a snoop-SYNC bus operation.

Table E-28. Coherency Actions—Snoop-SYNC Operations

Cache		Bus		Snoop Response	Processor Response
WIM	MESI	WIM	TT[0–4]		
(n/a)	(n/a)	xx1	01000	(None)	If there are no TLB invalidates pending that were initiated by this processor, no-op
				$\overline{\text{ARTRY}}$	If there is a TLB invalidate pending that was initiated by this processor, respond with retry. OR If there is a snoop push address tenure (write/w/kill) pending due to previous snoop.

E.28 Snoop-EIEIO Operations

Table E-29 describes the behavior of the 60x bus interface in response to a snoop-EIEIO bus operation.

Table E-29. Coherency Actions—Snoop-EIEIO Operations

Cache		Bus		Snoop Response	Processor Response
WIM	MESI	WIM	TT[0–4]		
(n/a)	(n/a)	xx1	10000	(None)	No-op. The 604 family never asserts $\overline{\text{ARTRY}}$ for an EIEIO snoop.
				$\overline{\text{ARTRY}}$	

E.29 Snoop-TLBSYNC Operations

Table E-30 describes the behavior of the 60x bus interface in response to a snoop-TLBSYNC bus operation.

Table E-30. Coherency Actions—Snoop-TLBSYNC Operations

Cache		Bus		Snoop Response	Processor Response
WIM	MESI	WIM	TT[0–4]		
(n/a)	(n/a)	xx1	01001	(None)	If no TLB invalidates are pending and there are no marked transactions, no-op. Note that all queues in the processor with translated addresses are considered marked whenever a TLBI snoop operation completes. These transactions may hit in the cache internally and clear the mark. Other times, these transactions must complete a 60x bus address tenure before these marks can be cleared. TLBSYNC snoops are $\overline{\text{ARTRY}}$d until all marks are cleared.
				$\overline{\text{ARTRY}}$	If a TLB invalidate is pending or if any marked transactions are pending, respond with retry

E.30 Snoop-ICBI Operations

Table E-31 describes the behavior of the 60x bus interface in response to a snoop-ICBI bus operation.

Table E-31. Coherency Actions—Snoop-ICBI Operations

Cache		Bus		Snoop Response	Processor Response
WIM	MESI	WIM	TT[0–4]		
(n/a)	I	xx1	011001	(None)	No-op
	VAL				Invalidate entry in I-cache

E.31 Snoop-RWNITC Operations

Table E-32 describes the behavior of the 60x bus interface in response to a snoop-RWNITC bus operation.

Table E-32. Coherency Actions—Snoop-RWNITC Operations

Cache		Processor	Bus		Reservation	Snoop Response	Processor Response
WIM	MESI		WIM	TT[0–4]			
(n/a)	I	60x	xx1	01011	None	(None)	No-op
		60x			Yes	$\overline{SHD}$	No-op
		603				(n/a)	(n/a)
	S[1]	60x			(n/a)	$\overline{SHD}$	No-op
	E	60x				$\overline{SHD}$	No-op
		603				(None)	Mark cache block E
	M	601				$\overline{ARTRY}$&$\overline{SHD}$	Try to write cache block back to main memory If successful, mark cache block S
		604					Try to write cache block back to main memory If successful, mark cache block E
		603				$\overline{ARTRY}$	No-op

Note: [1]Because it does not implement shared state, these entries are not applicable to the 603.

Glossary of Terms and Abbreviations

The glossary contains an alphabetical list of terms, phrases, and abbreviations used in this book. Some of the terms and definitions included in the glossary are reprinted from *IEEE Std 754-1985, IEEE Standard for Binary Floating-Point Arithmetic*, copyright ©1985 by the Institute of Electrical and Electronics Engineers, Inc. with the permission of the IEEE.

A

Architecture. A detailed specification of requirements for a processor or computer system. It does not specify details of how the processor or computer system must be implemented; instead it provides a template for a family of compatible *implementations*.

Asynchronous exception. *Exceptions* that are caused by events external to the processor's execution. In this document, the term 'asynchronous exception' is used interchangeably with the word *interrupt*.

Atomic access. A bus access that attempts to be part of a read-write operation to the same address uninterrupted by any other access to that address (the term refers to the fact that the transactions are indivisible). The 60x processors implement atomic accesses through the **lwarx**/**stwcx.** instruction pair.

B

BAT (block address translation) mechanism. A software-controlled array that stores the available block address translations on-chip.

Beat. A single state on the 60x bus interface that may extend across multiple bus cycles. A 60x transaction can be composed of multiple address or data beats.

Biased exponent. An *exponent* whose range of values is shifted by a constant (bias). Typically a bias is provided to allow a range of positive values to express a range that includes both positive and negative values.

Big-endian. A byte-ordering method in memory where the address n of a word corresponds to the *most-significant byte*. In an addressed memory word, the bytes are ordered (left to right) 0, 1, 2, 3, with 0 being the most-significant byte.

Block. An area of memory that ranges from 128 Kbyte to 256 Mbyte, whose size, translation, and protection attributes are controlled by the *BAT mechanism.*

Boundedly undefined. A characteristic of results of certain operations that are not rigidly prescribed by the PowerPC architecture. Boundedly-undefined results for a given operation may vary among implementations, and between execution attempts in the same implementation.

Although the architecture does not prescribe the exact behavior for when results are allowed to be boundedly undefined, the results of executing instructions in contexts where results are allowed to be boundedly undefined are constrained to ones that could have been achieved by executing an arbitrary sequence of defined instructions, in valid form, starting in the state the machine was in before attempting to execute the given instruction.

Burst. A multiple beat data transfer whose total size is typically equal to a cache block.

Bus clock. Clock that causes the bus state transitions.

Bus master. The owner of the address or data bus; the device that initiates or requests the transaction.

C

Cache. High-speed memory component containing recently-accessed data and/or instructions (subset of main memory).

Cache block. A small region of contiguous memory that is copied from memory into a *cache*. The size of a cache block may vary among processors; the maximum block size is one *page*. In PowerPC processors, *cache coherency* is maintained on a cache-block basis. Note that the term ‘cache block’ is often used interchangeably with ‘cache line’.

Cache coherency. An attribute wherein an accurate and common view of memory is provided to all devices that share the same memory system. Caches are coherent if a processor performing a read from its cache is supplied with data corresponding to the most recent value written to memory or to another processor’s cache.

Cache flush. An operation that removes from a cache any data from a specified address range. This operation ensures that any modified data within the specified address range is written back to main memory. This operation is generated typically by a Data Cache Block Flush (**dcbf**) instruction.

Caching-inhibited. A memory update policy in which the *cache* is bypassed and the load or store is performed to or from main memory.

Cast-outs. *Cache blocks* that must be written to memory when a cache miss causes a cache block to be replaced.

Clear. To cause a bit or bit field to register a value of zero. *See also* Set.

Context synchronization. An operation that ensures that all instructions in execution complete past the point where they can produce an *exception*, that all instructions in execution complete in the context in which they began execution, and that all subsequent instructions are *fetched* and executed in the new context. Context synchronization may result from executing specific instructions (such as **isync** or **rfi**) or when certain events occur (such as an exception).

Copy-back. An operation in which modified data in a *cache block* is copied back to memory.

D

Denormalized number. A nonzero floating-point number whose *exponent* has a reserved value, usually the format's minimum, and whose explicit or implicit leading significand bit is zero.

Direct-mapped cache. A cache in which each main memory address can appear in only one location within the cache, operates more quickly when the memory request is a cache hit.

Direct-store. Interface available on PowerPC processors only to support direct-store devices from the POWER architecture. When the T bit of a *segment descriptor* is set, the descriptor defines the region of memory that is to be used as a direct-store segment. Note that this facility is being phased out of the architecture and will not likely be supported in future devices. Therefore, software should not depend on it and new software should not use it.

E

Effective address (EA). The 32- or 64-bit address specified for a load, store, or an instruction fetch. This address is then submitted to the MMU for translation to either a *physical memory* address or an I/O address.

E**xception**. A condition encountered by the processor that requires special, supervisor-level processing.

Exception handler. A software routine that executes when an exception is taken. Normally, the exception handler corrects the condition that caused the exception, or performs some other meaningful task (that may include aborting the program that caused the exception). The address for each exception handler is identified by an exception vector offset defined by the architecture and a prefix selected via the MSR.

Exclusive state. MESI state (E) in which only one caching device contains data that is also in system memory.

Execution synchronization. A mechanism by which all instructions in execution are architecturally complete before beginning execution (appearing to begin execution) of the next instruction. Similar to context synchronization but doesn't force the contents of the instruction buffers to be deleted and refetched.

Exponent. In the binary representation of a floating-point number, the exponent is the component that normally signifies the integer power to which the value two is raised in determining the value of the represented number. *See also* Biased exponent.

F

Feed-forwarding. A feature that reduces the number of clock cycles that an execution unit must wait to use a register. When the source register of the current instruction is the same as the destination register of the previous instruction, the result of the previous instruction is routed to the current instruction at the same time that it is written to the register file. With feed-forwarding, the destination bus is gated to the waiting execution unit over the appropriate source bus, saving the cycles which would be used for the write and read.

Fetch. Retrieving instructions from either the cache or main memory and placing them into the instruction queue.

Floating-point register (GPR). Any of the 32 registers in the floating-point register file. These registers provide the source operands and destination results for all floating-point data manipulation instructions. Floating-point load instructions move data from memory to registers, and floating-point store instructions move data from registers to memory.

Flush. An operation that causes a modified cache block to be invalidated and the data to be written to memory.

Fraction. In the binary representation of a floating-point number, the field of the *significand* that lies to the right of its implied binary point.

G

General-purpose register (GPR). Any of the 32 registers in the general purpose register file. These registers provide the source operands and destination results for all integer data manipulation instructions. Load instructions move data from memory to registers, and store instructions move data from registers to memory.

H

Harvard architecture. An architectural model featuring separate caches for instruction and data.

I

IEEE 754. A standard written by the Institute of Electrical and Electronics Engineers that defines operations and representations of binary floating-point arithmetic.

Implementation. A particular processor that conforms to the PowerPC architecture, but may differ from other architecture-compliant implementations for example in design, feature set, and implementation of *optional* features. The PowerPC architecture has many different implementations.

Implementation-dependent. An aspect of a feature in a processor’s design that is defined by a processor’s design specifications rather than by the PowerPC architecture.

Implementation-specific. An aspect of a feature in a processor’s design that is not required by the PowerPC architecture, but for which the PowerPC architecture may provide concessions to ensure that processors that implement the feature do so consistently.

Imprecise exception. A type of *synchronous exception* that is allowed not to adhere to the precise exception model (*see* Precise exception). The PowerPC architecture allows only floating-point exceptions to be handled imprecisely.

Inexact. Loss of accuracy in an arithmetic operation when the rounded result differs from the infinitely precise value with unbounded range.

In-order. An aspect of an operation that adheres to a sequential model. An operation is said to be performed in-order if, at the time that it is performed, it is known to be required by the sequential execution model. *See* Out-of-order.

Instruction queue. A holding place for instructions fetched from the current instruction stream.

Interrupt. An external signal that causes the processor to suspend current execution and take a predefined exception.

Invalid state. State of a cache entry that does not currently contain a valid copy of a cache block from memory.

K

Key bits. A set of key bits referred to as Ks and Kp in each segment register and each BAT register. The key bits determine whether supervisor or user programs can access a *page* within that *segment* or *block*.

Kill. An operation that causes a *cache block* to be invalidated.

L

Latency. The number of clock cycles necessary to perform an action, such as a memory access.

Least-significant bit (lsb). The bit of least value in an address, register, data element, or instruction encoding.

Least-significant byte (LSB). The byte of least value in an address, register, data element, or instruction encoding.

Little-endian. A byte-ordering method in memory where the address *n* of a word corresponds to the *least-significant byte*. In an addressed memory word, the bytes are ordered (left to right) 3, 2, 1, 0, with 3 being the *most-significant byte*. *See* Big-endian.

Livelock. A state in which processors interact in a way such that no processor makes progress.

M

Memory-mapped accesses. Accesses whose addresses use the segmented or block address translation mechanisms provided by the MMU and that occur externally with the bus protocol defined for memory.

Memory coherency. Refers to memory agreement between caches and system memory (for example, MESI cache coherency).

Memory consistency. Refers to levels of memory with respect to a single processor and system memory (for example, on-chip cache, secondary cache, and system memory).

Memory management unit. The functional unit that is capable of translating an *effective* (logical) *address* to a physical address, providing protection mechanisms, and defining caching methods.

MESI (modified/exclusive/shared/invalid). *Cache coherency* protocol used to manage caches on different devices that share a memory system. Note that the PowerPC architecture does not specify the implementation of a MESI protocol to ensure cache coherency.

Modified state. When a cache block is in the modified state, it has been modified by the processor since it was copied from memory. *See* MESI.

Multiprocessing. The capability of software, especially operating systems, to support execution on more than one processor at the same time.

Most-significant bit (msb). The highest-order bit in an address, registers, data element, or instruction encoding.

Most-significant byte (MSB). The highest-order byte in an address, registers, data element, or instruction encoding.

O

OEA (operating environment architecture). The level of the architecture that describes PowerPC memory management model, supervisor-level registers, synchronization requirements, and the exception model. It also defines the time-base feature from a supervisor-level perspective. Implementations that conform to the PowerPC OEA also conform to the PowerPC UISA and VEA.

Optional. A feature, such as an instruction, a register, or an exception, that is defined by the PowerPC architecture but not required to be implemented.

Out-of-order. An aspect of an operation that allows it to be performed ahead of one that may have preceded it in the sequential model, for example, speculative operations. An operation is said to be performed out-of-order if, at the time that it is performed, it is not known to be required by the sequential execution model. *See* In-order.

Overflow. An error condition that occurs during arithmetic operations when the result cannot be stored accurately in the destination register(s). For example, if two 32-bit numbers are multiplied, the result may not be representable in 32 bits.

P

Packet. A term used with respect to direct-store operations.

Page. A region in memory. The OEA defines a page as a 4-Kbyte area of memory, aligned on a 4-Kbyte boundary.

Page table entry (PTE). Data structures containing information used to translate *effective address* to physical address on a 4-Kbyte page basis. A PTE consists of 8 bytes of information in a 32-bit processor.

Park. The act of allowing a bus master to maintain mastership of the bus without having to arbitrate.

Pipelining. A technique that breaks operations, such as instruction processing or bus transactions, into smaller distinct stages or tenures (respectively) so that a subsequent operation can begin before the previous one has completed.

Precise exceptions. The pipeline can be stopped so the instructions that preceded the faulting instruction can complete, and subsequent instructions can be executed following the execution of the exception handler. The system is precise unless one of the imprecise modes for invoking the floating-point enabled exception is in effect.

Physical memory. The actual memory that can be accessed through the system's memory bus.

Q

Quad word. A group of 16 contiguous locations starting at an address divisible by 16.

Quiesce. To come to rest. The processor is said to quiesce when an exception is taken or a **sync** instruction is executed. The instruction stream is stopped at the decode stage and executing instructions are allowed to complete to create a controlled context for instructions that may be affected by out-of-order, parallel execution. *See* Context synchronization.

R

Reservation. The processor establishes a reservation on a *cache block* of memory space when it executes an **lwarx** instruction to read a memory semaphore into a GPR.

Reserved field. In a register, a reserved field is one that is not assigned a function. A reserved field may be a single bit. The handling of reserved bits is *implementation-dependent*. Software is permitted to write any value to such a bit. A subsequent reading of the bit returns 0 if the value last written to the bit was 0 and returns an undefined value (0 or 1) otherwise.

RISC (reduced instruction set computing). An *architecture* characterized by fixed-length instructions with nonoverlapping functionality and by a separate set of load and store instructions that perform memory accesses.

S

Segment. A 256-Mbyte area of *virtual memory* that is the most basic memory space defined by the PowerPC architecture. Each segment is configured through a unique *segment descriptor.*

Segment descriptors. Information used to generate the interim *virtual address*. The segment descriptors reside in 16 on-chip segment registers for 32-bit implementations.

Set (*v*). To write a nonzero value to a bit or bit field; the opposite of *clear.* The term ‘set’ may also be used to generally describe the updating of a bit or bit field.

Set (*n*). A subdivision of a *cache*. Cacheable data can be stored in a given location in any one of the sets, typically corresponding to its lower-order address bits. Because several memory locations can map to the same location, cached data is typically placed in the set whose *cache block* corresponding to that address was used least recently.
See Set-associativity.

Set-associativity. Aspect of cache organization in which the cache space is divided into sections, called *sets*. The cache controller associates a particular main memory address with the contents of a particular set, or region, within the cache.

Significand. The component of a binary floating-point number that consists of an explicit or implicit leading bit to the left of its implied binary point and a fraction field to the right.

Slave. The device addressed by a master device. The slave is identified in the address tenure and is responsible for supplying or latching the requested data for the master during the data tenure.

Snooping. Monitoring addresses driven by a bus master to detect the need for coherency actions.

Snoop push. Write-backs due to a snoop hit. The block will transition to an invalid or exclusive state.

Split-transaction. A transaction with independent request and response tenures.

Split-transaction bus. A bus that allows address and data transactions from different processors to occur independently.

Strong ordering. A memory access model that requires exclusive access to an address before making an update, to prevent another device from using stale data.

Supervisor mode. The privileged operation state. In supervisor mode, software can access all control registers and can access the supervisor memory space, among other privileged operations.

Synchronization. A process to ensure that operations occur strictly *in order. See* Context synchronization and Execution synchronization.

Synchronous exception. An *exception* that is generated by the execution of a particular instruction or instruction sequence. There are two types of synchronous exceptions, *precise* and *imprecise*.

System memory. The physical memory available to a processor.

T

TLB (translation lookaside buffer) A cache that holds recently-used *page table entries*.

Tenure. The period of bus mastership. There can be separate address bus tenures and data bus tenures. A tenure consists of three phases: arbitration, transfer, termination.

Throughput. The measure of the number of instructions that are processed per clock cycle.

Transaction. A complete exchange between two bus devices. A transaction is minimally comprised of an address tenure; one or more data tenures may be involved in the exchange. There are two kinds of transactions: address/data and address-only.

Transfer termination. Signal that refers to both signals that acknowledge the transfer of individual beats (of both single-beat transfer and individual beats of a burst transfer) and to signals that mark the end of the tenure.

U

UISA (user instruction set architecture). 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, floating-point memory conventions and exception model as seen by user programs, and the memory and programming models.

Unified cache. Combined data and instruction cache.

User mode. The unprivileged operating state of a processor used typically by application software. In user mode, software can only access certain control registers and can access only user memory space. No privileged operations can be performed. Also referred to as problem state.

V

VEA (virtual environment architecture). The level of the *architecture* that describes the memory model for an environment in which multiple devices can access memory, defines aspects of the cache model, defines cache control instructions, and defines the time-base facility from a user-level perspective. *Implementations* that conform to the PowerPC VEA also adhere to the UISA, but may not necessarily adhere to the OEA.

Virtual address. An intermediate address used in the translation of an *effective address* to a physical address.

W

Weak ordering. A memory access model that allows bus operations to be reordered dynamically, which improves overall performance and in particular reduces the effect of memory latency on instruction throughput.

Word. A 32-bit data element.

Write-back. A cache memory update policy in which processor write cycles are directly written only to the cache. External memory is updated only indirectly, for example, when a modified cache block is *cast out* to make room for newer data.

Write-through. A cache memory update policy in which all processor write cycles are written to both the cache and memory.

INDEX

INDEX

INDEX

INDEX