MPC603EUM/AD
11/97
REV. 1

MPC603e & EC603e™

RISC Microprocessors User's Manual
with Supplement for PowerPC 603™ Microprocessor

MOTOROLA

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

Paragraph Number	Title	Page Number

CONTENTS

CONTENTS

CONTENTS

Paragraph Number	Title	Page Number

CONTENTS

ILLUSTRATIONS

ILLUSTRATIONS

ILLUSTRATIONS

ILLUSTRATIONS

Figure Number	Title	Page Number

TABLES

TABLES

TABLES

TABLES

TABLES

About This Book

The primary objective of this user's manual is to define the functionality of the PowerPC 603™ and PowerPC 603e™ microprocessors for use by software and hardware developers. Although the emphasis of this manual is upon the 603e, all of the information within applies to both the 603 and 603e, except for those differences noted in Appendix C, "PowerPC 603 Processor System Design and Programming Considerations." Those readers who are primarily interested in the 603 should begin with Appendix C.

In addition, this book describes the EC603e™ microprocessor. The EC603e microprocessor for embedded systems is functionally equivalent to the 603e with the exception of the floating-point unit which is not supported on the EC603e microprocessor; therefore, the term 'EC603e' is used only when it is necessary to distinguish functional differences with the EC603e microprocessor.

The 603e is built upon the low-power dissipation, low-cost and high-performance attributes of the 603 while providing the system designer additional capabilities through higher processor clock speeds, increases in cache size (16-Kbyte instruction and data caches) and set-associativity (4-way), and greater system clock flexibility. The 603e only implements the 32-bit portion of the PowerPC™ architecture.

The 603e and EC603e microprocessors are implemented in both a 2.5-volt version (PID 0007v 603e microprocessor, abbreviated as PID7v-603e) and a 3.3-volt version (PID 0006 603e microprocessor, abbreviated as PID6-603e).

In this document, the term '603e' is used as an abbreviation for 'PowerPC 603e microprocessor' and the term '603' is an abbreviation for 'PowerPC 603 microprocessor'. The PowerPC 603e microprocessors are available from Motorola as MPC603e. The EC603e microprocessors are available from Motorola as MPE603e.

It is important to note that this book is intended as a companion to the *PowerPC Microprocessor Family: The Programming Environments*, referred to as *The Programming Environments Manual*; contact your local sales representative to obtain a copy. Because the PowerPC architecture is designed to be flexible to support a broad range of processors, *The Programming Environments Manual* provides a general description of features that are common to PowerPC processors and indicates those features that are optional or that may be implemented differently in the design of each processor.

This document summarizes features of the 603e that are not defined by the architecture. This document and *The Programming Environments Manual* distinguish between the three levels, or programming environments, of the PowerPC architecture, which are as follows:

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

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

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

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

For ease in reference, this book has arranged topics described by the architecture into topics that build upon one another, beginning with a description and complete summary of 603e-specific registers and progressing to more specialized topics such as 603e-specific details regarding the cache, exception, and memory management models. As such, chapters may include information from multiple levels of the architecture. (For example, the discussion of the cache model uses information from both the VEA and the OEA.)

The PowerPC Architecture: A Specification for a New Family of RISC Processors defines the architecture from the perspective of the three programming environments and remains the defining document for the PowerPC architecture.

The information in this book is subject to change without notice, as described in the disclaimers on the title page of this book. As with any technical documentation, it is the readers' responsibility to be sure they are using the most recent version of the documentation. For more information, contact your sales representative.

Audience

This manual is intended for system software and hardware developers and applications programmers who want to develop products using the 603e microprocessors. It is assumed that the reader understands operating systems, microprocessor system design, the basic principles of RISC processing, and details of the PowerPC architecture.

Organization

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

- Chapter 1, "Overview," is useful for readers who want a general understanding of the features and functions of the PowerPC architecture and the 603e. This chapter describes the flexible nature of the PowerPC architecture definition, and provides an overview of how the PowerPC architecture defines the register set, operand conventions, addressing modes, instruction set, cache model, exception model, and memory management model.
- Chapter 2, "Programming Model," provides a brief synopsis of the registers implemented in the 603e, operand conventions, an overview of the PowerPC addressing modes, and a list of the instructions implemented by the 603e. Instructions are organized by function.
- Chapter 3, "Instruction and Data Cache Operation," provides a discussion of the cache and memory model as implemented on the 603e.
- Chapter 4, "Exceptions," describes the exception model defined in the PowerPC OEA and the specific exception model implemented on the 603e.
- Chapter 5, "Memory Management," describes the 603e's implementation of the memory management unit specifications provided by the PowerPC OEA for PowerPC processors.
- Chapter 6, "Instruction Timing," provides information about latencies, interlocks, special situations, and various conditions to help make programming more efficient. This chapter is of special interest to software engineers and system designers.
- Chapter 7, "Signal Descriptions," provides descriptions of individual signals of the 603e.
- Chapter 8, "System Interface Operation," describes signal timings for various operations. It also provides information for interfacing to the 603e.
- Chapter 9, "Power Management," provides information about power saving modes for the 603e.

- Appendix A, “PowerPC Instruction Set Listings,” lists all the PowerPC instructions while indicating those instructions that are not implemented by the 603e; it also includes the instructions that are specific to the 603e. Instructions are grouped according to mnemonic, opcode, function, and form. Also included is a quick reference table that contains general information, such as the architecture level, privilege level, and form, and indicates if the instruction is 64-bit and optional.
- Appendix B, “Instructions Not Implemented,” provides a list of PowerPC instructions not implemented by the 603e.
- Appendix C, “PowerPC 603 Processor System Design and Programming Considerations,” provides a discussion of the hardware and software differences between the 603 and 603e.
- This manual also includes a glossary and an index.

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 available from the sources listed on the back cover of this manual; the document order numbers are included in parentheses for ease in ordering:

- 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 604™ RISC Microprocessor User's Manual*: MPC604UM/AD (Motorola order #)
  - — *MPC750 RISC Microprocessor User's Manual*: MPC750UM/AD (Motorola order #)
  - — *PowerPC 620™ RISC Microprocessor User's Manual*: MPC620UM/AD (Motorola 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 #)
  - — *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.motorola.com/PowerPC/.
- 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 604 RISC Microprocessor User's Manual*: *PowerPC 604e™ Microprocessor Supplement and User's Manual Errata*: MPC604UMAD/AD (Motorola 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 603 RISC Microprocessor Hardware Specifications*: MPC603EC/D (Motorola order #)
  - — *PowerPC 603e RISC Microprocessor Family: PID6-603e Hardware Specifications*: MPC603EEC/D (Motorola order #)

— *PowerPC 603e RISC Microprocessor Family: PID7v-603e Hardware Specifications*:
MPC603E7VEC/D (Motorola order #)

— *PowerPC 603e RISC Microprocessor Family: PID7t-603e Hardware Specifications*:
MPC603E7TEC/D (Motorola order #)

— *PowerPC 604 RISC Microprocessor Hardware Specifications*:
MPC604EC/D (Motorola order #)

— *PowerPC 604e RISC Microprocessor Family: PID9v-604e Hardware Specifications*:
MPC604E9VEC/D (Motorola order #)

— *PowerPC 604e RISC Microprocessor Family: PID9q-604e Hardware Specifications*:
MPC604E9QEC/D (Motorola order #)

— *MPC750 RISC Microprocessor Hardware Specifications*
MPC750EC/D (Motorola order #)

— *EC603e Embedded RISC Microprocessor (PID6) Hardware Specifications*:
MPE603EEC/D (Motorola order #)

— *EC603e Embedded RISC Microprocessor (PID7v) Hardware Specifications*:
MPE603E7VEC/D (Motorola 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, 603, 603e, 604, 604e, and EC603e microprocessors which can be ordered as follows:

— *EC603e Embedded RISC Microprocessor Technical Summary*:
MPE603E/D (Motorola order #)

• *PowerPC Microprocessor Family: The Bus Interface for 32-Bit Microprocessors*: MPCBUSIF/AD (Motorola order #) provides a detailed functional description of the 60x bus interface, as implemented on the 601, 603, and 604 family of PowerPC microprocessors. This document is intended to help system and chipset developers by providing a centralized reference source to identify the bus interface presented by the 60x family of PowerPC microprocessors.

• *PowerPC Microprocessor Family: The Programmer's Reference Guide*: MPCPRG/D (Motorola order #) is a concise reference that includes the register summary, memory control model, exception vectors, and the PowerPC instruction set.

• *PowerPC Microprocessor Family: The Programmer's Pocket Reference Guide*: MPCPRGREF/D (Motorola order #)
This foldout card provides an overview of the PowerPC registers, instructions, and exceptions for 32-bit implementations.

- 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/SPS/PowerPC/.

Conventions

This document uses the following notational conventions:

mnemonics	Instruction mnemonics are shown in lowercase bold.
italics	Italics indicate variable command parameters, for example, **bcctr***x*. Book titles in text are set in italics.
0x0	Prefix to denote 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
¬	NOT logical operator
&	AND logical operator
|	OR logical operator
0 0 0 0	Indicates reserved bits or bit fields in a register. Although these bits may be written to as either ones or zeros, they are always read as zeros.

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
ATE	Automatic test equipment
ASR	Address space register
BAT	Block address translation
BIST	Built-in self test
BIU	Bus interface unit
BPU	Branch processing unit
BUC	Bus unit controller
BUID	Bus unit ID
CAR	Cache address register
CIA	Current instruction address
CMOS	Complementary metal-oxide semiconductor
COP	Common on-chip processor
CR	Condition register
CRTRY	Cache retry queue
CTR	Count register
DAR	Data address register
DBAT	Data BAT
DCMP	Data TLB compare
DEC	Decrementer register
DMISS	Data TLB miss address
DSISR	Register used for determining the source of a DSI exception
DTLB	Data translation lookaside buffer
EA	Effective address
EAR	External access register
ECC	Error checking and correction
FIFO	First-in-first-out
FPR	Floating-point register (Note that the EC603e microprocessor does not support the floating-point unit.)
FPSCR	Floating-point status and control register (Note that the EC603e microprocessor does not support the floating-point unit.)

Table i. Acronyms and Abbreviated Terms (Continued)

Term	Meaning
FPU	Floating-point unit (Note that the EC603e microprocessor does not support the floating-point unit.)
GPR	General-purpose register
HASH1	Primary hash address
HASH2	Secondary hash address
IABR	Instruction address breakpoint register
IBAT	Instruction BAT
ICMP	Instruction TLB compare
IEEE	Institute for Electrical and Electronics Engineers
IMISS	Instruction TLB miss address
IQ	Instruction queue
ITLB	Instruction translation lookaside buffer
IU	Integer unit
L2	Secondary cache
LIFO	Last-in-first-out
LR	Link register
LRU	Least recently used
LSB	Least-significant byte
lsb	Least-significant bit
LSU	Load/store unit
MEI	Modified/exclusive/invalid
MESI	Modified/exclusive/shared/invalid—cache coherency protocol
MMU	Memory management unit
MQ	MQ register
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
PIR	Processor identification register
PLL	Phase-locked loop

Table i. Acronyms and Abbreviated Terms (Continued)

Term	Meaning
POWER	Performance Optimized with Enhanced RISC architecture
PTE	Page table entry
PTEG	Page table entry group
PVR	Processor version register
RAW	Read-after-write
RISC	Reduced instruction set computing
RPA	Required physical address
RTL	Register transfer language
RWITM	Read with intent to modify
SDR1	Register that specifies the page table base address for virtual-to-physical address translation
SLB	Segment lookaside buffer
SPR	Special-purpose register
SR	Segment register
SRR0	Machine status save/restore register 0
SRR1	Machine status save/restore register 1
SRU	System register unit
TAP	Test access port
TB	Time base facility
TBL	Time base lower register
TBU	Time base upper register
TLB	Translation lookaside buffer
TTL	Transistor-to-transistor logic
UIMM	Unsigned immediate value
UISA	User instruction set architecture
UTLB	Unified translation lookaside buffer
UUT	Unit under test
VEA	Virtual environment architecture
WAR	Write-after-read
WAW	Write-after-write
WIMG	Write-through/caching-inhibited/memory-coherency enforced/guarded bits
XATC	Extended address transfer code
XER	Register used for indicating conditions such as carries and overflows for integer operations

Terminology Conventions

Table ii describes terminology conventions used in this manual.

Table ii. Terminology Conventions

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

Table iii describes instruction field notation used in this manual.

Table iii. Instruction Field Conventions

The Architecture Specification	Equivalent to:
BA, BB, BT	**crb**A, **crb**B, **crb**D (respectively)
BF, BFA	**crf**D, **crf**S (respectively)
D	d
DS	ds
FLM	FM
FRA, FRB, FRC, FRT, FRS	**fr**A, **fr**B, **fr**C, **fr**D, **fr**S (respectively)
FXM	CRM
RA, RB, RT, RS	**r**A, **r**B, **r**D, **r**S (respectively)
SI	SIMM
U	IMM
UI	UIMM
/, //, ///	0...0 (shaded)

Chapter 1
Overview

This chapter provides an overview of features of the PowerPC 603e™ microprocessor and the PowerPC™ architecture, and information about how the 603e implementation complies with the architectural definitions. In addition, this book describes the EC603e microprocessor. Note that the 603e and EC603e microprocessors are implemented in both a 2.5-volt version (PID 0007v 603e microprocessor, abbreviated as PID7v-603e) and a 3.3-volt version (PID 0006 603e microprocessor, abbreviated as PID6-603e).

1.1 Overview

This section describes the details of the 603e, provides a block diagram showing the major functional units, and describes briefly how those units interact. Any differences between the PID6-603e, PID7v-603e, and EC603e implementations are noted.

The 603e is a low-power implementation of the PowerPC microprocessor family of reduced instruction set computing (RISC) microprocessors. The 603e implements the 32-bit portion of the PowerPC architecture, which provides 32-bit effective addresses, integer data types of 8, 16, and 32 bits, and floating-point data types of 32 and 64 bits.

The 603e is a superscalar processor that can issue and retire as many as three instructions per clock. Instructions can execute out of order for increased performance; however, the 603e makes completion appear sequential.

The 603e integrates five execution units—an integer unit (IU), a floating-point unit (FPU) (not supported on the EC603e microprocessor), a branch processing unit (BPU), a load/store unit (LSU), and a system register unit (SRU). The ability to execute five instructions in parallel and the use of simple instructions with rapid execution times yield high efficiency and throughput for 603e-based systems. Most integer instructions execute in one clock cycle. On the 603e, the FPU is pipelined so a single-precision multiply-add instruction can be issued and completed every clock cycle. (Note that the EC603e microprocessor does not support the floating-point unit.)

The 603e provides independent on-chip, 16-Kbyte, four-way set-associative, physically addressed caches for instructions and data and on-chip instruction and data memory management units (MMUs). The MMUs contain 64-entry, two-way set-associative, data and instruction translation lookaside buffers (DTLB and ITLB) that provide support for demand-paged virtual memory address translation and variable-sized block translation. The TLBs and caches use a least recently used (LRU) replacement algorithm. The 603e also supports block address translation through the use of two independent instruction and data block address translation (IBAT and DBAT) arrays of four entries each. Effective addresses are compared simultaneously with all four entries in the BAT array during block translation. In accordance with the PowerPC architecture, if an effective address hits in both the TLB and BAT array, the BAT translation takes priority.

The 603e has a selectable 32- or 64-bit data bus and a 32-bit address bus. The 603e interface protocol allows multiple masters to compete for system resources through a central external arbiter. The 603e provides a three-state coherency protocol that supports the exclusive, modified, and invalid cache states. This protocol is a compatible subset of the MESI (modified/exclusive/shared/invalid) four-state protocol and operates coherently in systems that contain four-state caches. The 603e supports single-beat and burst data transfers for memory accesses, and supports memory-mapped I/O operations.

The 603e is fabricated using an advanced CMOS process technology and is fully compatible with TTL devices.

1.1.1 Features

This section describes the major features of the 603e noting where the PID6-603e, PID7v-603e, and EC603e implementations differ:

- High-performance, superscalar microprocessor
  - As many as three instructions issued and retired per clock
  - As many as five instructions in execution per clock
  - Single-cycle execution for most instructions
  - Pipelined FPU for all single-precision and most double-precision operations (The EC603e microprocessor does not support the floating-point unit.)
- Five independent execution units and two register files
  - BPU featuring static branch prediction
  - A 32-bit IU
  - Fully IEEE 754-compliant FPU for both single- and double-precision operations (The EC603e microprocessor does not support the floating-point unit.)
  - LSU for data transfer between data cache and GPRs and FPRs (The EC603e microprocessor does not support the floating-point unit.)
  - SRU that executes condition register (CR), special-purpose register (SPR), and integer add/compare instructions
  - Thirty-two GPRs for integer operands

- Thirty-two FPRs for single- or double-precision operands
  (The EC603e microprocessor does not support the floating-point unit.)

- High instruction and data throughput
  - Zero-cycle branch capability (branch folding)
  - Programmable static branch prediction on unresolved conditional branches
  - Instruction fetch unit capable of fetching two instructions per clock from the instruction cache
  - A six-entry instruction queue that provides lookahead capability
  - Independent pipelines with feed-forwarding that reduces data dependencies in hardware
  - 16-Kbyte data cache—four-way set-associative, physically addressed; LRU replacement algorithm
  - 16-Kbyte instruction cache—four-way set-associative, physically addressed; LRU replacement algorithm
  - Cache write-back or write-through operation programmable on a per page or per block basis
  - BPU that performs CR lookahead operations
  - Address translation facilities for 4-Kbyte page size, variable block size, and 256-Mbyte segment size
  - A 64-entry, two-way set-associative ITLB
  - A 64-entry, two-way set-associative DTLB
  - Four-entry data and instruction BAT arrays providing 128-Kbyte to 256-Mbyte blocks
  - Software table search operations and updates supported through fast trap mechanism
  - 52-bit virtual address; 32-bit physical address
- Facilities for enhanced system performance
  - A 32- or 64-bit split-transaction external data bus with burst transfers
  - Support for one-level address pipelining and out-of-order bus transactions
  - Hardware support for misaligned little-endian accesses (PID7v-603e)

- Integrated power management
  - — Low-power 2.5-volt and 3.3-volt designs
  - — Internal processor/bus clock multiplier ratios as follows:
    - – 1/1, 1.5/1, 2/1, 2.5/1, 3/1, 3.5/1, and 4/1 (PID6-603e)
    - – 2/1, 2.5/1, 3/1, 3.5/1, 4/1, 4.5/1, 5/1, 5.5/1, and 6/1 (PID7v-603e)
  - — Three power-saving modes: doze, nap, and sleep
  - — Automatic dynamic power reduction when internal functional units are idle
- In-system testability and debugging features through JTAG boundary-scan capability

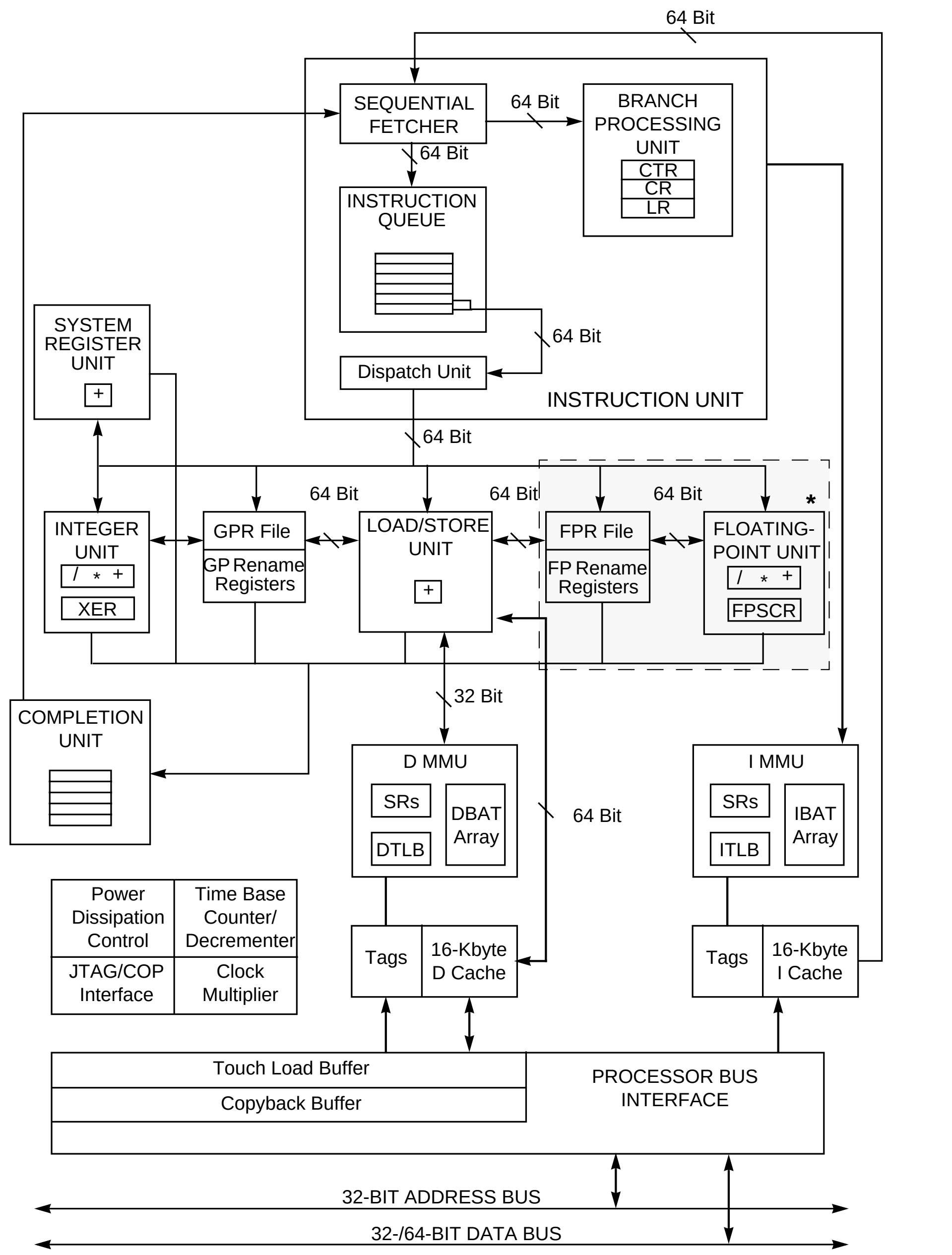
Features specific to the PID7v-603e follow:

- Enhancements to the register set
  - — The PID7v-603e adds two new bits to the HID0 register:
    - – The address bus enable (ABE) bit, bit 28, gives the PID7v-603e microprocessor the ability to broadcast **dcbf**, **dcbi**, and **dcbst** onto the 60x bus.
    - – The instruction fetch enable M (IFEM) bit, bit 24, allows the PID7v-603e to reflect the value of the M-bit onto the 60x bus during instruction translation.
  - — The Run_N counter register (Run_N) has been extended from 16 to 32 bits.
- Enhancements to 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 utilizing an external copyback L2 cache.
  - — All of the cache control instructions (**icbi**, **dcbi**, **dcbf**, and **dcbst**, excluding **dcbz**) require that the HID0[ABE] configuration bit be enabled in order to execute.
- Exceptions
  - — The PID7v-603e now offers hardware support for misaligned little-endian accesses. Little-endian load/store accesses that are not on a word boundary, with the exception of strings and multiples, generate exceptions under the same circumstances as big-endian accesses.
  - — The PID7v-603e removed misalignment support for **eciwx** and **ecowx** graphics instructions.These instructions cause an alignment exception if the access is not on a word boundary.

- Bus clock—New bus multipliers of 4.5x, 5x, 5.5x, and 6x that are selected by the unused encodings of the PLL_CFG[0–3]. Bus multipliers of 1x and 1.5x are not supported by PID7v-603e.
- Power management—Internal voltage supply changed from 3.3 volts to 2.5 volts. The core logic of the chip now uses a 2.5-volt supply.
- Signals—The Run_N counter, which affects the JTAG/COP, has been extended from 16 bits to 32 bits.
- Instruction timing
  - The integer divide instructions **divwu**[**o**][**.**] and **divw**[**o**][**.**] execute in 20 clock cycles; execution of these instructions in the PID6-603e requires 37 clock cycles.
  - Support for single-cycle store
  - An adder/comparator added to system register unit that allows dispatch and execution of multiple integer add and compare instructions on each cycle.

Figure 1-1 provides a block diagram of the 603e that illustrates how the execution units—IU, FPU (not supported by the EC603e microprocessor), BPU, LSU, and SRU—operate independently and in parallel. Note that this is a conceptual diagram and does not attempt to show how these features are physically implemented on the chip. For more information on the execution units, refer to *PowerPC 603e RISC Microprocessor Technical Summary.*

The 603e provides address translation and protection facilities, including an ITLB, DTLB, and instruction and data BAT arrays. Instruction fetching and issuing is handled in the instruction unit. Translation of addresses for cache or external memory accesses are handled by the MMUs. Both units are discussed in more detail in Sections 1.1.3, “Instruction Unit,” and 1.1.5.1, “Memory Management Units (MMUs).”



* Note that the EC603e microprocessor does not support the floating-point unit or the floating-point register file.

Figure 1-1. Block Diagram

1.1.2 System Design and Programming Considerations

The 603e is built upon the low power dissipation, low cost and high performance attributes of the 603 while providing the system designer additional capabilities through higher processor clock speeds (to 100 MHz), increases in cache size (16-Kbyte instruction and data caches) and set associativity (four-way), and greater system clock flexibility. The following subsections describe the differences between the 603 and the 603e that affect the system designer and programmer already familiar with the operation of the 603.

The design enhancements to the 603e are described in the following sections as changes that can require a modification to the hardware or software configuration of a system designed for the 603.

1.1.2.1 Hardware Features

The following hardware features of the 603e may require system designers to modify systems designed for the 603.

1.1.2.1.1 Replacement of $\overline{\text{XATS}}$ Signal by CSE1 Signal

The 603e employs four-way set associativity for both the instruction and data caches, in place of the two-way set associativity used in the 603. This change requires the use of an additional cache set entry (CSE1) signal to indicate which member of the cache set is being loaded during a cache line fill. The CSE1 signal on the 603e is in the same pin location as the $\overline{\text{XATS}}$ signal on the 603. Note that the $\overline{\text{XATS}}$ signal is no longer needed by the 603e because support for access to direct-store segments has been removed.

Table 1-1 shows the CSE[0–1] signal encoding indicating the cache set element selected during a cache load operation.

Table 1-1. CSE[0–1] Signals

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

1.1.2.1.2 Addition of Half-Clock Bus Multipliers

Some of the reserved clock configuration signal settings of the 603 are redefined to allow more flexible selection of higher internal and bus clock frequencies. The 603e provides programmable internal processor clock rates of 1x, 1.5x, 2x, 2.5x, 3x, 3.5x, and 4x multiples of the externally supplied clock frequency. For additional information, refer to the appropriate device-specific hardware specifications.

1.1.2.2 Software Features

The features of the 603e described in the following sections affect software originally written for the 603.

1.1.2.2.1 16-Kbyte Instruction and Data Caches

The instruction and data caches of the 603e are 16 Kbytes in size, compared to the 8-Kbyte instruction and data caches of the 603. The increase in cache size may require modification of cache flush routines. The increase in cache size is also reflected in four-way set associativity of the instruction and data caches in place of the two-way set associativity in the 603.

1.1.2.2.2 Clock Configuration Available in HID1 Register

Bits 0–3 in the new HID1 register (SPR 1009) provides software read-only access to the configuration of the PLL_CFG signals. The HID1 register is not implemented in the 603.

1.1.2.2.3 Performance Enhancements

The following enhancements provide improved performance without any required changes to software (other than compiler optimization) or hardware designed for the 603:

- Support for single-cycle store.
- Addition of adder/comparator in system register unit allows dispatch and execution of multiple integer add and compare instructions on each cycle.
- Addition of a key bit (bit 12) to SRR1 to provide information about memory protection violations prior to page table search operations. This key bit is set when the combination of the settings in the appropriate Kx bit in the segment register and the MSR[PR] bit indicates that when the PP bits in the PTE are set to either 00 or 01, a protection violation exists; if this is the case for a data write operation with a DTLB miss, the changed (C) bit in the page tables should not be updated (see Table 1-2). This reduces the time required to execute the page table search routine since the software no longer has to explicitly read both the Kx and MSR[PR] bits to determine whether a protection violation exists before updating the C bit.

Table 1-2. Generated SRR1 [Key] Bit

Segment Register [Ks, Kp]	MSR[PR]	SRR1[Key] Generated on DTLB Misses
0x	0	0
x0	1	0
1x	0	1
x1	1	1

Note that this key bit indicates a protection violation if the PTE[pp] bits are either 00 or 01.

1.1.3 Instruction Unit

As shown in Figure 1-1, the 603e instruction unit, which contains a fetch unit, instruction queue, dispatch unit, and BPU, provides centralized control of instruction flow to the execution units. The instruction unit determines the address of the next instruction to be fetched based on information from the sequential fetcher and from the BPU.

The instruction unit fetches the instructions from the instruction cache into the instruction queue. The BPU extracts branch instructions from the fetcher and uses static branch prediction on unresolved conditional branches to allow the instruction unit to fetch instructions from a predicted target instruction stream while a conditional branch is evaluated. The BPU folds out branch instructions for unconditional branches or conditional branches unaffected by instructions in progress in the execution pipeline.

Instructions issued beyond a predicted branch do not complete execution until the branch is resolved, preserving the programming model of sequential execution. If any of these instructions are to be executed in the BPU, they are decoded but not issued. Instructions to be executed by the FPU, IU, LSU, and SRU are issued and allowed to complete up to the register write-back stage. (Note that the FPU is not supported on the EC603e microprocessor.) Write-back is allowed when a correctly predicted branch is resolved, and instruction execution continues without interruption along the predicted path.

If branch prediction is incorrect, the instruction unit flushes all predicted path instructions, and instructions are issued from the correct path.

1.1.3.1 Instruction Queue and Dispatch Unit

The instruction queue (IQ), shown in Figure 1-1, holds as many as six instructions and loads up to two instructions from the instruction unit during a single cycle. The instruction fetch unit continuously loads as many instructions as space in the IQ allows. Instructions are dispatched to their respective execution units from the dispatch unit at a maximum rate of two instructions per cycle. Dispatching is facilitated to the IU, FPU (not supported on the EC603e microprocessor), LSU, and SRU by the provision of a reservation station at each unit. The dispatch unit performs source and destination register dependency checking, determines dispatch serializations, and inhibits subsequent instruction dispatching as required.

For a more detailed overview of instruction dispatch, see Section 1.3.6, “Instruction Timing.”

1.1.3.2 Branch Processing Unit (BPU)

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

The BPU uses a bit in the instruction encoding to predict the direction of the conditional branch. Therefore, when an unresolved conditional branch instruction is encountered, the

603e fetches instructions from the predicted target stream until the conditional branch is resolved.

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

1.1.4 Independent Execution Units

The PowerPC architecture's support for independent execution units allows implementation of processors with out-of-order instruction execution. For example, because branch instructions do not depend on GPRs or FPRs, branches can often be resolved early, eliminating stalls caused by taken branches.

In addition to the BPU, the 603e provides four other execution units and a completion unit, which are described in the following sections.

1.1.4.1 Integer Unit (IU)

The IU executes all integer instructions. The IU executes one integer instruction at a time, performing computations with its arithmetic logic unit (ALU), multiplier, divider, and XER register. Most integer instructions are single-cycle instructions. Thirty-two general-purpose registers are provided to support integer operations. Stalls due to contention for GPRs are minimized by the automatic allocation of rename registers. The 603e writes the contents of the rename registers to the appropriate GPR when integer instructions are retired by the completion unit.

1.1.4.2 Floating-Point Unit (FPU)

The FPU (not supported by the EC603e microprocessor) contains a single-precision multiply-add array and the floating-point status and control register (FPSCR). The multiply-add array allows the 603e to efficiently implement multiply and multiply-add operations. The FPU is pipelined so that single-precision instructions and double-precision instructions can be issued back-to-back. Thirty-two floating-point registers are provided to support floating-point operations. Stalls due to contention for FPRs are minimized by the automatic allocation of rename registers. The 603e writes the contents of the rename registers to the appropriate FPR when floating-point instructions are retired by the completion unit.

The 603e supports all IEEE 754 floating-point data types (normalized, denormalized, NaN, zero, and infinity) in hardware, eliminating the latency incurred by software exception routines. (The term, ‘exception’ is also referred to as ‘interrupt’ in the architecture specification.)

1.1.4.3 Load/Store Unit (LSU)

The LSU executes all load and store instructions and provides the data transfer interface between the GPRs, FPRs, and the cache/memory subsystem. The LSU calculates effective addresses, performs data alignment, and provides sequencing for load/store string and multiple instructions. (Note that the EC603e microprocessor does not support the floating-point register file.)

Load and store instructions are issued and translated in program order; however, the actual memory accesses can occur out of order. Synchronizing instructions are provided to enforce strict ordering.

Cacheable loads, when free of data dependencies, execute in an out-of-order manner with a maximum throughput of one per cycle and a two-cycle total latency. Data returned from the cache is held in a rename register until the completion logic commits the value to a GPR or FPR (not supported by the EC603e microprocessor). Stores cannot be executed in a predicted manner and are held in the store queue until the completion logic signals that the store operation is to be completed to memory. The 603e executes store instructions with a maximum throughput of one per cycle and a three-cycle total latency. The time required to perform the actual load or store operation varies depending on whether the operation involves the cache, system memory, or an I/O device.

1.1.4.4 System Register Unit (SRU)

The SRU executes various system-level instructions, including condition register logical operations and move to/from special-purpose register instructions, and also executes integer add/compare instructions. In order to maintain system state, most instructions executed by the SRU are completion-serialized; that is, the instruction is held for execution in the SRU until all prior instructions issued have completed. Results from completion-serialized instructions executed by the SRU are not available or forwarded for subsequent instructions until the instruction completes.

1.1.4.5 Completion Unit

The completion unit tracks instructions from dispatch through execution, and then retires, or “completes” them in program order. Completing an instruction commits the 603e to any architectural register changes caused by that instruction. In-order completion ensures the correct architectural state when the 603e must recover from a mispredicted branch or any exception.

Instruction state and other information required for completion is kept in a first-in-first-out (FIFO) queue of five completion buffers. A single completion buffer is allocated for each instruction once it enters the dispatch unit. An available completion buffer is a required resource for instruction dispatch; if no completion buffers are available, instruction dispatch stalls. A maximum of two instructions per cycle are completed in order from the queue.

1.1.5 Memory Subsystem Support

The 603e provides support for cache and memory management through dual instruction and data memory management units. The 603e also provides dual 16-Kbyte instruction and data caches, and an efficient processor bus interface to facilitate access to main memory and other bus subsystems. The memory subsystem support functions are described in the following subsections.

1.1.5.1 Memory Management Units (MMUs)

The 603e's MMUs support up to 4 Petabytes (2^{52}) of virtual memory and 4 Gigabytes (2^{32}) of physical memory (referred to as real memory in the architecture specification) for instruction and data. The MMUs also control access privileges for these spaces on block and page granularities. Referenced and changed status is maintained by the processor for each page to assist implementation of a demand-paged virtual memory system. A key bit is implemented to provide information about memory protection violations prior to page table search operations.

The LSU calculates effective addresses for data loads and stores, performs data alignment to and from cache memory, and provides the sequencing for load and store string and multiple word instructions. The instruction unit calculates the effective addresses for instruction fetching.

After an address is generated, the higher-order bits of the effective address are translated by the appropriate MMU into physical address bits. Simultaneously, the lower-order address bits (that are untranslated and therefore, considered both logical and physical), are directed to the on-chip caches where they form the index into the four-way set-associative tag array. After translating the address, the MMU passes the higher-order bits of the physical address to the cache, and the cache lookup completes. For caching-inhibited accesses or accesses that miss in the cache, the untranslated lower-order address bits are concatenated with the translated higher-order address bits; the resulting 32-bit physical address is then used by the memory unit and the system interface, which accesses external memory.

The MMU also directs the address translation and enforces the protection hierarchy programmed by the operating system in relation to the supervisor/user privilege level of the access and in relation to whether the access is a load or store.

For instruction accesses, the MMU performs an address lookup in both the 64 entries of the ITLB, and in the IBAT array. If an effective address hits in both the ITLB and the IBAT array, the IBAT array translation takes priority. Data accesses cause a lookup in the DTLB and DBAT array for the physical address translation. In most cases, the physical address translation resides in one of the TLBs and the physical address bits are readily available to the on-chip cache.

When the physical address translation misses in the TLBs, the 603e provides hardware assistance for software to perform a search of the translation tables in memory. The hardware assist consists of the following features:

- Automatic storage of the missed effective address in the IMISS and DMISS registers
- Automatic generation of the primary and secondary hashed real address of the page table entry group (PTEG), which are readable from the HASH1 and HASH2 register locations.

  The HASH data is generated from the contents of the IMISS or DMISS register. Which register is selected depends on which miss (instruction or data) was last acknowledged.
- Automatic generation of the first word of the page table entry (PTE) for which the tables are being searched
- A real page address (RPA) register that matches the format of the lower word of the PTE
- Two TLB access instructions (**tlbli** and **tlbld**) that are used to load an address translation into the instruction or data TLBs
- Shadow registers for GPRs 0–3 that allow miss code to execute without corrupting the state of any of the existing GPRs.

  These shadow registers are only used for servicing a TLB miss.

See Section 1.3.5.2, "Implementation-Specific Memory Management," for more information about memory management for the 603e.

1.1.5.2 Cache Units

The 603e provides independent 16-Kbyte, four-way set-associative instruction and data caches. The cache line size is 32 bytes in length. The caches are designed to adhere to a write-back policy, but the 603e allows control of cacheability, write policy, and memory coherency at the page and block levels. The caches use a least recently used (LRU) replacement policy.

As shown in Figure 1-1, the caches provide a 64-bit interface to the instruction fetch unit and load/store unit. The surrounding logic selects, organizes, and forwards the requested information to the requesting unit. Write operations to the cache can be performed on a byte basis, and a complete read-modify-write operation to the cache can occur in each cycle.

The load/store and instruction fetch units provide the caches with the address of the data or instruction to be fetched. In the case of a cache hit, the cache returns two words to the requesting unit.

Since the 603e data cache tags are single ported, simultaneous load or store and snoop accesses cause resource contention. Snoop accesses have the highest priority and are given first access to the tags, unless the snoop access coincides with a tag write, in which case the snoop is retried and must re-arbitrate for access to the cache. Loads or stores that are deferred due to snoop accesses are executed on the clock cycle following the snoop.

1.1.6 Processor Bus Interface

Because the caches on the 603e are on-chip, write-back caches, the predominant type of transaction for most applications is burst-read memory operations, followed by burst-write memory operations, and single-beat (noncacheable or write-through) memory read and write operations. Additionally, there can be address-only operations, variants of the burst and single-beat operations, (for example, global memory operations that are snooped and atomic memory operations), and address retry activity (for example, when a snooped read access hits a modified line in the cache).

Memory accesses can occur in single-beat (1–8 bytes) and four-beat burst (32 bytes) data transfers when the bus is configured as 64 bits, and in single-beat (1–4 bytes), two-beat (8 bytes), and eight-beat (32 bytes) data transfers when the bus is configured as 32 bits. The address and data buses operate independently to support pipelining and split transactions during memory accesses. The 603e can pipeline its own transactions to a depth of one level.

Access to the system interface is granted through an external arbitration mechanism that allows devices to compete for bus mastership. This arbitration mechanism is flexible, allowing the 603e to be integrated into systems that implement various fairness and bus parking procedures to avoid arbitration overhead.

Typically, memory accesses are weakly ordered—sequences of operations, including load/store string and multiple instructions, do not necessarily complete in the order they begin—maximizing the efficiency of the bus without sacrificing coherency of the data. The 603e allows read operations to precede store operations (except when a dependency exists, or in cases where a non-cacheable access is performed), and provides support for a write operation to proceed a previously queued read data tenure (for example, allowing a snoop push to be enveloped by the address and data tenures of a read operation). Because the processor can dynamically optimize run-time ordering of load/store traffic, overall performance is improved.

1.1.7 System Support Functions

The 603e implements several support functions that include power management, time base/decrementer registers for system timing tasks, an IEEE 1149.1(JTAG)/common on-chip processor (COP) test interface, and a phase-locked loop (PLL) clock multiplier. These system support functions are described in the following subsections.

1.1.7.1 Power Management

The 603e provides four power modes selectable by setting the appropriate control bits in the machine state register (MSR) and hardware implementation register 0 (HID0) registers. The four power modes are as follows:

- Full-power–This is the default power state of the 603e. The 603e is fully powered and the internal functional units are operating at the full processor clock speed. If the dynamic power management mode is enabled, functional units that are idle will automatically enter a low-power state without affecting performance, software execution, or external hardware.
- Doze–All the functional units of the 603e are disabled except for the time base/decrementer registers and the bus snooping logic. When the processor is in doze mode, an external asynchronous interrupt, a system management interrupt, a decrementer exception, a hard or soft reset, or machine check brings the 603e into the full-power state. The 603e in doze mode maintains the PLL in a fully powered state and locked to the system external clock input (SYSCLK) so a transition to the full-power state takes only a few processor clock cycles.
- Nap–The nap mode further reduces power consumption by disabling bus snooping, leaving only the time base register and the PLL in a powered state. The 603e returns to the full-power state upon receipt of an external asynchronous interrupt, a system management interrupt, a decrementer exception, a hard or soft reset, or a machine check input ($\overline{\text{MCP}}$) signal. A return to full-power state from a nap state takes only a few processor clock cycles.
- Sleep–Sleep mode reduces power consumption to a minimum by disabling all internal functional units, after which external system logic may disable the PLL and SYSCLK. Returning the 603e to the full-power state requires the enabling of the PLL and SYSCLK, followed by the assertion of an external asynchronous interrupt, a system management interrupt, a hard or soft reset, or a machine check input ($\overline{\text{MCP}}$) signal after the time required to relock the PLL.

The PID7v-603e implementation offers the following enhancements to the 603e family:

- Lower-power design
- 2.5-volt core and 3.3-volt I/O

1.1.7.2 Time Base/Decrementer

The time base is a 64-bit register (accessed as two 32-bit registers) that is incremented once every four bus clock cycles; external control of the time base is provided through the time base enable (TBEN) signal. The decrementer is a 32-bit register that generates a decrementer interrupt exception after a programmable delay. The contents of the decrementer register are decremented once every four bus clock cycles, and the decrementer exception is generated as the count passes through zero.

1.1.7.3 IEEE 1149.1 (JTAG)/COP Test Interface

The 603e provides IEEE 1149.1 and COP functions for facilitating board testing and chip debug. The IEEE 1149.1 test interface provides a means for boundary-scan testing the 603e and the board to which it is attached. The COP function shares the IEEE 1149.1 test port, provides a means for executing test routines, and facilitates chip and software debugging.

1.1.7.4 Clock Multiplier

The internal clocking of the 603e is generated from and synchronized to the external clock signal, SYSCLK, by means of a voltage-controlled oscillator-based PLL. The PLL provides programmable internal processor clock rates of 1x, 1.5x, 2x, 2.5x, 3x, 3.5x, and 4x multiples of the externally supplied clock frequency. The bus clock is the same frequency and is synchronous with SYSCLK. The configuration of the PLL can be read by software from the hardware implementation register 1 (HID1).

1.2 PowerPC Architecture Implementation

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

- PowerPC user instruction set architecture (UISA)—Defines the base user-level instruction set, user-level registers, data types, floating-point exception model, memory models for a uniprocessor environment, and programming model for a uniprocessor environment.
- PowerPC virtual environment architecture (VEA)—Describes the memory model for a multiprocessor environment, defines cache control instructions, and describes other aspects of virtual environments. Implementations that conform to the VEA also adhere to the UISA, but may not necessarily adhere to the OEA.
- PowerPC operating environment architecture (OEA)—Defines the memory management model, supervisor-level registers, synchronization requirements, and the exception model. Implementations that conform to the OEA also adhere to the UISA and the VEA.

The PowerPC architecture allows a wide range of designs for such features as cache and system interface implementations.

1.3 Implementation-Specific Information

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

This section describes the PowerPC architecture in general, and specific details about the implementation of the 603e as a low-power, 32-bit member of the PowerPC processor family. The main topics addressed are as follows:

- Section 1.3.1, “Programming Model,” describes the registers for the operating environment architecture common among PowerPC processors and describes the programming model. It also describes the additional registers that are unique to the 603e.
- Section 1.3.2, “Instruction Set and Addressing Modes,” describes the PowerPC instruction set and addressing modes for the PowerPC operating environment architecture, and defines and describes the PowerPC instructions implemented in the 603e.
- Section 1.3.3, “Cache Implementation,” describes the cache model that is defined generally for PowerPC processors by the virtual environment architecture. It also provides specific details about the 603e cache implementation.
- Section 1.3.4, “Exception Model,” describes the exception model of the PowerPC operating environment architecture and the differences in the 603e exception model.
- Section 1.3.5, “Memory Management,” describes generally the conventions for memory management among the PowerPC processors. This section also describes the 603e’s implementation of the 32-bit PowerPC memory management specification.
- Section 1.3.6, “Instruction Timing,” provides a general description of the instruction timing provided by the superscalar, parallel execution supported by the PowerPC architecture and the 603e.
- Section 1.3.7, “System Interface,” describes the signals implemented on the 603e.

The 603e is a high-performance, superscalar PowerPC microprocessor. The PowerPC architecture allows optimizing compilers to schedule instructions to maximize performance through efficient use of the PowerPC instruction set and register model. The multiple, independent execution units allow compilers to optimize instruction throughput. Compilers that take advantage of the flexibility of the PowerPC architecture can additionally optimize system performance of the PowerPC processors.

The following sections summarize the features of the 603e, including both those that are defined by the architecture and those that are unique to the various 603e implementations.

Specific features of the 603e are listed in Section 1.1.1, “Features.”

1.3.1 Programming Model

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

PowerPC processors have two levels of privilege—supervisor mode of operation (typically used by the operating system) and user mode of operation (used by the application software). The programming models incorporate 32 GPRs, 32 FPRs (not supported by the EC603e microprocessor), special-purpose registers (SPRs), and several miscellaneous registers. Each PowerPC microprocessor also has its own unique set of hardware implementation (HID) registers.

Having access to privileged instructions, registers, and other resources allows the operating system to control the application environment (providing virtual memory and protecting operating-system and critical machine resources). Instructions that control the state of the processor, the address translation mechanism, and supervisor registers can be executed only when the processor is operating in supervisor mode.

Figure 1-2 shows all the 603e registers available at the user and supervisor level. The numbers to the right of the SPRs indicate the number that is used in the syntax of the instruction operands to access the register.

The following subsections describe the PID7v-603e implementation-specific features as they apply to registers.

1.3.1.1 Processor Version Register (PVR)

The processor version number is 6 for the PID6-603e and 7 for the PID7v-603e. The processor revision level starts at 0x0100 and changes for each chip revision. The revision level is updated on all silicon revisions.

1.3.1.2 Hardware Implementation Register 0 (HID0)

PID7v-603e (designated by PVR level 0x0200) defines additional bits in the hardware implementation register 0 (HID0), a supervisor-level register that provides the means for enabling the 603e's checkstops and features, and allows software to read the configuration of the PLL configuration signals.

The HID0 bits with changed bit assignments are shown in Table 1-3. The HID0 bits that are not shown here are implemented as they are in Section 2.1.2.1, "Hardware Implementation Registers (HID0 and HID1)."

Table 1-3. Additional/Changed HID0 Bits

Bit(s)	Description
24	Instruction fetch enable M (IFEM) bit—Enables the M bit on the bus. Used for instruction fetches.
25–26	Reserved
28	Address broadcast enable (ABE)—This configuration bit allows for the broadcast of **dcbf**, **dcbi**, and **dcbst** on the bus. Note that these cache control instruction broadcasts are not snooped by the PID7v-603e. Refer to Section 1.3.3, "Cache Implementation," for more information.
29–30	Reserved

1.3.1.3 Run_N Counter Register (Run_N)

The 33-bit Run_N counter register is unique to the PID7v-603e. The Run_N counter is used by the COP to control the number of processor cycles that the processor runs before halting. The most-significant 32 bits form a 32-bit counter. The function of the least-significant bit remains unchanged.

1.3.1.4 General-Purpose Registers (GPRs)

The PowerPC architecture defines 32 user-level, general-purpose registers (GPRs). These registers are either 32 bits wide in 32-bit PowerPC microprocessors and 64 bits wide in 64-bit PowerPC microprocessors. The GPRs serve as the data source or destination for all integer instructions.

1.3.1.5 Floating-Point Registers (FPRs)

The PowerPC architecture also defines 32 user-level, 64-bit floating-point registers (FPRs) (not supported by the EC603e microprocessor). The FPRs serve as the data source or destination for floating-point instructions. These registers can contain data objects of either single- or double-precision floating-point formats.

1.3.1.6 Condition Register (CR)

The CR is a 32-bit user-level register that consists of eight four-bit fields that reflect the results of certain operations, such as move, integer and floating-point compare, arithmetic, and logical instructions, and provide a mechanism for testing and branching.

1.3.1.7 Floating-Point Status and Control Register (FPSCR)

The floating-point status and control register (FPSCR) is a user-level register that contains all exception signal bits, exception summary bits, exception enable bits, and rounding control bits needed for compliance with the IEEE 754 standard. (Note that this is not supported by the EC603e microprocessor.)

1.3.1.8 Machine State Register (MSR)

The machine state register (MSR) is a supervisor-level register that defines the state of the processor. The contents of this register are saved when an exception is taken and restored when the exception handling completes. The 603e implements the MSR as a 32-bit register; 64-bit PowerPC processors implement a 64-bit MSR. To ensure proper operation of the EC603e microprocessor, the MSR[FP] bit should remain cleared to zero.

1.3.1.9 Segment Registers (SRs)

For memory management, 32-bit PowerPC microprocessors implement sixteen 32-bit segment registers (SRs). To speed access, the 603e implements the segment registers as two arrays; a main array (for data memory accesses) and a shadow array (for instruction memory accesses). Loading a segment entry with the Move to Segment Register (**mtsr**) instruction loads both arrays.

1.3.1.10 Special-Purpose Registers (SPRs)

The PowerPC operating environment architecture defines numerous special-purpose registers that serve a variety of functions, such as providing controls, indicating status, configuring the processor, and performing special operations. During normal execution, a program can access the registers, shown in Figure 2-1, depending on the program's access privilege (supervisor or user, determined by the privilege-level (PR) bit in the MSR). Note that registers such as the GPRs and FPRs (not supported by the EC603e microprocessor) are accessed through operands that are part of the instructions. Access to registers can be explicit (that is, through the use of specific instructions for that purpose such as Move to Special-Purpose Register (**mtspr**) and Move from Special-Purpose Register (**mfspr**) instructions) or implicit, as the part of the execution of an instruction. Some registers are accessed both explicitly and implicitly

In the 603e, all SPRs are 32 bits wide.

1.3.1.10.1 User-Level SPRs

The following 603e SPRs are accessible by user-level software:

- Link register (LR)—The link register can be used to provide the branch target address and to hold the return address after branch and link instructions. The LR is 32 bits wide in 32-bit implementations.
- Count register (CTR)—The CTR is decremented and tested automatically as a result of branch-and-count instructions. The CTR is 32 bits wide in 32-bit implementations.
- XER register—The 32-bit XER contains the summary overflow bit, integer carry bit, overflow bit, and a field specifying the number of bytes to be transferred by a Load String Word Indexed (**lswx**) or Store String Word Indexed (**stswx**) instruction.

1.3.1.10.2 Supervisor-Level SPRs

The 603e also contains SPRs that can be accessed only by supervisor-level software. These registers consist of the following:

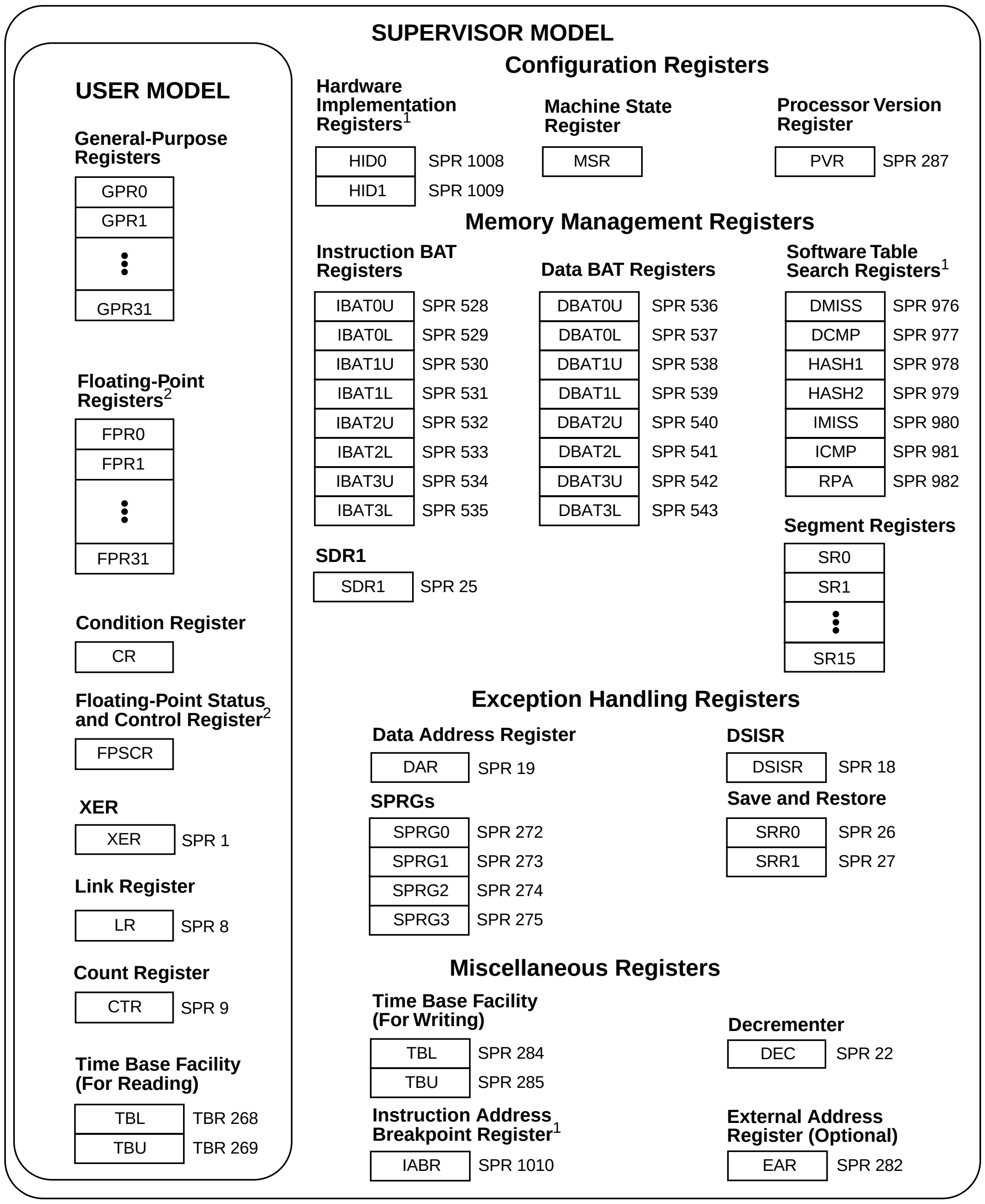
- The 32-bit DSISR defines the cause of data access and alignment exceptions.
- The data address register (DAR) is a 32-bit register that holds the address of an access after an alignment or DSI exception.
- Decrementer register (DEC) is a 32-bit decrementing counter that provides a mechanism for causing a decrementer exception after a programmable delay.
- The 32-bit SDR1 specifies the page table format used in virtual-to-physical address translation for pages. (Note that physical address is referred to as real address in the architecture specification.)
- The machine status save/restore register 0 (SRR0) is a 32-bit register that is used by the 603e for saving the address of the instruction that caused the exception, and the address to return to when a Return from Interrupt (**rfi**) instruction is executed.

- The machine status save/restore register 1 (SRR1) is a 32-bit register used to save machine status on exceptions and to restore machine status when an **rfi** instruction is executed.
- The 32-bit SPRG0–SPRG3 registers are provided for operating system use.
- The external access register (EAR) is a 32-bit register that controls access to the external control facility through the External Control In Word Indexed (**eciwx**) and External Control Out Word Indexed (**ecowx**) instructions.
- The time base register (TB) is a 64-bit register that maintains the time of day and operates interval timers. The TB consists of two 32-bit fields—time base upper (TBU) and time base lower (TBL).
- The processor version register (PVR) is a 32-bit, read-only register that identifies the version (model) and revision level of the PowerPC processor.
- Block address translation (BAT) arrays—The PowerPC architecture defines 16 BAT registers, divided into four pairs of data BATs (DBATs) and four pairs of instruction BATs (IBATs). See Figure 2-1 for a list of the SPR numbers for the BAT arrays.

The following supervisor-level SPRs are implementation-specific to the 603e:

- The DMISS and IMISS registers are read-only registers that are loaded automatically upon an instruction or data TLB miss.
- The HASH1 and HASH2 registers contain the physical addresses of the primary and secondary page table entry groups (PTEGs).
- The ICMP and DCMP registers contain a duplicate of the first word in the page table entry (PTE) for which the table search is looking.
- The required physical address (RPA) register is loaded by the processor with the second word of the correct PTE during a page table search.
- The hardware implementation (HID0 and HID1) registers provide the means for enabling the 603e’s checkstops and features, and allows software to read the configuration of the PLL configuration signals.
- The instruction address breakpoint register (IABR) is loaded with an instruction address that is compared to instruction addresses in the dispatch queue. When an address match occurs, an instruction address breakpoint exception is generated.

Figure 2-1 shows all the 603e registers available at the user and supervisor level. The numbers to the right of the SPRs indicate the number that is used in the syntax of the instruction operands to access the register.



Notes: [1]These registers are 603e–specific (PID6-603e and PID7v-603e) registers. They may not be supported by other PowerPC processors.

[2]Not supported on the EC603e microprocessor.

Figure 1-2. Programming Model—Registers

1.3.2 Instruction Set and Addressing Modes

The following subsections describe the PowerPC instruction set and addressing modes in general.

1.3.2.1 PowerPC Instruction Set and Addressing Modes

All PowerPC instructions are encoded as single-word (32-bit) opcodes. Instruction formats are consistent among all instruction types, permitting efficient decoding to occur in parallel with operand accesses. This fixed instruction length and consistent format greatly simplifies instruction pipelining.

1.3.2.1.1 PowerPC Instruction Set

The PowerPC instructions are divided into the following categories:

- Integer instructions—These include computational and logical instructions.
  - Integer arithmetic instructions
  - Integer compare instructions
  - Integer logical instructions
  - Integer rotate and shift instructions
- Floating-point instructions—These include floating-point computational instructions, as well as instructions that affect the FPSCR. (Note that these instructions are not implemented on the EC603e microprocessor.)
  - Floating-point arithmetic instructions
  - Floating-point multiply/add instructions
  - Floating-point rounding and conversion instructions
  - Floating-point compare instructions
  - Floating-point status and control instructions
- Load/store instructions—These include integer and floating-point load and store instructions.
  - Integer load and store instructions
  - Integer load and store multiple instructions
  - Floating-point load and store (not implemented on the EC603e microprocessor)
  - Primitives used to construct atomic memory operations (**lwarx** and **stwcx.** instructions)
- Flow control instructions—These include branching instructions, condition register logical instructions, trap instructions, and other instructions that affect the instruction flow.
  - Branch and trap instructions
  - Condition register logical instructions

- Processor control instructions—These instructions are used for synchronizing memory accesses and management of caches, TLBs, and the segment registers.
  - Move to/from SPR instructions
  - Move to/from MSR
  - Synchronize
  - Instruction synchronize
- Memory control instructions—These instructions provide control of caches, TLBs, and segment registers.
  - Supervisor-level cache management instructions
  - User-level cache instructions
  - Segment register manipulation instructions
  - Translation lookaside buffer management instructions

Note that this grouping of the instructions does not indicate which execution unit executes a particular instruction or group of instructions.

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

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

PowerPC processors follow the program flow when they are in the normal execution state. However, the flow of instructions can be interrupted directly by the execution of an instruction or by an asynchronous event. Either kind of exception may cause one of several components of the system software to be invoked.

1.3.2.1.2 Calculating Effective Addresses

The effective address (EA) is the 32-bit address computed by the processor when executing a memory access or branch instruction or when fetching the next sequential instruction.

The PowerPC architecture supports two simple memory addressing modes:

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

These simple addressing modes allow efficient address generation for memory accesses. Calculation of the effective address for aligned transfers occurs in a single clock cycle.

For a memory access instruction, if the sum of the effective address and the operand length exceeds the maximum effective address, the memory operand is considered to wrap around from the maximum effective address to effective address 0.

Effective address computations for both data and instruction accesses use 32-bit unsigned binary arithmetic. A carry from bit 0 is ignored in 32-bit implementations.

1.3.2.2 Implementation-Specific Instruction Set

The 603e instruction set is defined as follows:

- The 603e provides hardware support for all 32-bit PowerPC instructions.
- The 603e provides two implementation-specific instructions used for software table search operations following TLB misses:
  - Load Data TLB Entry (**tlbld**)
  - Load Instruction TLB Entry (**tlbli**)
- The 603e implements the following instructions which are defined as optional by the PowerPC architecture:
  - External Control In Word Indexed (**eciwx**)
  - External Control Out Word Indexed (**ecowx**)
  - Floating Select (**fsel**)
    (Not supported by the EC603e microprocessor)
  - Floating Reciprocal Estimate Single-Precision (**fres**)
    (Not supported by the EC603e microprocessor)
  - Floating Reciprocal Square Root Estimate (**frsqrte**)
    (Not supported by the EC603e microprocessor)
  - Store Floating-Point as Integer Word (**stfiwx**)
    (Not supported by the EC603e microprocessor)

1.3.3 Cache Implementation

The following subsections describe the general cache characteristics as implemented in the PowerPC architecture, and the 603e implementation, specifically. PID7v-603e specific information is noted where applicable.

1.3.3.1 PowerPC Cache Characteristics

The PowerPC architecture does not define hardware aspects of cache implementations. For example, some PowerPC processors, including the 603e, have separate instruction and data caches (Harvard architecture), while others, such as the PowerPC 601® microprocessor, implement a unified cache.

PowerPC microprocessors control the following memory access modes on a page or block basis:

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

Note that in the 603e, a cache block is defined as eight words. The VEA defines cache management instructions that provide a means by which the application programmer can affect the cache contents.

1.3.3.2 Implementation-Specific Cache Implementation

The 603e has two 16-Kbyte, four-way set-associative (instruction and data) caches. The caches are physically addressed, and the data cache can operate in either write-back or write-through mode as specified by the PowerPC architecture.

The data cache is configured as 128 sets of four blocks each. Each block consists of 32 bytes, two state bits, and an address tag. The two state bits implement the three-state MEI (modified/exclusive/invalid) protocol. Each block contains eight 32-bit words. Note that the PowerPC architecture defines the term ‘block’ as the cacheable unit. For the 603e, the block size is equivalent to a cache line. A block diagram of the data cache organization is shown in Figure 1-3.

The instruction cache also consists of 128 sets of four blocks, and each block consists of 32 bytes, an address tag, and a valid bit. The instruction cache may not be written to except through a block fill operation. In the PID7v-603e, the instruction cache is blocked only until the critical load completes. The PID7v-603e supports instruction fetching from other instruction cache lines following the forwarding of the critical first double word of a cache line load operation. Successive instruction fetches from the cache line being loaded are forwarded, and accesses to other instruction cache lines can proceed during the cache line load operation. The instruction cache is not snooped, and cache coherency must be maintained by software. A fast hardware invalidation capability is provided to support cache maintenance. The organization of the instruction cache is very similar to the data cache shown in Figure 1-3.

Each cache block contains eight contiguous words from memory that are loaded from an 8-word boundary (that is, bits A27–A31 of the effective addresses are zero); thus, a cache block never crosses a page boundary. Misaligned accesses across a page boundary can incur a performance penalty.

The 603e’s cache blocks are loaded in four beats of 64 bits each when the 603e is configured with a 64-bit data bus; when the 603e is configured with a 32-bit bus, cache block loads are performed with eight beats of 32 bits each. The burst load is performed as critical double word first. The data cache is blocked to internal accesses until the load completes; the instruction cache allows sequential fetching during a cache block load. In the PID7v-603e, the critical double word is simultaneously written to the cache and forwarded to the requesting unit, thus minimizing stalls due to load delays.

To ensure coherency among caches in a multiprocessor (or multiple caching-device) implementation, the 603e implements the MEI protocol. These three states, modified, exclusive, and invalid, indicate the state of the cache block as follows:

- Modified—The cache block is modified with respect to system memory; that is, data for this address is valid only in the cache and not in system memory.
- Exclusive—This cache block holds valid data that is identical to the data at this address in system memory. No other cache has this data.
- Invalid—This cache block does not hold valid data.

Cache coherency is enforced by on-chip bus snooping logic. Since the 603e's data cache tags are single-ported, a simultaneous load or store and snoop access represent a resource contention. The snoop access is given first access to the tags. The load or store then occurs on the clock following the snoop.

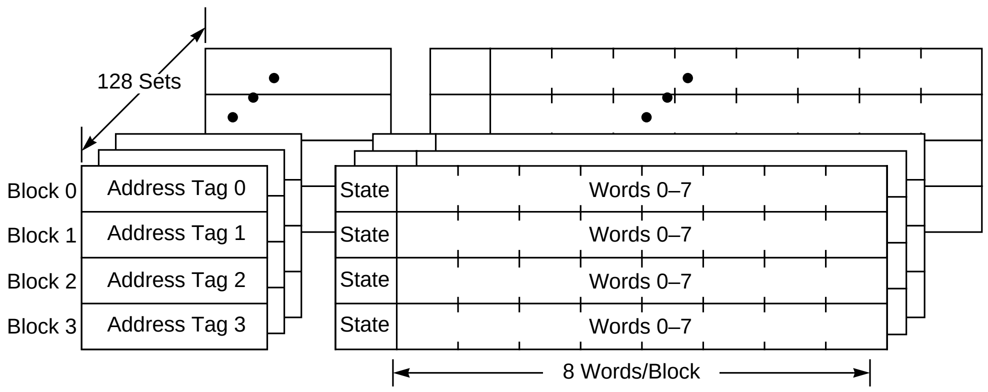


Figure 1-3. Data Cache Organization

1.3.4 Exception Model

This section describes the PowerPC exception model and the 603e implementation, specifically. PID7v-603e–specific information is noted where applicable.

1.3.4.1 PowerPC Exception Model

The PowerPC exception mechanism allows the processor to change to supervisor state as a result of external signals, errors, or unusual conditions arising in the execution of instructions, and differ from the arithmetic exceptions defined by the IEEE for floating-point operations. When exceptions occur, information about the state of the processor is saved to certain registers and the processor begins execution at an address (exception vector) predetermined for each exception. Processing of exceptions occurs in supervisor mode.

Although multiple exception conditions can map to a single exception vector, a more specific condition may be determined by examining a register associated with the exception—for example, the DSISR and the FPSCR. Additionally, some exception conditions can be explicitly enabled or disabled by software.

The PowerPC architecture requires that exceptions be handled in program order; therefore, although a particular implementation may recognize exception conditions out of order, they are presented strictly in order. When an instruction-caused exception is recognized, any unexecuted instructions that appear earlier in the instruction stream, including any that have not yet entered the execute stage, are required to complete before the exception is taken. Any exceptions caused by those instructions are handled first. Likewise, exceptions that are asynchronous and precise are recognized when they occur, but are not handled until the instruction currently in the completion stage successfully completes execution or generates an exception, and the completed store queue is emptied.

Unless a catastrophic condition causes a system reset or machine check exception, only one exception is handled at a time. If, for example, a single instruction encounters multiple exception conditions, those conditions are handled sequentially. After the exception handler handles an exception, the instruction execution continues until the next exception condition is encountered. However, in many cases there is no attempt to re-execute the instruction. This method of recognizing and handling exception conditions sequentially guarantees that exceptions are recoverable.

Exception handlers should save the information stored in SRR0 and SRR1 early to prevent the program state from being lost due to a system reset or machine check exception or to an instruction-caused exception in the exception handler, and before enabling external interrupts.

The PowerPC architecture supports four types of exceptions:

- Synchronous, precise—These are caused by instructions. All instruction-caused exceptions are handled precisely; that is, the machine state at the time the exception occurs is known and can be completely restored. This means that (excluding the trap and system call exceptions) the address of the faulting instruction is provided to the exception handler and that neither the faulting instruction nor subsequent instructions in the code stream will complete execution before the exception is taken. Once the exception is processed, execution resumes at the address of the faulting instruction (or at an alternate address provided by the exception handler). When an exception is taken due to a trap or system call instruction, execution resumes at an address provided by the handler.
- Synchronous, imprecise—The PowerPC architecture defines two imprecise floating-point exception modes, recoverable and nonrecoverable. Even though the 603e provides a means to enable the imprecise modes, it implements these modes

identically to the precise mode (that is, all enabled floating-point enabled exceptions are always precise on the 603e). (Note that the EC603e microprocessor does not support floating-point operations.)

- Asynchronous, maskable—The external, system management interrupt (SMI), and decrementer interrupts are maskable asynchronous exceptions. When these exceptions occur, their handling is postponed until the next instruction, and any exceptions associated with that instruction, completes execution. If there are no instructions in the execution units, the exception is taken immediately upon determination of the correct restart address (for loading SRR0).
- Asynchronous, nonmaskable—There are two nonmaskable asynchronous exceptions: system reset and the machine check exception. These exceptions may not be recoverable, or may provide a limited degree of recoverability. All exceptions report recoverability through the MSR[RI] bit.

1.3.4.2 Implementation-Specific Exception Model

As specified by the PowerPC architecture, all 603e exceptions can be described as either precise or imprecise and either synchronous or asynchronous. Asynchronous exceptions (some of which are maskable) are caused by events external to the processor's execution; synchronous exceptions, which are all handled precisely by the 603e, are caused by instructions. The 603e exception classes are shown in Figure 1-4.

Figure 1-4. Exception Classifications

Synchronous/Asynchronous	Precise/Imprecise	Exception Type
Asynchronous, nonmaskable	Imprecise	Machine check System reset
Asynchronous, maskable	Precise	External interrupt Decrementer System management interrupt
Synchronous	Precise	Instruction-caused exceptions

Although exceptions have other characteristics as well, such as whether they are maskable or nonmaskable, the distinctions shown in Figure 1-4 define categories of exceptions that the 603e handles uniquely. Note that Figure 1-4 includes no synchronous imprecise instructions. While the PowerPC architecture supports imprecise handling of floating-point exceptions, the 603e, with the exception of the EC603e microprocessor, implements floating-point exception modes as precise exceptions.

The 603e's exceptions, and conditions that cause them, are listed in Figure 1-5.

Figure 1-5. Exceptions and Conditions

Exception Type	Vector Offset (hex)	Causing Conditions
Reserved	00000	—

Figure 1-5. Exceptions and Conditions (Continued)

Exception Type	Vector Offset (hex)	Causing Conditions
System reset	00100	A system reset is caused by the assertion of either $\overline{\text{SRESET}}$ or $\overline{\text{HRESET}}$.
Machine check	00200	A machine check is caused by the assertion of the $\overline{\text{TEA}}$ signal during a data bus transaction, assertion of $\overline{\text{MCP}}$, or an address or data parity error.
DSI	00300	The cause of a DSI exception can be determined by the bit settings in the DSISR, listed as follows: 1 Set if the translation of an attempted access is not found in the primary hash table entry group (HTEG), or in the rehashed secondary HTEG, or in the range of a DBAT register; otherwise cleared. 4 Set if a memory access is not permitted by the page or DBAT protection mechanism; otherwise cleared. 5 Set by an **eciwx** or **ecowx** instruction if the access is to an address that is marked as write-through, or execution of a load/store instruction that accesses a direct-store segment. 6 Set for a store operation and cleared for a load operation. 11 Set if **eciwx** or **ecowx** is used and EAR[E] is cleared.
ISI	00400	An ISI exception is caused when an instruction fetch cannot be performed for any of the following reasons: • The effective (logical) address cannot be translated. That is, there is a page fault for this portion of the translation, so an ISI exception must be taken to load the PTE (and possibly the page) into memory. • The fetch access is to a direct-store segment (indicated by SRR1[3] set). • The fetch access violates memory protection (indicated by SRR1[4] set). If the key bits (Ks and Kp) in the segment register and the PP bits in the PTE are set to prohibit read access, instructions cannot be fetched from this location.
External interrupt	00500	An external interrupt is caused when MSR[EE] = 1 and the $\overline{\text{INT}}$ signal is asserted.
Alignment	00600	An alignment exception is caused when the 603e cannot perform a memory access for any of the reasons described below: • The operand of a floating-point load or store instruction is not word-aligned. • The operand of **lmw**, **stmw**, **lwarx**, and **stwcx.** instructions are not aligned. • The operand of a single-register load or store operation is not aligned, and the 603e is in little-endian mode (PID6-603e only). • The execution of a floating-point load or store instruction to a direct-store segment. • The operand of a load, store, load multiple, store multiple, load string, or store string instruction crosses a segment boundary into a direct-store segment, or crosses a protection boundary. • Execution of a misaligned **eciwx** or **ecowx** instruction (PID7v-603e only). • The instruction is **lmw**, **stmw**, **lswi**, **lswx**, **stswi**, **stswx** and the 603e is in little-endian mode. • The operand of **dcbz** is in memory that is write-through-required or caching-inhibited.

Figure 1-5. Exceptions and Conditions (Continued)

Exception Type	Vector Offset (hex)	Causing Conditions
Program	00700	A program exception is caused by one of the following exception conditions, which correspond to bit settings in SRR1 and arise during execution of an instruction: • Floating-point enabled exception—A floating-point enabled exception condition is generated when the following condition is met: (MSR[FE0] | MSR[FE1]) & FPSCR[FEX] is 1. (Not supported by the EC603e microprocessor.) FPSCR[FEX] is set by the execution of a floating-point instruction that causes an enabled exception or by the execution of one of the “move to FPSCR” instructions that results in both an exception condition bit and its corresponding enable bit being set in the FPSCR. (Not supported by the EC603e microprocessor.) • Illegal instruction—An illegal instruction program exception is generated when execution of an instruction is attempted with an illegal opcode or illegal combination of opcode and extended opcode fields (including PowerPC instructions not implemented in the 603e), or when execution of an optional instruction not provided in the 603e is attempted (these do not include those optional instructions that are treated as no-ops). • Privileged instruction—A privileged instruction type program exception is generated when the execution of a privileged instruction is attempted and the MSR register user privilege bit, MSR[PR], is set. In the 603e, this exception is generated for **mtspr** or **mfspr** with an invalid SPR field if SPR[0] = 1 and MSR[PR] = 1. This may not be true for all PowerPC processors. • Trap—A trap type program exception is generated when any of the conditions specified in a trap instruction is met.
Floating-point unavailable	00800	A floating-point unavailable exception is caused by an attempt to execute a floating-point instruction (including floating-point load, store, and move instructions) when the floating-point available bit is disabled (MSR[FP] = 0). Note that the EC603e microprocessor takes a floating-point unavailable exception when execution of a floating-point instruction is attempted.
Decrementer	00900	The decrementer exception occurs when the most significant bit of the decrementer (DEC) register transitions from 0 to 1. Must also be enabled with the MSR[EE] bit.
Reserved	00A00–00BFF	—
System call	00C00	A system call exception occurs when a System Call (**sc**) instruction is executed.
Trace	00D00	A trace exception is taken when MSR[SE] =1 or when the currently completing instruction is a branch and MSR[BE] =1.
Reserved	00E00	The 603e does not generate an exception to this vector. Other PowerPC processors may use this vector for floating-point assist exceptions.
Reserved	00E10–00FFF	—
Instruction translation miss	01000	An instruction translation miss exception is caused when an effective address for an instruction fetch cannot be translated by the ITLB.
Data load translation miss	01100	A data load translation miss exception is caused when an effective address for a data load operation cannot be translated by the DTLB.

Figure 1-5. Exceptions and Conditions (Continued)

Exception Type	Vector Offset (hex)	Causing Conditions
Data store translation miss	01200	A data store translation miss exception is caused when an effective address for a data store operation cannot be translated by the DTLB, or where a DTLB hit occurs, and the change bit in the PTE must be set due to a data store operation.
Instruction address breakpoint	01300	An instruction address breakpoint exception occurs when the address (bits 0–29) in the IABR matches the next instruction to complete in the completion unit, and the IABR enable bit (bit 30) is set.
System management interrupt	01400	A system management interrupt is caused when MSR[EE] = 1 and the $\overline{\text{SMI}}$ input signal is asserted.
Reserved	01500–02FFF	—

1.3.5 Memory Management

The following subsections describe the memory management features of the PowerPC architecture, and the 603e implementation, respectively.

1.3.5.1 PowerPC Memory Management

The primary functions of the MMU are to translate logical (effective) addresses to physical addresses for memory accesses, and to provide access protection on blocks and pages of memory.

There are two types of accesses generated by the 603e that require address translation—instruction accesses, and data accesses to memory generated by load and store instructions.

The PowerPC MMU and exception model support demand-paged virtual memory. Virtual memory management permits execution of programs larger than the size of physical memory; demand-paged implies that individual pages are loaded into physical memory from system memory only when they are first accessed by an executing program.

The hashed page table is a variable-sized data structure that defines the mapping between virtual page numbers and physical page numbers. The page table size is a power of 2, and its starting address is a multiple of its size.

The page table contains a number of page table entry groups (PTEGs). A PTEG contains eight page table entries (PTEs) of eight bytes each; therefore, each PTEG is 64 bytes long. PTEG addresses are entry points for table search operations.

Address translations are enabled by setting bits in the MSR—MSR[IR] enables instruction address translations and MSR[DR] enables data address translations.

1.3.5.2 Implementation-Specific Memory Management

The instruction and data memory management units in the 603e provide 4 Gbytes of logical address space accessible to supervisor and user programs with a 4-Kbyte page size and

256-Mbyte segment size. Block sizes range from 128 Kbyte to 256 Mbyte and are software selectable. In addition, the 603e uses an interim 52-bit virtual address and hashed page tables for generating 32-bit physical addresses. The MMUs in the 603e rely on the exception processing mechanism for the implementation of the paged virtual memory environment and for enforcing protection of designated memory areas.

Instruction and data TLBs provide address translation in parallel with the on-chip cache access, incurring no additional time penalty in the event of a TLB hit. A TLB is a cache of the most recently used page table entries. Software is responsible for maintaining the consistency of the TLB with memory. The 603e's TLBs are 64-entry, two-way set-associative caches that contain instruction and data address translations. The 603e provides hardware assist for software table search operations through the hashed page table on TLB misses. Supervisor software can invalidate TLB entries selectively.

The 603e also provides independent four-entry BAT arrays for instructions and data that maintain address translations for blocks of memory. These entries define blocks that can vary from 128 Kbytes to 256 Mbytes. The BAT arrays are maintained by system software.

As specified by the PowerPC architecture, the hashed page table is a variable-sized data structure that defines the mapping between virtual page numbers and physical page numbers. The page table size is a power of 2, and its starting address is a multiple of its size.

Also as specified by the PowerPC architecture, the page table contains a number of page table entry groups (PTEGs). A PTEG contains eight page table entries (PTEs) of eight bytes each; therefore, each PTEG is 64 bytes long. PTEG addresses are entry points for table search operations.

1.3.6 Instruction Timing

The 603e is a pipelined superscalar processor. A pipelined processor is one in which the processing of an instruction is reduced into discrete stages. Because the processing of an instruction is broken into a series of stages, an instruction does not require the entire resources of an execution unit. For example, after an instruction completes the decode stage, it can pass on to the next stage, while the subsequent instruction can advance into the decode stage. This improves the throughput of the instruction flow. For example, it may take three cycles for a floating-point instruction to complete, but if there are no stalls in the floating-point pipeline, a series of floating-point instructions can have a throughput of one instruction per cycle.

The instruction pipeline in the 603e has four major pipeline stages, described as follows:

- The fetch pipeline stage primarily involves retrieving instructions from the memory system and determining the location of the next instruction fetch. Additionally, the BPU decodes branches during the fetch stage and folds out branch instructions before the dispatch stage if possible.
- The dispatch pipeline stage is responsible for decoding the instructions supplied by the instruction fetch stage, and determining which of the instructions are eligible to be dispatched in the current cycle. In addition, the source operands of the instructions are read from the appropriate register file and dispatched with the instruction to the execute pipeline stage. At the end of the dispatch pipeline stage, the dispatched instructions and their operands are latched by the appropriate execution unit.
- During the execute pipeline stage each execution unit that has an executable instruction executes the selected instruction (perhaps over multiple cycles), writes the instruction's result into the appropriate rename register, and notifies the completion stage that the instruction has finished execution. In the case of an internal exception, the execution unit reports the exception to the completion/writeback pipeline stage and discontinues instruction execution until the exception is handled. The exception is not signaled until that instruction is the next to be completed. Execution of most floating-point instructions is pipelined within the FPU allowing up to three instructions to be executing in the FPU concurrently. The pipeline stages for the floating-point unit are multiply, add, and round-convert. Execution of most load/store instructions is also pipelined. The load/store unit has two pipeline stages. The first stage is for effective address calculation and MMU translation and the second stage is for accessing the data in the cache. (Note that the EC603e microprocessor does not support the floating-point unit.)
- The complete/writeback pipeline stage maintains the correct architectural machine state and transfers the contents of the rename registers to the GPRs and FPRs as instructions are retired. If the completion logic detects an instruction causing an exception, all following instructions are cancelled, their execution results in rename registers are discarded, and instructions are fetched from the correct instruction stream.

A superscalar processor is one that issues multiple independent instructions into multiple pipelines allowing instructions to execute in parallel. The 603e has five independent execution units, one each for integer instructions, floating-point instructions (floating-point instructions are trapped by the floating-point unavailable exception on the EC603e microprocessor), branch instructions, load/store instructions, and system register instructions. The IU and the FPU each have dedicated register files for maintaining operands (GPRs and FPRs, respectively), allowing integer calculations and floating-point calculations to occur simultaneously without interference. Integer division performance of the PID7v-603e has been improved, with the **divwu***x* and **divw***x* instructions executing in 20 clock cycles, instead of the 37 cycles required in the PID6-603e.

The 603e provides support for single-cycle store and it provides an adder/comparator in the system register unit that allows the dispatch and execution of multiple integer add and compare instructions on each cycle. Refer to Chapter 6, "Instruction Timing," for more information.

Because the PowerPC architecture can be applied to such a wide variety of implementations, instruction timing among various PowerPC processors varies accordingly.

1.3.7 System Interface

The system interface is specific for each PowerPC microprocessor implementation.

The 603e provides a versatile system interface that allows for a wide range of implementations. The interface includes a 32-bit address bus, a 32- or 64-bit data bus, and 56 control and information signals (see Figure 1-6). The system interface allows for address-only transactions as well as address and data transactions. The 603e control and information signals include the address arbitration, address start, address transfer, transfer attribute, address termination, data arbitration, data transfer, data termination, and processor state signals. Test and control signals provide diagnostics for selected internal circuits.

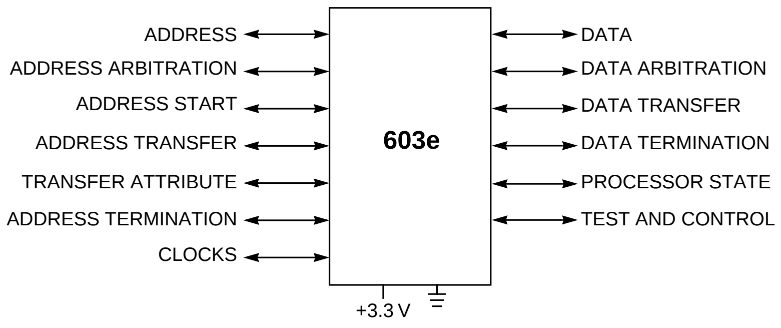


Figure 1-6. System Interface

The system interface supports bus pipelining, which allows the address tenure of one transaction to overlap the data tenure of another. The extent of the pipelining depends on external arbitration and control circuitry. Similarly, the 603e supports split-bus transactions for systems with multiple potential bus masters—one device can have mastership of the address bus while another has mastership of the data bus. Allowing multiple bus transactions to occur simultaneously increases the available bus bandwidth for other activity and as a result, improves performance.

The 603e supports multiple masters through a bus arbitration scheme that allows various devices to compete for the shared bus resource. The arbitration logic can implement priority protocols, such as fairness, and can park masters to avoid arbitration overhead. The MEI protocol ensures coherency among multiple devices and system memory. Also, the 603e's on-chip caches and TLBs and optional second-level caches can be controlled externally.

The 603e’s clocking structure allows the bus to operate at integer multiples of the processor cycle time.

The following sections describe the 603e bus support for memory operations. Note that some signals perform different functions depending upon the addressing protocol used.

1.3.7.1 Memory Accesses

The 603e’s data bus is configured at power-up to either a 32- or 64-bit width. When the 603e is configured with a 32-bit data bus, memory accesses allow transfer sizes of 8, 16, 24, or 32 bits in one bus clock cycle. Data transfers occur in either single-beat transactions, or two-beat or eight-beat burst transactions, with a single-beat transaction transferring as many as 32 bits. Single- or double-beat transactions are caused by noncached accesses that access memory directly (that is, reads and writes when caching is disabled, caching-inhibited accesses, and stores in write-through mode). Eight-beat burst transactions, which always transfer an entire cache line (32 bytes), are initiated when a line is read from or written to memory.

When the 603e is configured with a 64-bit data bus, memory accesses allow transfer sizes of 8, 16, 24, 32, 40, 48, 56, or 64 bits in one bus clock cycle. Data transfers occur in either single-beat transactions or four-beat burst transactions. Single-beat transactions are caused by noncached accesses that access memory directly (that is, reads and writes when caching is disabled, caching-inhibited accesses, and stores in write-through mode). Four-beat burst transactions, which always transfer an entire cache line (32 bytes), are initiated when a line is read from or written to memory.

1.3.7.2 Signals

The 603e signals are grouped as follows:

- Address arbitration signals—The 603e uses these signals to arbitrate for address bus mastership.
- Address transfer start signals—These signals indicate that a bus master has begun a transaction on the address bus.
- Address transfer signals—These signals, which consist of the address bus, address parity, and address parity error signals, are used to transfer the address and to ensure the integrity of the transfer.
- Transfer attribute signals—These signals provide information about the type of transfer, such as the transfer size and whether the transaction is bursted, write-through, or caching-inhibited.
- Address transfer termination signals—These signals are used to acknowledge the end of the address phase of the transaction. They also indicate whether a condition exists that requires the address phase to be repeated.

- Data arbitration signals—The 603e uses these signals to arbitrate for data bus mastership.
- Data transfer signals—These signals, which consist of the data bus, data parity, and data parity error signals, are used to transfer the data and to ensure the integrity of the transfer.
- Data transfer termination signals—Data termination signals are required after each data beat in a data transfer. In a single-beat transaction, the data termination signals also indicate the end of the tenure, while in burst accesses, the data termination signals apply to individual beats and indicate the end of the tenure only after the final data beat. They also indicate whether a condition exists that requires the data phase to be repeated.
- System status signals—These signals include the interrupt signal, checkstop signals, and both soft- and hard-reset signals. These signals are used to interrupt and, under various conditions, to reset the processor.
- Processor state signals—These signals indicate the state of the reservation coherency bit, enable the time base, provide machine quiesce control, and cause a machine halt on execution of a **tlbsync** instruction.
- IEEE 1149.1(JTAG)/COP interface signals—The IEEE 1149.1 test unit and the common on-chip processor (COP) unit are accessed through a shared set of input, output, and clocking signals. The IEEE 1149.1/COP interface provides a means for boundary scan testing and internal debugging of the 603e.
- Test interface signals—These signals are used for production testing.
- Clock signals—These signals determine the system clock frequency. These signals can also be used to synchronize multiprocessor systems.

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.

1.3.7.3 Signal Configuration

Figure 1-7 illustrates the 603e's logical pin configuration, showing how the signals are grouped.

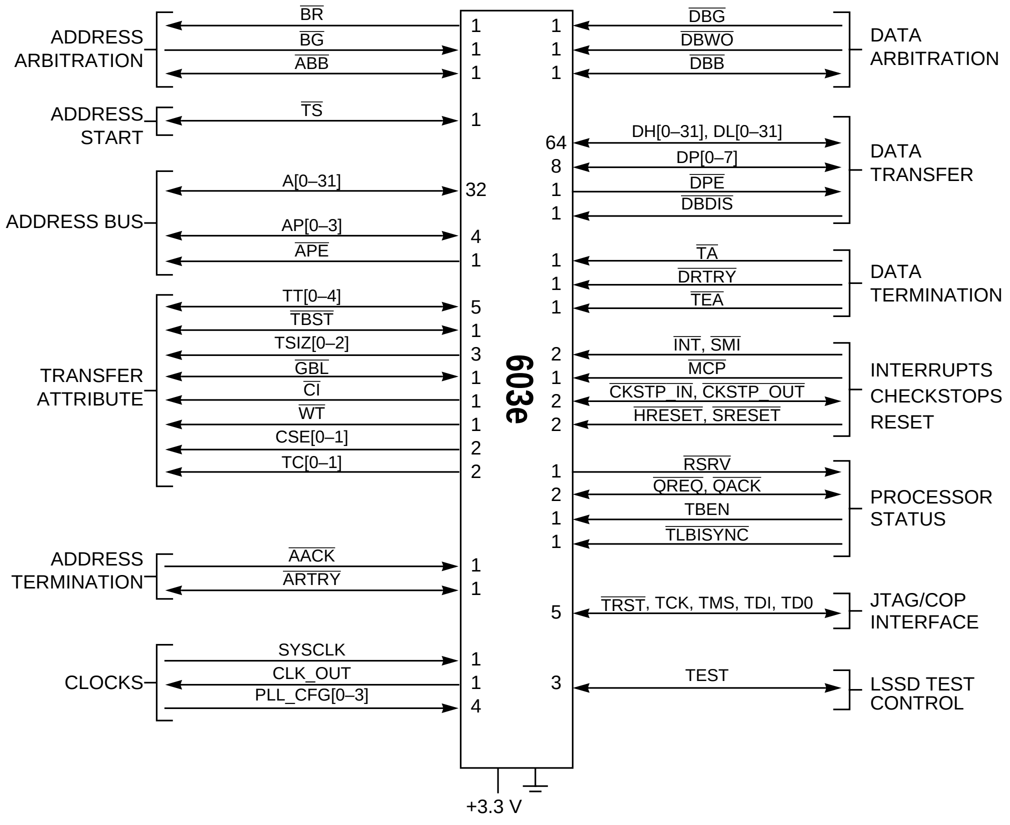


Figure 1-7. Signal Groups

Chapter 2
Programming Model

This chapter describes the PowerPC programming model with respect to the PowerPC 603e microprocessor. It consists of three major sections that describe the following:

- Registers implemented in the 603e
- Operand conventions
- The 603e instruction set

2.1 Register Set

This section describes the register organization in the 603e as defined by the three levels of the PowerPC architecture—the user instruction set architecture (UISA), the virtual environment architecture (VEA), and the operating environment architecture (OEA), as well as the 603e implementation-specific registers. Full descriptions of the basic register set defined by the PowerPC architecture are provided in Chapter 2, “PowerPC Register Set,” in *The Programming Environments Manual*.

The PowerPC architecture defines register-to-register operations for all computational instructions. Source data for these instructions is accessed from the on-chip registers or is provided as an immediate value embedded in the opcode. The three-register instruction format allows specification of a target register distinct from the two source registers, thus preserving the original data for use by other instructions and reducing the number of instructions required for certain operations. Data is transferred between memory and registers with explicit load and store instructions only.

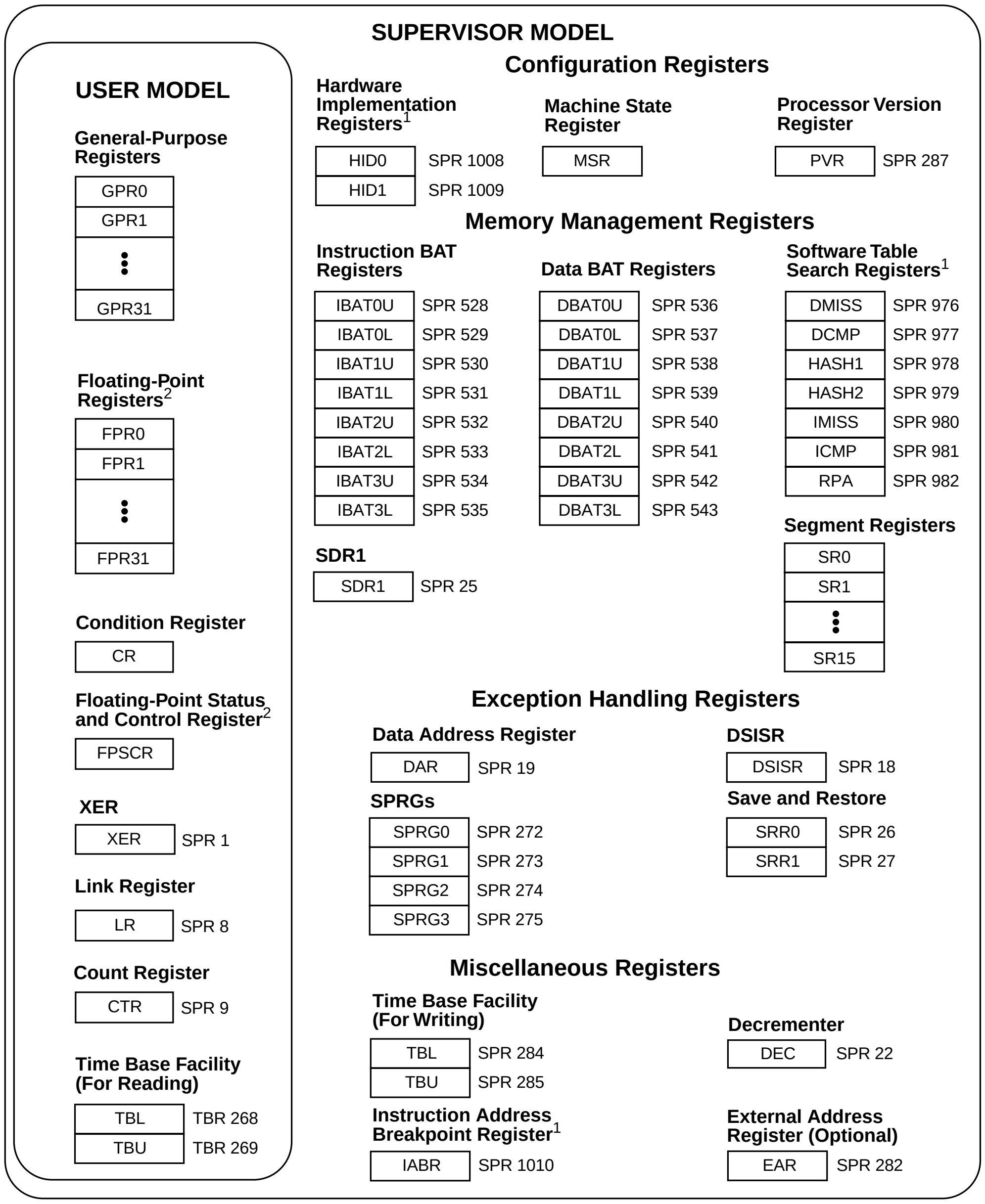
Note that there may be registers common to other PowerPC processors that are not implemented in the 603e. When the 603e detects special-purpose register (SPR) encodings other than those defined in this document, it either takes an exception or it treats the instruction as a no-op. (Note that exceptions are referred to as interrupts in the architecture specification.) Conversely, some SPRs in the 603e may not be implemented in other PowerPC processors, or may not be implemented in the same way in other PowerPC processors.

2.1.1 PowerPC Register Set

The PowerPC UISA registers, shown in Figure 2-1, can be accessed by either user- or supervisor-level instructions (the architecture specification refers to user- and supervisor-level as problem state and privileged state, respectively). The general-purpose registers (GPRs) and floating-point registers (FPRs) are accessed through instruction operands. (Note that the EC603e microprocessor does not support the floating-point register file; an attempt to access the floating-point register file will result in a floating-point unavailable exception.) Access to registers can be explicit (that is, through the use of specific instructions for that purpose such as the **mtspr** and **mfspr** instructions) or implicit as part of the execution (or side effect) of an instruction. Some registers are accessed both explicitly and implicitly.

The number to the right of the register name indicates the number that is used in the syntax of the instruction operands to access the register (for example, the number used to access the XER is SPR1).

For more information on the PowerPC register set, refer to Chapter 2, “PowerPC Register Set,” in *The Programming Environments Manual*.



Notes: [1]These registers are 603e–specific (PID6-603e and PID7v-603e) registers. They may not be supported by other PowerPC processors.
[2]Not supported on the EC603e microprocessor.

Figure 2-1. Programming Model—Registers

The 603e’s user-level registers are described as follows:

- **User-level registers (UISA)**—The user-level registers can be accessed by all software with either user or supervisor privileges. The user-level register set includes the following:
  - — General-purpose registers (GPRs). The general-purpose register file consists of thirty-two 32-bit GPRs designated as GPR0–GPR31. This register file serves as the data source or destination for all integer instructions and provides data for generating addresses.
  - — Floating-point registers (FPRs). The floating-point register file consists of thirty-two 64-bit FPRs designated as FPR0–FPR31, which serves as the data source or destination for all floating-point instructions. These registers can contain data objects of either single- or double-precision floating-point format. (The floating-point register file is not supported on the EC603e microprocessor; an attempt to access the floating-point register file will result in a floating-point unavailable exception.)
  - — Condition register (CR). The CR is a 32-bit register, divided into eight 4-bit fields, CR0–CR7, that reflects the results of certain arithmetic operations and provides a mechanism for testing and branching.
  - — Floating-point status and control register (FPSCR). The FPSCR is a user-control register that contains all floating-point exception signal bits, exception summary bits, exception enable bits, and rounding control bits needed for compliance with the IEEE 754 standard. (The FPU is not supported on the EC603e microprocessor; an attempt to access the floating-point register file will result in a floating-point unavailable exception.)

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

  - — XER register (XER). The XER is a 32-bit register that indicates overflow and carries for integer operations. It is set implicitly by many instructions.
  - — Link register (LR). The 32-bit link register provides the branch target address for the Branch Conditional to Link Register (**bclr***x*) instruction, and can optionally be used to hold the logical address (referred to as the effective address in the architecture specification) of the instruction that follows a branch and link instruction, typically used for linking to subroutines.
  - — Count register (CTR). The CTR is a 32-bit register for holding a loop count that can be decremented during execution of appropriately coded branch instructions. The CTR can also provide the branch target address for the Branch Conditional to Count Register (**bcctr***x*) instruction.

- **User-level registers (VEA)**—The PowerPC VEA introduces the time base facility (TB) for reading. The TB is a 64-bit register pair whose contents are incremented once every four bus clock cycles. The TB consists of two 32-bit registers—time base upper (TBU) and time base lower (TBL). Note that the time base registers are read-only when in user state.

The 603e's supervisor-level registers are described as follows:

- **Supervisor-level registers (OEA)**—The OEA defines the registers that are used typically by an operating system for such operations as memory management, configuration, and exception handling. The supervisor-level registers defined by the PowerPC architecture for 32-bit implementations are described as follows:
  - **Configuration registers**
    - Machine state register (MSR). The MSR defines the state of the processor. The MSR can be modified by the Move to Machine State Register (**mtmsr**), System Call (**sc**), and Return from Exception (**rfi**) instructions. It can be read by the Move from Machine State Register (**mfmsr**) instruction.

      Implementation Note—The 603e defines MSR[13] as the power management enable (POW) bit and MSR[14] as the temporary GPR remapping (TGPR) bit. These additional bits are described in Table 2-1.

Table 2-1. MSR[POW] and MSR[TGPR] Bits

Bit	Name	Description
13	POW	Power management enable (603e-specific) 0 Disables programmable power modes (normal operation mode). 1 Enables programmable power modes (nap, doze, or sleep mode). This bit controls the programmable power modes only, it has no effect on dynamic power management (DPM). MSR[POW] may be altered with an **mtmsr** instruction only. Also, when altering the POW bit, software may alter only this bit in the MSR and no others. The **mtmsr** instruction must be followed by a context-synchronizing instruction. See Chapter 9, "Power Management," for more information on power management.
14	TGPR	Temporary GPR remapping (603e-specific) 0 Normal operation 1 TGPR mode. GPR0–GPR3 are remapped to TGPR0–TGPR3 for use by TLB miss routines. The contents of GPR0–GPR3 remain unchanged while MSR[TGPR] = 1. Attempts to use GPR4–GPR31 with MSR[TGPR] = 1 yield undefined results. Overlays TGPR0–TGPR3 over GPR0–GPR3 for use by TLB miss routines. When this bit is set, all instruction accesses to GPR0–GPR3 are mapped to TGPR0–TGPR3, respectively. The contents of GPR0–GPR3 are unchanged as long as this bit remains set. Attempts to use GPR4–GPR31 when this bit is set yields undefined results.The TGPR bit is set when either an instruction TLB miss, data read miss, or data write miss exception is taken. The TGPR bit is cleared by an **rfi** instruction.

– Processor version register (PVR). This register is a read-only register that identifies the version (model) and revision level of the PowerPC processor.

  Implementation Note—The processor version number is 6 for the PID6-603e and 7 for the PID7v-603e. The processor revision level starts at 0x0100 and changes for each chip revision. The revision level is updated on all silicon revisions.

— **Memory management registers**

– Block-address translation (BAT) registers. The 603e includes eight block-address translation registers (BATs), consisting of four pairs of instruction BATs (IBAT0U–IBAT3U and IBAT0L–IBAT3L) and four pairs of data BATs (DBAT0U–DBAT3U and DBAT0L–DBAT3L). See Figure 2-1 for a list of the SPR numbers for the BAT registers.

– SDR1. The SDR1 register specifies the page table base address used in virtual-to-physical address translation. (Note that physical address is referred to as real address in the architecture specification.)

– Segment registers (SR). The PowerPC OEA defines sixteen 32-bit segment registers (SR0–SR15). Note that SRs are implemented on 32-bit implementations only. The fields in the segment register are interpreted differently depending on the value of bit 0.

— **Exception handling registers**

– Data address register (DAR). After a data access or an alignment exception, the DAR is set to the effective address generated by the faulting instruction.

– SPRG0–SPRG3. The SPRG0–SPRG3 registers are provided for operating system use.

– DSISR. The DSISR defines the cause of data access and alignment exceptions.

– Machine status save/restore register 0 (SRR0). The SRR0 is used to save machine status on exceptions and to restore machine status when an **rfi** instruction is executed.

– Machine status save/restore register 1 (SRR1). The SRR1 is used to save machine status on exceptions and to restore machine status when an **rfi** instruction is executed.

  Implementation Note—The 603e implements the Key bit (bit 12) in the SRR1 register in order to simplify the table search software. For more information refer to Chapter 5, "Memory Management."

— **Miscellaneous registers**

– The time base facility (TB) for writing. The TB is a 64-bit register pair that can be used to provide time of day or interval timing. It consists of two 32-bit registers—time base upper (TBU) and time base lower (TBL). The TB is incremented once every four clock cycles.

– Decrementer (DEC). The DEC register is a 32-bit decrementing counter that provides a mechanism for causing a decrementer exception after a programmable delay. The DEC is decremented once every four bus clock cycles.

– External access register (EAR). The EAR is a 32-bit register used in conjunction with the **eciwx** and **ecowx** instructions. While the PowerPC architecture specifies that the low-order six bits of the EAR (bits 26–31) are used to select a device, the 603e only implements the low-order 4 bits (bits 28–31). Note that the EAR register and the **eciwx** and **ecowx** instructions are optional in the PowerPC architecture and may not be supported in all PowerPC processors that implement the OEA.

2.1.2 Implementation-Specific Registers

The 603e includes several implementation-specific SPRs that are not defined by the PowerPC architecture. They are the DMISS, IMISS, DCMP, ICMP, HASH1, HASH2, RPA, HID0, HID1, and IABR registers. These registers can be accessed by supervisor-level instructions only. Any attempt to access these SPRs with user-level instructions results in a supervisor-level exception. The SPR numbers for these registers are shown in Figure 2-1.

The DMISS, IMISS, DCMP, ICMP, HASH1, HASH2, and RPA registers are used for software table search operations and should only be accessed when address translation is disabled (that is, MSR[IR] = 0 and MSR[DR] = 0). For a complete discussion of software table search operations, refer to Section 5.5.2, "Implementation-Specific Table Search Operation."

2.1.2.1 Hardware Implementation Registers (HID0 and HID1)

The HID0 and HID1 registers, shown in Figure 2-2 and Figure 2-3 respectively, define enable bits for various 603e-specific features.

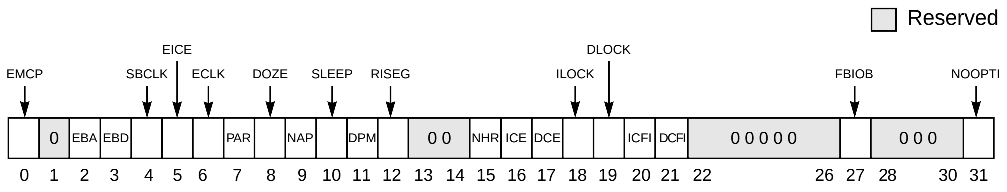


Figure 2-2. Hardware Implementation Register 0 (HID0)

Table 2-2 shows the bit definitions for HID0.

Table 2-2. HID0 Bit Settings

Bit(s)	Name	Description
0	EMCP	Enable machine check pin
1	—	Reserved
2	EBA	Enable bus address parity checking
3	EBD	Enable bus data parity checking
4	SBCLK	Select bus clock for test clock pin
5	EICE	Enable ICE outputs—pipeline tracking support
6	ECLK	Enable external test clock pin
7	PAR	Disable precharge of $\overline{\text{ARTRY}}$ and shared signals
8	DOZE	Doze mode—PLL, time base, and snooping active[1]
9	NAP	Nap mode—PLL and time base active[1]
10	SLEEP	Sleep mode—no external clock required[1]
11	DPM	Enable dynamic power management[1]
12	RISEG	Reserved for test
13–14	—	Reserved
15	NHR	Reserved
16	ICE	Instruction cache enable[2]
17	DCE	Data cache enable[2]
18	ILOCK	Instruction cache LOCK[2]
19	DLOCK	Data cache LOCK[2]
20	ICFI	Instruction cache flash invalidate[2]
21	DCFI	Data cache flash invalidate[2]
22–23	—	Reserved
24	IFEM	Instruction fetch enable M (PID7v-603e only)
25–26	—	Reserved
27	FBIOB	Force branch indirect on bus
28	ABE	Address broadcast enable[2] (PID7v-603e only)
29–30	—	Reserved
31	NOOPTI	No-op touch instructions

Notes:
1. See Chapter 9, “Power Management,” for more information.
2. See Chapter 3, “Instruction and Data Cache Operation,” for more information.

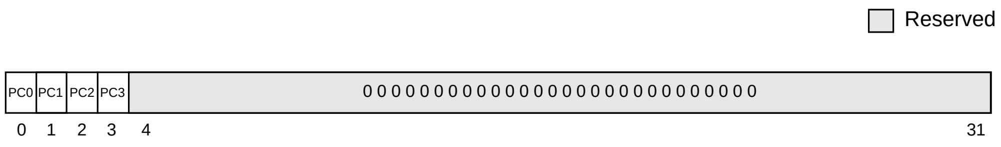


Figure 2-3. Hardware Implementation Register 1 (HID1)

Table 2-3 shows the bit definitions for HID1.

Table 2-3. HID1 Bit Settings

Bit(s)	Name	Description
0	PC0	PLL configuration bit 0 (read-only)
1	PC1	PLL configuration bit 1 (read-only)
2	PC2	PLL configuration bit 2 (read-only)
3	PC3	PLL configuration bit 3 (read-only)
4–31	—	Reserved

Note: The clock configuration bits reflect the state of the PLL_CFG[0–3] signals.

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

The DMISS and IMISS registers have the same format as shown in Figure 2-4. They are loaded automatically upon a data or instruction TLB miss. The DMISS and IMISS contain the effective page address of the access that caused the TLB miss exception. The contents are used by the 603e when calculating the values of HASH1 and HASH2, and by the **tlbld** and **tlbli** instructions when loading a new TLB entry. Note that the 603e always loads the DMISS register with a big-endian address, even when MSR[LE] is set. These registers are read and write to the software.

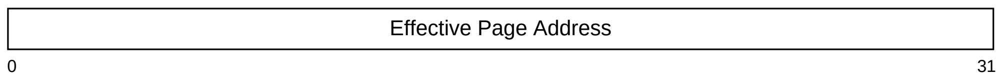


Figure 2-4. DMISS and IMISS Registers

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

The DCMP and ICMP registers are shown in Figure 2-5. These registers contain the first word in the required PTE. The contents are constructed automatically from the contents of the segment registers and the effective address (DMISS or IMISS) when a TLB miss exception occurs. Each PTE read from the tables during the table search process should be compared with this value to determine whether or not the PTE is a match. Upon execution of a **tlbld** or **tlbli** instruction the upper 25 bits of the DCMP or ICMP register and 11 bits of the effective address operand are loaded into the first word of the selected TLB entry. These registers are read and write to the software.

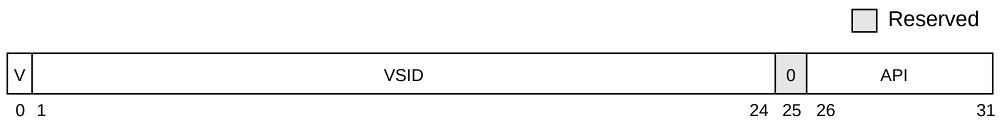


Figure 2-5. DCMP and ICMP Registers

Table 2-4 describes the bit settings for the DCMP and ICMP registers.

Table 2-4. DCMP and ICMP Bit Settings

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

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

The HASH1 and HASH2 registers contain the physical addresses of the primary and secondary PTEGs for the access that caused the TLB miss exception. For convenience, the 603e automatically constructs the full physical address by routing bits 0–6 of SDR1 into HASH1 and HASH2 and clearing the lower 6 bits. These registers are read-only and are constructed from the contents of the DMISS or IMISS register (the register choice is determined by which miss was last acknowledged). The format for the HASH1 and HASH2 registers is shown in Figure 2-6.

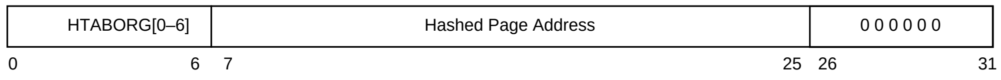


Figure 2-6. HASH1 and HASH2 Registers

Table 2-5 describes the bit settings of the HASH1 and HASH2 registers.

Table 2-5. HASH1 and HASH2 Bit Settings

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

2.1.2.5 Required Physical Address Register (RPA)

The RPA register is shown in Figure 2-7. During a page table search operation, the software must load the RPA with the second word of the correct PTE. When the **tlbld** or **tlbli** instruction is executed, the contents of the RPA register and the DMISS or IMISS register are merged and loaded into the selected TLB entry. The referenced (R) bit is ignored when the write occurs (no location exists in the TLB entry for this bit). The RPA register is read and write to the software.

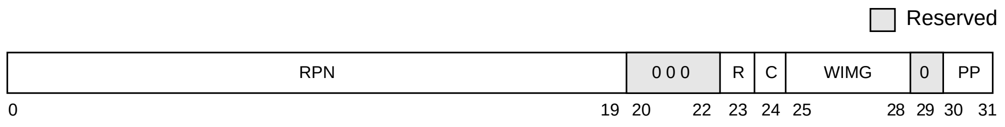


Figure 2-7. Required Physical Address Register (RPA)

Table 2-6 describes the bit settings of the RPA register.

Table 2-6. RPA Bit Settings

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

2.1.2.6 Instruction Address Breakpoint Register (IABR)

The IABR, shown in Figure 2-8, controls the instruction address breakpoint exception. IABR[CEA] holds an effective address to which each instruction is compared. The exception is enabled by setting bit 30 of IABR. The exception is taken when there is an instruction address breakpoint match on the next instruction to complete. The instruction tagged with the match will not be completed before the breakpoint exception is taken.

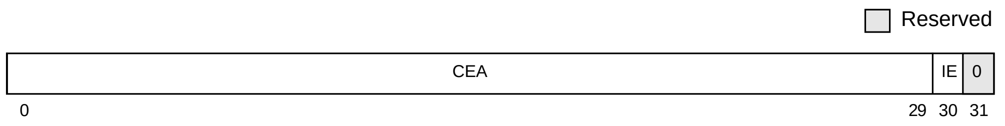


Figure 2-8. Instruction Address Breakpoint Register (IABR)

The bits in the IABR are defined as shown in Table 2-7.

Table 2-7. Instruction Address Breakpoint Register Bit Settings

Bit	Description
0–29	Word address to be compared
30	IABR enabled. Setting this bit indicates that the IABR exception is enabled.
31	Reserved

2.1.2.7 Run_N Counter Register (Run_N)

The 33-bit Run_N counter register is unique to the PID7v-603e. The Run_N counter is used by the COP to control the number of processor cycles that the processor runs before halting. The most-significant 32 bits form a 32-bit counter. The function of the least-significant bit remains unchanged.

2.2 Operand Conventions

This section describes the operand conventions as they are represented in two levels of the PowerPC architecture. It also provides detailed descriptions of conventions used for storing values in registers and memory, accessing the 603e’s registers, and representation of data in these registers.

2.2.1 Floating-Point Execution Models—UISA

Note that the floating-point execution models are not supported on the EC603e microprocessor.

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

The PowerPC UISA follows these guidelines:

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

For arithmetic instructions, conversions from double- to single-precision must be done explicitly by software, while conversions from single- to double-precision are done implicitly.

All PowerPC implementations provide the equivalent of the following execution models to ensure that identical results are obtained. The definition of the arithmetic instructions for infinities, denormalized numbers, and NaNs follow conventions described in the following sections.

Although the double-precision format specifies an 11-bit exponent, exponent arithmetic uses two additional bit positions to avoid potential transient overflow conditions. An extra bit is required when denormalized double-precision numbers are prenormalized. A second bit is required to permit computation of the adjusted exponent value in the following examples when the corresponding exception enable bit is 1:

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

2.2.2 Data Organization in Memory and Data Transfers

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

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

2.2.3 Alignment and Misaligned Accesses

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

Operands for single-register memory access instructions have the characteristics shown in Table 2-8. (Although not permitted as memory operands, quad words are shown because quad-word alignment is desirable for certain memory operands.)

Table 2-8. Memory Operands

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

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

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

Implementation Notes—The following describes how the 603e handles alignment and misaligned accesses:

- The 603e provides hardware support for some misaligned memory accesses. However, misaligned accesses will suffer a performance degradation compared to aligned accesses of the same type.
- The 603e does not provide hardware support for floating-point load/store operations that are not word-aligned. In such a case, the 603e will invoke an alignment exception and the exception handler must break up the misaligned access. For this reason, floating-point single- and double-word accesses should always be word-aligned. Note that a floating-point double-word access on a word-aligned boundary requires an extra cycle to complete. (Floating-point operations are not supported on the EC603e microprocessor.)

Any memory access that crosses an alignment boundary must be broken into multiple discrete accesses. This includes half-word, word, double-word, and string references. For the case of string accesses, the hardware makes no attempt to get aligned in an effort to reduce the number of discrete accesses. (Multiword accesses are architecturally required to be aligned.) The resulting performance degradation depends upon how well each individual access behaves with respect to the memory hierarchy. At a minimum, additional cache access cycles are required. More dramatically, for the case of access to a noncacheable page, each discrete access involves an individual bus operation which will reduce the effective bandwidth of the bus.

The frequent use of misaligned accesses is discouraged since they can compromise the overall performance of the processor.

2.2.4 Floating-Point Operand

The 603e provides hardware support for all single- and double-precision floating-point operations (not supported on the EC603e microprocessor) for most value representations and all rounding modes. The PowerPC architecture provides for hardware to implement a floating-point system as defined in ANSI/IEEE standard 754-1985, *IEEE Standard for Binary Floating Point Arithmetic*. For detailed information about the floating-point execution model refer to Chapter 3, “Operand Conventions,” in *The Programming Environments Manual*.

2.2.5 Effect of Operand Placement on Performance

The VEA states that the placement (location and alignment) of operands in memory affect the relative performance of memory accesses. The best performance is guaranteed if memory operands are aligned on natural boundaries. To obtain the best performance from the 603e, the programmer should assume the performance model described in Chapter 3, “Operand Conventions,” in *The Programming Environments Manual*.

2.3 Instruction Set Summary

This section describes instructions and addressing modes defined for the 603e. These instructions are divided into the following functional categories:

- Integer instructions—These include arithmetic and logical instructions. For more information, see Section 2.3.4.1, “Integer Instructions.”
- Floating-point instructions—These include floating-point arithmetic instructions, as well as instructions that affect the floating-point status and control register (FPSCR). For more information, see Section 2.3.4.2, “Floating-Point Instructions.” (Note that floating-point operations are not supported on the EC603e microprocessor)
- Load and store instructions—These include integer and floating-point load and store instructions. For more information, see Section 2.3.4.3, “Load and Store Instructions.”
- Flow control instructions—These include branching instructions, condition register logical instructions, and other instructions that affect the instruction flow. For more information, see Section 2.3.4.4, “Branch and Flow Control Instructions.”
- Trap instructions—These instructions are used to test for a specified set of conditions; see Section 2.3.4.5, “Trap Instructions,” for more information.
- Processor control instructions—These instructions are used for synchronizing memory accesses and managing caches, TLBs, and segment registers. For more information, see Sections 2.3.4.6, 2.3.5.1, and 2.3.6.2.
- Memory synchronization instructions—These instructions are used for memory synchronizing. See Sections 2.3.4.7 and Section 2.3.5.2 for more information.
- Memory control instructions—These instructions provide control of caches, TLBs, and segment registers. For more information, see Sections 2.3.5.3 and 2.3.6.3.
- System linkage instructions—For more information, see Section 2.3.6.1, “System Linkage Instructions.”
- External control instructions—These include instructions for use with special input/output devices. For more information, see Section 2.3.5.4, “External Control Instructions.”

Note that this grouping of instructions does not necessarily indicate the execution unit that processes a particular instruction or group of instructions. This information, which is useful in taking full advantage of the 603e’s superscalar parallel instruction execution, is provided in Chapter 8, “Instruction Set,” in *The Programming Environments Manual*.

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

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

The description of each instruction includes the mnemonic and a formatted list of operands. To simplify assembly language programming, a set of simplified mnemonics (extended mnemonics in the architecture specification) and symbols is provided for some of the frequently-used instructions; see Appendix F, “Simplified Mnemonics,” in *The Programming Environments Manual* for a complete list of simplified mnemonic examples.

2.3.1 Classes of Instructions

The 603e instructions belong to one of the following three classes:

- Defined
- Illegal
- Reserved

Note that while the definitions of these terms are consistent among the PowerPC processors, the assignment of these classifications is not. For example, an instruction that is specific to 64-bit implementations is considered defined for 64-bit implementations but illegal for 32-bit implementations such as the 603e.

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

In future versions of the PowerPC architecture, instruction codings that are now illegal may become assigned to instructions in the architecture, or may be reserved by being assigned to processor-specific instructions.

2.3.1.1 Definition of Boundedly Undefined

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

2.3.1.2 Defined Instruction Class

Defined instructions are guaranteed to be supported in all PowerPC implementations, except as stated in the instruction descriptions in Chapter 8, “Instruction Set,” in *The Programming Environments Manual*. The 603e provides hardware support for all instructions defined for 32-bit implementations (the EC603e microprocessor supports all 32-bit instructions with the exception of those defined for floating-point operations).

A PowerPC processor invokes the illegal instruction error handler (part of the program exception) when the unimplemented PowerPC instructions are encountered so they may be emulated in software, as required.

A defined instruction can have invalid forms, as described in the following subsection.

2.3.1.3 Illegal Instruction Class

Illegal instructions can be grouped into the following categories:

- Instructions that are not implemented in the PowerPC architecture. These opcodes are available for future extensions of the PowerPC architecture; that is, future versions of the PowerPC architecture may define any of these instructions to perform new functions.

  The following primary opcodes are defined as illegal but may be used in future extensions to the architecture:

  1, 4, 5, 6, 9, 22, 56, 57, 60, 61
- Instructions that are implemented in the PowerPC architecture but are not implemented in a specific PowerPC implementation. For example, instructions that can be executed on 64-bit PowerPC processors are considered illegal by 32-bit processors.

  The following primary opcodes are defined for 64-bit implementations only and are illegal on the 603e:

  2, 30, 58, 62
- All unused extended opcodes are illegal. The unused extended opcodes can be determined from information in Section A.2, “Instructions Sorted by Opcode,” and Section 2.3.1.4, “Reserved Instruction Class.” Notice that extended opcodes for instructions that are defined only for 64-bit implementations are illegal in 32-bit implementations, and vice versa.

  The following primary opcodes have unused extended opcodes.

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

An attempt to execute an illegal instruction invokes the illegal instruction error handler (a program exception) but has no other effect. See Section 4.5.7, “Program Exception (0x00700),” for additional information about illegal and invalid instruction exceptions.

With the exception of the instruction consisting entirely of binary zeros, the illegal instructions are available for further additions to the PowerPC architecture.

2.3.1.4 Reserved Instruction Class

Reserved instructions are allocated to specific implementation-dependent purposes not defined by the PowerPC architecture. An attempt to execute an unimplemented reserved instruction invokes the illegal instruction error handler (a program exception). See Section 4.5.7, “Program Exception (0x00700),” for additional information about illegal and invalid instruction exceptions.

The following types of instructions are included in this class:

- Implementation-specific instructions (for example, Load Data TLB Entry (**tlbld**) and Load Instruction TLB Entry (**tlbli**) instructions)
- Optional instructions defined by the PowerPC architecture but not implemented by the 603e (for example, Floating Square Root (**fsqrt**) and Floating Square Root Single (**fsqrts**) instructions)

2.3.2 Addressing Modes

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

2.3.2.1 Memory Addressing

A program references memory using the effective (logical) address computed by the processor when it executes a memory access or branch instruction or when it fetches the next sequential instruction.

2.3.2.2 Memory Operands

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

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

Environments Manual for more information about big-endian and little-endian byte ordering.

The operand of a single-register memory access instruction has a natural alignment boundary equal to the operand length. In other words, the “natural” address of an operand is an integral multiple of the operand length. A memory operand is said to be aligned if it is aligned at its natural boundary; otherwise it is misaligned. For a detailed discussion about memory operands, see Chapter 3, “Operand Conventions,” in *The Programming Environments Manual*.

2.3.2.3 Effective Address Calculation

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

Effective address computations for both data and instruction accesses use 32-bit unsigned binary arithmetic. A carry from bit 0 is ignored.

Load and store operations have three categories of effective address generation:

- Register indirect with immediate index mode
- Register indirect with index mode
- Register indirect mode

Refer to Section 2.3.4.3.2, “Integer Load and Store Address Generation,” for further discussion of effective address generation for load and store operations.

Branch instructions have three categories of effective address generation:

- Immediate
- Link register indirect
- Count register indirect

Refer to Section 2.3.4.4.1, “Branch Instruction Address Calculation,” for further discussion of branch instruction effective address generation.

2.3.2.4 Synchronization

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

2.3.2.4.1 Context Synchronization

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

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

2.3.2.4.2 Execution Synchronization

An instruction is execution synchronizing if all previously initiated instructions appear to have completed before the instruction is initiated or, in the case of the Synchronize (**sync**) and Instruction Synchronize (**isync**) instructions, before the instruction completes. For example, the Move to Machine State Register (**mtmsr**) instruction is execution synchronizing. It ensures that all preceding instructions have completed execution and will not cause an exception before the instruction executes, but does not ensure subsequent instructions execute in the newly established environment. For example, if the **mtmsr** sets the MSR[PR] bit, unless an **isync** immediately follows the **mtmsr** instruction, a privileged instruction could be executed or privileged access could be performed without causing an exception even though the MSR[PR] bit indicates user mode.

2.3.2.4.3 Instruction-Related Exceptions

There are two kinds of exceptions in the 603e—those caused directly by the execution of an instruction and those caused by an asynchronous event. Either may cause components of the system software to be invoked.

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

- An attempt to execute an illegal instruction causes the illegal instruction (program exception) handler to be invoked. An attempt by a user-level program to execute the supervisor-level instructions listed below causes the privileged instruction (program exception) handler to be invoked. The 603e provides the following supervisor-level instructions: **dcbi**, **mfmsr**, **mfspr**, **mfsr**, **mfsrin**, **mtmsr**, **mtspr**, **mtsr**, **mtsrin**, **rfi**, **tlbie**, **tlbsync, tlbld**, and **tlbli**. Note that the privilege level of the **mfspr** and **mtspr** instructions depends on the SPR encoding.
- An attempt to access memory that is not available (page fault) causes the ISI exception handler to be invoked.
- An attempt to access memory with an effective address alignment that is invalid for the instruction causes the alignment exception handler to be invoked.

- The execution of an **sc** instruction invokes the system call exception handler that permits a program to request the system to perform a service.
- The execution of a trap instruction invokes the program exception trap handler.
- The execution of a floating-point instruction when floating-point instructions are disabled or unavailable invokes the floating-point unavailable exception handler.
- The execution of an instruction that causes a floating-point exception while exceptions are enabled in the MSR invokes the program exception handler.

Exceptions caused by asynchronous events are described in Chapter 4, “Exceptions.”

2.3.3 Instruction Set Overview

This section provides a brief overview of the PowerPC instructions implemented in the 603e and highlights any special information with respect to how the 603e implements a particular instruction. Note that the categories used in this section correspond to those used in Chapter 4, “Addressing Modes and Instruction Set Summary,” in *The Programming Environments Manual*. These categorizations are somewhat arbitrary and are provided for the convenience of the programmer and do not necessarily reflect the PowerPC architecture specification.

Note that some of the instructions have the following optional features:

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

2.3.4 PowerPC UISA Instructions

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

2.3.4.1 Integer Instructions

This section describes the integer instructions. These consist of the following:

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

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

2.3.4.1.1 Integer Arithmetic Instructions

Table 2-9 lists the integer arithmetic instructions for the 603e.

Table 2-9. Integer Arithmetic Instructions

Name	Mnemonic	Operand Syntax
Add Immediate	**addi**	**r**D,**r**A,SIMM
Add Immediate Shifted	**addis**	**r**D,**r**A,SIMM
Add	**add (add. addo addo.)**	**r**D,**r**A,**r**B
Subtract From	**subf (subf. subfo subfo.)**	**r**D,**r**A,**r**B
Add Immediate Carrying	**addic**	**r**D,**r**A,SIMM
Add Immediate Carrying and Record	**addic.**	**r**D,**r**A,SIMM
Subtract from Immediate Carrying	**subfic**	**r**D,**r**A,SIMM
Add Carrying	**addc (addc. addco addco.)**	**r**D,**r**A,**r**B
Subtract from Carrying	**subfc (subfc. subfco subfco.)**	**r**D,**r**A,**r**B
Add Extended	**adde (adde. addeo addeo.)**	**r**D,**r**A,**r**B
Subtract from Extended	**subfe (subfe. subfeo subfeo.)**	**r**D,**r**A,**r**B
Add to Minus One Extended	**addme (addme. addmeo addmeo.)**	**r**D,**r**A
Subtract from Minus One Extended	**subfme (subfme. subfmeo subfmeo.)**	**r**D,**r**A
Add to Zero Extended	**addze (addze. addzeo addzeo.)**	**r**D,**r**A
Subtract from Zero Extended	**subfze (subfze. subfzeo subfzeo.)**	**r**D,**r**A
Negate	**neg (neg. nego nego.)**	**r**D,**r**A
Multiply Low Immediate	**mulli**	**r**D,**r**A,SIMM
Multiply Low	**mullw (mullw. mullwo mullwo.)**	**r**D,**r**A,**r**B
Multiply High Word	**mulhw (mulhw.)**	**r**D,**r**A,**r**B
Multiply High Word Unsigned	**mulhwu (mulhwu.)**	**r**D,**r**A,**r**B
Divide Word	**divw (divw. divwo divwo.)**	**r**D,**r**A,**r**B
Divide Word Unsigned	**divwu (divwu. divwuo divwuo.)**	**r**D,**r**A,**r**B

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

2.3.4.1.2 Integer Compare Instructions

The integer compare instructions algebraically or logically compare the contents of **r**A with either the UIMM operand, the SIMM operand, or the contents of **r**B. The comparison is signed for the **cmpi** and **cmp** instructions, and unsigned for the **cmpli** and **cmpl** instructions. Table 2-10 lists the integer compare instructions.

Table 2-10. Integer Compare Instructions

Name	Mnemonic	Operand Syntax
Compare Immediate	**cmpi**	**crf**D,L,**r**A,SIMM
Compare	**cmp**	**crf**D,L,**r**A,**r**B
Compare Logical Immediate	**cmpli**	**crf**D,L,**r**A,UIMM
Compare Logical	**cmpl**	**crf**D,L,**r**A,**r**B

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

For more information refer to Appendix F, “Simplified Mnemonics,” in *The Programming Environments Manual*.

2.3.4.1.3 Integer Logical Instructions

The logical instructions shown in Table 2-11 perform bit-parallel operations. Logical instructions with the CR update enabled and instructions **andi.** and **andis.** set CR field CR0 to characterize the result of the logical operation. These fields are set as if the sign-extended low-order 32 bits of the result were algebraically compared to zero. Logical instructions without CR update and the remaining logical instructions do not modify the CR. Logical instructions do not affect the XER[SO], XER[OV], and XER[CA] bits.

For simplified mnemonics examples for the integer logical operations see Appendix F, “Simplified Mnemonics,” in *The Programming Environments Manual*.

Table 2-11. Integer Logical Instructions

Name	Mnemonic	Operand Syntax
AND Immediate	**andi.**	**r**A,**r**S,UIMM
AND Immediate Shifted	**andis.**	**r**A,**r**S,UIMM
OR Immediate	**ori**	**r**A,**r**S,UIMM
OR Immediate Shifted	**oris**	**r**A,**r**S,UIMM
XOR Immediate	**xori**	**r**A,**r**S,UIMM
XOR Immediate Shifted	**xoris**	**r**A,**r**S,UIMM
AND	**and (and.)**	**r**A,**r**S,**r**B
OR	**or (or.)**	**r**A,**r**S,**r**B
XOR	**xor (xor.)**	**r**A,**r**S,**r**B
NAND	**nand (nand.)**	**r**A,**r**S,**r**B
NOR	**nor (nor.)**	**r**A,**r**S,**r**B

Table 2-11. Integer Logical Instructions (Continued)

Name	Mnemonic	Operand Syntax
Equivalent	**eqv (eqv.)**	**r**A,**r**S,**r**B
AND with Complement	**andc (andc.)**	**r**A,**r**S,**r**B
OR with Complement	**orc (orc.)**	**r**A,**r**S,**r**B
Extend Sign Byte	**extsb (extsb.)**	**r**A,**r**S
Extend Sign Half Word	**extsh (extsh.)**	**r**A,**r**S
Count Leading Zeros Word	**cntlzw (cntlzw.)**	**r**A,**r**S

2.3.4.1.4 Integer Rotate and Shift Instructions

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

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

The integer rotate instructions are listed in Table 2-12.

Table 2-12. Integer Rotate Instructions

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

The integer shift instructions perform left and right shifts. Immediate-form logical (unsigned) shift operations are obtained by specifying masks and shift values for certain rotate instructions. Simplified mnemonics are provided to make coding of such shifts simpler and easier to understand.

Multiple-precision shifts can be programmed as shown in Appendix C, "Multiple-Precision Shifts," in *The Programming Environments Manual*.

The integer shift instructions are listed in Table 2-13.

Table 2-13. Integer Shift Instructions

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

2.3.4.2 Floating-Point Instructions

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

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

The EC603e microprocessor provides hardware support for all 32-bit PowerPC instructions with the exception of floating-point instructions, which, when implemented on the EC603e microprocessor, take a floating-point unavailable exception.

See Section 2.3.4.3, “Load and Store Instructions,” for information about floating-point loads and stores.

The PowerPC architecture supports a floating-point system as defined in the IEEE 754 standard, but requires software support to conform with that standard. All floating-point operations conform to the IEEE 754 standard, except if software sets the non-IEEE mode bit (NI) in the FPSCR; the 603e is in the nondenormalized mode when the NI bit is set in the FPSCR. If a denormalized result is produced, a default result of zero is generated. The generated zero has the same sign as the denormalized number. The 603e performs single- and double-precision floating-point operations compliant with the IEEE-754 floating-point standard.

Implementation Note—Single-precision denormalized results require two additional processor clock cycles to round. When loading or storing a single-precision denormalized number, the load/store unit may take up to 24 processor clock cycles to convert between the internal double-precision format and the external single-precision format.

2.3.4.2.1 Floating-Point Arithmetic Instructions

The floating-point arithmetic instructions are listed in Table 2-14. (Floating-point instructions are not supported on the EC603e microprocessor.)

Table 2-14. Floating-Point Arithmetic Instructions

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

2.3.4.2.2 Floating-Point Multiply-Add Instructions

These instructions combine multiply and add operations without an intermediate rounding operation. The fractional part of the intermediate product is 106 bits wide, and all 106 bits take part in the add/subtract portion of the instruction.

The floating-point multiply-add instructions are listed in Table 2-15. (Floating-point instructions are not supported on the EC603e microprocessor.)

Table 2-15. Floating-Point Multiply-Add Instructions

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

Implementation Note—Single-precision multiply-type instructions operate faster than their double-precision equivalents. See Chapter 6, “Instruction Timing,” for more information.

2.3.4.2.3 Floating-Point Rounding and Conversion Instructions

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

The PowerPC architecture defines bits 0–31 of floating-point register **fr**D as undefined when executing the Floating Convert to Integer Word (**fctiw**) and Floating Convert to Integer Word with Round toward Zero (**fctiwz**) instructions.

Examples of uses of these instructions to perform various conversions can be found in Appendix D, “Floating-Point Models,” in *The Programming Environments Manual*. The floating-point rounding instructions are shown in Table 2-16. (Floating-point instructions are not supported on the EC603e microprocessor.)

Table 2-16. Floating-Point Rounding and Conversion Instructions

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

2.3.4.2.4 Floating-Point Compare Instructions

Floating-point compare instructions compare the contents of two floating-point registers. The comparison ignores the sign of zero (that is +0 = –0). The floating-point compare instructions are listed in Table 2-17. (Floating-point instructions are not supported on the EC603e microprocessor.)

Table 2-17. Floating-Point Compare Instructions

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

2.3.4.2.5 Floating-Point Status and Control Register Instructions

Every FPSCR instruction appears to synchronize the effects of all floating-point instructions executed by a given processor. Executing an FPSCR instruction ensures that all floating-point instructions previously initiated by the given processor appear to have completed before the FPSCR instruction is initiated and that no subsequent floating-point instructions appear to be initiated by the given processor until the FPSCR instruction has completed. The FPSCR instructions are listed in Table 2-18. (Floating-point instructions are not supported on the EC603e microprocessor.)

Table 2-18. Floating-Point Status and Control Register Instructions

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

Implementation Note—The architecture notes that, in some implementations, the Move to FPSCR Fields (**mtfsf***x*) instruction may perform more slowly when only a portion of the fields are updated as opposed to all of the fields. This is not the case in the 603e.

2.3.4.2.6 Floating-Point Move Instructions

Floating-point move instructions copy data from one floating-point register to another. The floating-point move instructions do not modify the FPSCR. The CR update option in these instructions controls the placing of result status into CR1. Floating-point move instructions are listed in Table 2-18. (Floating-point instructions are not supported on the EC603e microprocessor.)

Table 2-19. Floating-Point Move Instructions

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

2.3.4.3 Load and Store Instructions

Load and store instructions are issued and translated in program order; however, the accesses can occur out of order. Synchronizing instructions are provided to enforce strict ordering. This section describes the load and store instructions of the 603e, which consist of the following:

- Integer load instructions
- Integer store instructions
- Integer load and store with byte-reverse instructions
- Integer load and store multiple instructions

- Integer load and store string instructions
- Floating-point load instructions
- Floating-point store instructions

2.3.4.3.1 Self-Modifying Code

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

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

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

Special care must be taken to avoid coherency paradoxes in systems that implement unified secondary caches, and designers should carefully follow the guidelines for maintaining cache coherency that are provided in the VEA, and discussed in Chapter 5, “Cache Model and Memory Coherency,” in *The Programming Environments Manual*. Because the 603e does not broadcast the M bit for instruction fetches, external caches are subject to coherency paradoxes.

2.3.4.3.2 Integer Load and Store Address Generation

Integer load and store operations generate effective addresses using register indirect with immediate index mode, register indirect with index mode, or register indirect mode. See Section 2.3.2.3, “Effective Address Calculation,” for information about calculating effective addresses. Note that the 603e is optimized for load and store operations that are aligned on natural boundaries, and operations that are not naturally aligned may suffer performance degradation. Refer to Section 4.5.6.1, “Integer Alignment Exceptions,” for additional information about load and store address alignment exceptions.

2.3.4.3.3 Register Indirect Integer Load Instructions

For integer load instructions, the byte, half word, word, or double word addressed by the EA is loaded into **r**D. Many integer load instructions have an update form, in which **r**A is updated with the generated effective address. For these forms, the EA is placed into **r**A and the memory element (byte, half word, word, or double word) addressed by EA is loaded into **r**D.

Implementation Note—In some implementations of the PowerPC architecture, the load half word algebraic instructions (**lha** and **lhax**) and the load with update (**lbzu**, **lbzux**, **lhzu**, **lhzux**, **lhau**, **lhaux**, **lwu**, and **lwux**) instructions may execute with greater latency than other types of load instructions. In the 603e, these instructions operate with the same latency as other load instructions.

Table 2-20 lists the integer load instructions.

Table 2-20. Integer Load Instructions

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

2.3.4.3.4 Integer Store Instructions

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

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

The 603e defines store with update instructions with **r**A = 0 and integer store instructions with the CR update option enabled (Rc field, bit 31, in the instruction encoding = 1) to be invalid forms. Table 2-21 provides a list of the integer store instructions for the 603e.

Table 2-21. Integer Store Instructions

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

2.3.4.3.5 Integer Load and Store with Byte-Reverse Instructions

Table 2-22 describes integer load and store with byte-reverse instructions. When used in a PowerPC system operating with the default big-endian byte order, these instructions have the effect of loading and storing data in little-endian order. Likewise, when used in a PowerPC system operating with little-endian byte order, these instructions have the effect of loading and storing data in big-endian order. For more information about big-endian and little-endian byte ordering, see “Byte Ordering” in Chapter 3, “Operand Conventions,” in *The Programming Environments Manual*.

Implementation Note—In some PowerPC implementations, load byte-reverse instructions (**lhbrx** and **lwbrx**) may have greater latency than other load instructions; however, these instructions operate with the same latency as other load instructions in the 603e.

Table 2-22. Integer Load and Store with Byte-Reverse Instructions

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

2.3.4.3.6 Integer Load and Store Multiple Instructions

The integer load/store multiple instructions are used to move blocks of data to and from the GPRs. In some implementations, these instructions are likely to have greater latency and take longer to execute, perhaps much longer, than a sequence of individual load or store instructions that produce the same results.

Implementation Notes—The following describes the 603e implementation of the load/store multiple instruction:

- The load multiple and store multiple instructions may have operands that require memory accesses crossing a 4-Kbyte page boundary. As a result, these instructions may be interrupted by a DSI exception associated with the address translation of the second page. In this case, the 603e performs some or all of the memory references from the first page, and none of the memory references from the second page before taking the exception. On return from the DSI exception, the load or store multiple instruction will re-execute from the beginning. For additional information, refer to "DSI Exception (0x00300)" in Chapter 6, "Exceptions," in *The Programming Environments Manual*.
- The PowerPC architecture defines the load multiple word (**lmw**) instruction with **r**A in the range of registers to be loaded as an invalid form. It defines the load multiple and store multiple instructions with misaligned operands (that is, the EA is not a multiple of 4) to cause an alignment exception. The 603e defines the load multiple word (**lmw**) instruction with **r**A in the range of registers to be loaded as an invalid form.
- The PowerPC architecture describes some preferred instruction forms for the integer load and store multiple instructions that may perform better than other forms in some implementations. None of these preferred forms have an effect on instruction performance in the 603e.

When the 603e is operating with little-endian byte order, execution of a load or store multiple instruction causes the system alignment error handler to be invoked; see "Byte Ordering" in Chapter 3, "Operand Conventions," in *The Programming Environments Manual* for more information. Table 2-23 lists the integer load and store multiple instructions for the 603e.

Table 2-23. Integer Load and Store Multiple Instructions

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

2.3.4.3.7 Integer Load and Store String Instructions

The integer load and store string instructions allow movement of data from memory to registers or from registers to memory without concern for alignment. These instructions can be used for a short move between arbitrary memory locations or to initiate a long move between misaligned memory fields.

When the 603e is operating with little-endian byte order, execution of a load or store string instruction causes the system alignment error handler to be invoked; see “Byte Ordering” in Chapter 3, “Operand Conventions,” in *The Programming Environments Manual* for more information.

Table 2-24 lists the integer load and store string instructions.

Table 2-24. Integer Load and Store String Instructions

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

Load string and store string instructions may involve operands that are not word-aligned. As described in “Alignment Exception (0x00600)” in Chapter 6, “Exceptions,” in *The Programming Environments Manual,* a misaligned string operation suffers a performance penalty compared to a word-aligned operation of the same type.

When a string operation crosses a 4-Kbyte boundary, the instruction may be interrupted by a DSI exception associated with the address translation of the second page. In this case, the 603e performs some or all memory references from the first page and none from the second before taking the exception. On return from the DSI exception, the load or store string instruction will re-execute from the beginning. For more information, refer to “DSI Exception (0x00300)” in Chapter 6, “Exceptions,” in *The Programming Environments Manual*.

Implementation Note—If **r**A is in the range of registers to be loaded for a Load String Word Immediate (**lswi**) instruction or if either **r**A or **r**B is in the range of registers to be loaded for a Load String Word Indexed (**lswx**) instruction, the PowerPC architecture defines the instruction to be of an invalid form. In addition, the **lswx** and **stswx** instructions that specify a string length of zero are defined to be invalid by the PowerPC architecture. However, neither of these cases holds true for the 603e which treats these cases as valid forms.

2.3.4.3.8 Floating-Point Load and Store Address Generation

Floating-point load and store operations generate effective addresses using the register indirect with immediate index addressing mode and register indirect with index addressing mode, the details of which are described below. Floating-point loads and stores are not supported for direct-store accesses. The use of the floating-point load and store operations for direct-store accesses will result in a DSI exception. (Note that floating-point instructions are not supported on the EC603e microprocessor.)

2.3.4.3.9 Floating-Point Load Instructions

There are two forms of the floating-point load instruction—single-precision and double-precision operand formats. Because the FPRs support only the floating-point double-precision format, single-precision floating-point load instructions convert single-precision data to double-precision format before loading the operands into the target FPR. This conversion is described fully in “Floating-Point Load Instructions” in Appendix D, “Floating-Point Models,” in *The Programming Environments Manual*.

Implementation Note—The PowerPC architecture defines load with update instructions with **r**A = 0 as an invalid form; however, the 603e treats this case as a valid form.

On the EC603e microprocessor, floating-point instructions are trapped by the floating-point unavailable exception vector and can be emulated in software.

Table 2-25 provides a list of the floating-point load instructions. (Floating-point instructions are not supported on the EC603e microprocessor.)

Table 2-25. Floating-Point Load Instructions

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

2.3.4.3.10 Floating-Point Store Instructions

There are three basic forms of the store instruction—single-precision, double-precision, and integer. The integer form is supported by the optional **stfiwx** instruction. Because the FPRs support only floating-point, double-precision format for floating-point data single-precision floating-point store instructions convert double-precision data to single-precision format before storing the operands. The conversion steps are described fully in “Floating-

Point Store Instructions” in Appendix D, “Floating-Point Models,” in *The Programming Environments Manual*.

Implementation Note—The PowerPC architecture defines store with update instructions with **r**A = 0 as an invalid form; however, the 603e treats this case as valid.

On the EC603e microprocessor, floating-point instructions are trapped by the floating-point unavailable exception vector and can be emulated in software.

Table 2-26 provides a list of the floating-point store instructions. (Floating-point instructions are not supported on the EC603e microprocessor.)

Table 2-26. Floating-Point Store Instructions

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

2.3.4.4 Branch and Flow Control Instructions

Branch instructions are executed by the branch processing unit (BPU). The BPU receives branch instructions from the fetch unit and performs condition register (CR) look-ahead operations on conditional branches to resolve them early, achieving the effect of a zero-cycle branch in many cases.

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

If an interlock is detected, the branch is considered unresolved and the direction of the branch is predicted using static branch prediction as described in “Conditional Branch Control” in Chapter 4, “Addressing Modes and Instruction Set Summary,” in *The Programming Environments Manual*. The interlock is monitored while instructions are fetched for the predicted branch. When the interlock is cleared, the branch processor determines whether the prediction was correct based on the value of the CR bit. If the prediction is correct, the branch is considered completed and instruction fetching continues.

If the prediction is incorrect, the fetched instructions are purged, and instruction fetching continues along the alternate path. See Chapter 8, “Instruction Timing,” in *The Programming Environments Manual* for more information about how branches are executed.

2.3.4.4.1 Branch Instruction Address Calculation

Branch instructions can alter the sequence of instruction execution. Instruction addresses are always assumed to be word aligned; the processor ignores the two low-order bits of the generated branch target address.

Branch instructions compute the effective address (EA) of the next instruction address using the following addressing modes:

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

2.3.4.4.2 Branch Instructions

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

Table 2-27. Branch Instructions

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

2.3.4.4.3 Condition Register Logical Instructions

Condition register logical instructions, shown in Table 2-28, and the Move Condition Register Field (**mcrf**) instruction are also defined as flow control instructions, although they are executed by the system register unit (SRU). Most instructions executed by the SRU are completion-serialized to maintain system state; that is, the instruction is held for execution in the SRU until all prior instructions issued have completed.

Table 2-28. Condition Register Logical Instructions

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

Note that if the LR update option is enabled for any of these instructions, these forms of the instructions are invalid in the 603e.

2.3.4.5 Trap Instructions

The trap instructions shown in Table 2-29 are provided to test for a specified set of conditions. If any of the conditions tested by a trap instruction are met, the system trap handler is invoked. If the tested conditions are not met, instruction execution continues normally.

Table 2-29. Trap Instructions

Name	Mnemonic	Operand Syntax
Trap Word Immediate	**twi**	TO,**r**A,SIMM
Trap Word	**tw**	TO,**r**A,**r**B

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

2.3.4.6 Processor Control Instructions

Processor control instructions are used to read from and write to the condition register (CR), machine state register (MSR), and special-purpose registers (SPRs), and to read from the time base register (TBU or TBL).

2.3.4.6.1 Move to/from Condition Register Instructions

Table 2-37 lists the instructions provided by the 603e for reading from or writing to the CR.

Table 2-30. Move to/from Condition Register Instructions

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

2.3.4.7 Memory Synchronization Instructions—UISA

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

The **sync** instruction delays execution of subsequent instructions until previous instructions have completed to the point that they can no longer cause an exception and until all previous memory accesses are performed globally; the **sync** operation is not broadcast onto the 603e bus interface. Additionally all load and store cache/bus activities initiated by prior instructions are completed. Touch load operations (**dcbt** and **dcbtst**) are required to complete at least through address translation, but not required to complete on the bus.

The functions performed by the **sync** instruction normally take a significant amount of time to complete; as a result, frequent use of this instruction may adversely affect performance. In addition, the number of cycles required to complete a **sync** instruction depends on system parameters and on the processor's state when the instruction is issued.

The proper paired use of the **lwarx** and **stwcx.** instructions allows programmers to emulate common semaphore operations such as "test and set," "compare and swap," "exchange memory," and "fetch and add." Examples of these semaphore operations can be found in Appendix E, "Synchronization Programming Examples," in *The Programming Environments Manual*. The **lwarx** instruction must be paired with an **stwcx.** instruction with the same effective address used for both instructions of the pair. Note that the reservation granularity is 32 bytes.

The concept behind the use of the **lwarx** and **stwcx.** instructions is that a processor may load a semaphore from memory, compute a result based on the value of the semaphore, and conditionally store it back to the same location (only if that location has not been modified since it was first read), and determine if the store was successful. The conditional store is performed based upon the existence of a reservation established by the preceding **lwarx** instruction. If the reservation exists when the store is executed, the store is performed and a bit is set in the CR. If the reservation does not exist when the store is executed, the target memory location is not modified and a bit is cleared in the CR.

If the store was successful, the sequence of instructions from the read of the semaphore to the store that updated the semaphore appear to have been executed atomically (that is, no other processor or mechanism modified the semaphore location between the read and the update), thus providing the equivalent of a real atomic operation. However, in reality, other processors may have read from the location during this operation. In the 603e, the reservations are made on behalf of aligned 32-byte sections of the memory address space.

The **lwarx** and **stwcx.** instructions require the EA to be aligned. Exception handling software should not attempt to emulate a misaligned **lwarx** or **stwcx.** instruction, because there is no correct way to define the address associated with the reservation.

In general, the **lwarx** and **stwcx.** instructions should be used only in system programs, which can be invoked by application programs as needed.

At most, one reservation exists simultaneously on any processor. The address associated with the reservation can be changed by a subsequent **lwarx** instruction. The conditional store is performed based upon the existence of a reservation established by the preceding **lwarx** regardless of whether the address generated by the **lwarx** matches that generated by the **stwcx.** instruction. A reservation held by the processor is cleared by one of the following:

- Executing an **stwcx.** instruction to any address
- Attempt by some other device to modify a location in the reservation granularity (32 bytes)

The **lwarx** and **stwcx.** instructions in write-through access mode do not cause a DSI exception.

Table 2-31 lists the UISA memory synchronization instructions for the 603e.

Table 2-31. Memory Synchronization Instructions—UISA

Name	Mnemonic	Operand Syntax
Load Word and Reserve Indexed	**lwarx**	**r**D,**r**A,**r**B
Store Word Conditional Indexed	**stwcx.**	**r**S,**r**A,**r**B
Synchronize	**sync**	—

2.3.5 PowerPC VEA Instructions

The PowerPC VEA describes the semantics of the memory model that can be assumed by software processes, and includes descriptions of the cache model, cache-control instructions, address aliasing, and other related issues.

2.3.5.1 Processor Control Instructions

In addition to the move to condition register instructions specified by the UISA, the VEA defines the Move from Time Base (**mftb**) instruction for reading the contents of the time base register. The **mftb** is a user-level instruction, it is shown in Table 2-32.

Simplified mnemonics are provided for the **mftb** instruction so it can be coded with the TBR name as part of the mnemonic rather than requiring it to be coded as an operand. The **mftb** instruction serves as both a basic and simplified mnemonic. Assemblers recognize an **mftb** mnemonic with two operands as the basic form, and an **mftb** mnemonic with one operand as the simplified form. Simplified mnemonics are also provided for Move from Time Base Upper (**mftbu**), which is a variant of the **mftb** instruction rather than of **mfspr**. The 603e ignores the extended opcode differences between **mftb** and **mfspr** by ignoring bit 25 of both instructions and treating them both identically. For more information refer to Appendix F, "Simplified Mnemonics," in *The Programming Environments Manual*.

Table 2-32. Move from Time Base Instruction

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

2.3.5.2 Memory Synchronization Instructions—VEA

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

Implementation Notes—The following describes how the 603e handles memory synchronization in the VEA.

- The Instruction Synchronize (**isync**) instruction causes the 603e to discard all prefetched instructions, wait for any preceding instructions to complete, and then branch to the next sequential instruction (which has the effect of clearing the pipeline behind the **isync** instruction).
- The Enforce In-Order Execution of I/O (**eieio**) instruction is used to ensure memory reordering of noncacheable memory access. Since the 603e does not reorder noncacheable memory accesses, the **eieio** instruction is treated as a no-op.

Table 2-31 lists the VEA memory synchronization instructions for the 603e.

Table 2-33. Memory Synchronization Instructions—VEA

Name	Mnemonic	Operand Syntax
Enforce In-Order Execution of I/O	**eieio**	—
Instruction Synchronize	**isync**	—

2.3.5.3 Memory Control Instructions—VEA

Memory control instructions include the following types:

- Cache management instructions
- Segment register manipulation instructions
- Translation lookaside buffer management instructions

This section describes the user-level cache management instructions defined by the VEA. See Section 2.3.6.3, “Memory Control Instructions—OEA,” for information about supervisor-level cache, segment register manipulation, and translation lookaside buffer management instructions.

The instructions listed in Table 2-34 provide user-level programs the ability to manage on-chip caches when they exist.

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

Note that when data address translation is disabled (MSR[DR] = 0), the Data Cache Block Set to Zero (**dcbz**) instruction allocates a cache block in the cache and may not verify that the physical address is valid. If a cache block is created for an invalid physical address, a machine check condition may result when an attempt is made to write that cache block back to memory. The cache block could be written back as a result of the execution of an instruction that causes a cache miss and the invalid addressed cache block is the target for replacement or a Data Cache Block Store (**dcbst**) instruction.

Note that any cache control instruction that generates an effective address that corresponds to a direct-store segment (SR[T] = 1) is treated as a no-op.

Table 2-34 lists the cache instructions that are accessible to user-level programs.

Table 2-34. User-Level Cache Instructions

Name	Mnemonic	Operand Syntax
Data Cache Block Touch	**dcbt**	**r**A,**r**B
Data Cache Block Touch for Store	**dcbtst**	**r**A,**r**B
Data Cache Block Set to Zero	**dcbz**	**r**A,**r**B
Data Cache Block Store	**dcbst**	**r**A,**r**B
Data Cache Block Flush	**dcbf**	**r**A,**r**B
Instruction Cache Block Invalidate	**icbi**	**r**A,**r**B

2.3.5.4 External Control Instructions

The external control instructions allow a user-level program to communicate with a special-purpose device. The MMU translation of the EA is not used to select the special-purpose device, as it is used in most instructions such as loads and stores. The EA is used instead as an address operand that is passed to the device over the address bus. Four other signals (the burst and size signals on the 60x bus) are used to select the device; these four signals output the 4-bit resource ID (RID) field that is located in the EAR register. Executing these instructions when MSR[DR] = 0 causes a programming error, and the physical address on the bus is undefined. Executing these instructions to a direct-store segment causes a DSI exception. The external control instructions are listed in Table 2-35.

Table 2-35. External Control Instructions

Name	Mnemonic	Operand Syntax
External Control In Word Indexed	**eciwx**	**r**D,**r**A,**r**B
External Control Out Word Indexed	**ecowx**	**r**S,**r**A,**r**B

2.3.6 PowerPC OEA Instructions

The PowerPC OEA includes the structure of the memory management model, supervisor-level registers, and the exception model.

2.3.6.1 System Linkage Instructions

This section describes the system linkage instructions (see Table 2-36). The **sc** instruction is a user-level instruction that permits a user program to call on the system to perform a service and causes the processor to take an exception. The Return from Interrupt (**rfi**) instruction is a supervisor-level instruction that is useful for returning from an exception handler.

Table 2-36. System Linkage Instructions

Name	Mnemonic	Operand Syntax
System Call	**sc**	—
Return from Interrupt	**rfi**	—

2.3.6.2 Processor Control Instructions—OEA

Processor control instructions are used to read from and write to the condition register (CR), machine state register (MSR), and special-purpose registers (SPRs), and to read from the time base register (TBU or TBL).

2.3.6.2.1 Move to/from Machine State Register Instructions

Table 2-37 lists the instructions provided by the 603e for reading from or writing to the MSR.

Table 2-37. Move to/from Machine State Register Instructions

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

2.3.6.2.2 Move to/from Special-Purpose Register Instructions

Simplified mnemonics are provided for the **mtspr** and **mfspr** instructions so they can be coded with the SPR name as part of the mnemonic rather than as a numeric operand. See Appendix F, "Simplified Mnemonics," in *The Programming Environments Manual* for simplified mnemonic examples. The **mtspr** and **mfspr** instructions are shown in Table 2-38.

Table 2-38. Move to/from Special-Purpose Register Instructions

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

For **mtspr** and **mfspr** instructions, the SPR number coded in assembly language does not appear directly as a 10-bit binary number in the instruction. The number coded is split into two 5-bit halves that are reversed in the instruction encoding, with the high-order 5 bits appearing in bits 16–20 of the instruction encoding and the low-order 5 bits in bits 11–15.

If the SPR field contains any value other than one of the values shown in Table 2-39, either the program exception handler is invoked or the results are boundedly undefined.

Table 2-39. Implementation-specific SPR Encodings (mfspr)

SPR*			Register Name
Decimal	spr[5–9]	spr[0–4]	
976	11110	10000	DMISS
977	11110	10001	DCMP
978	11110	10010	HASH1
979	11110	10011	HASH2
980	11110	10100	IMISS
981	11110	10101	ICMP

Table 2-39. Implementation-specific SPR Encodings (mfspr) (Continued)

SPR*			Register Name
Decimal	spr[5–9]	spr[0–4]	
982	11110	10110	RPA
1008	11111	10000	HID0
1009	11111	10001	HID1
1010	11111	10010	IABR

* Note that the order of the two 5-bit halves of the SPR number is reversed compared with actual instruction coding.

For **mtspr** and **mfspr** instructions, the SPR number coded in assembly language does not appear directly as a 10-bit binary number in the instruction. The number coded is split into two 5-bit halves that are reversed in the instruction, with the high-order 5 bits appearing in bits 16–20 of the instruction and the low-order 5 bits in bits 11–15.

Implementation Note—The 603e ignores the extended opcode differences between **mftb** and **mfspr** by ignoring TB[25] and treating both instructions identically.

2.3.6.3 Memory Control Instructions—OEA

This section describes memory control instructions, which include the following types:

- Cache management instructions
- Segment register manipulation instructions
- Translation lookaside buffer management instructions

2.3.6.3.1 Supervisor-Level Cache Management Instruction

Table 2-40 lists the only supervisor-level cache management instruction. See Section 2.3.5.3, “Memory Control Instructions—VEA,” for a description of cache instructions that provide user-level programs the ability to manage the on-chip caches. If the effective address references a direct-store segment, the instruction is treated as a no-op.

When data translation is disabled, MSR[DR] = 0, the **dcbz** instruction establishes a block in the cache and may not verify that the physical address is valid. If a block is created for an invalid real address, a machine check exception may result when an attempt is made to write that block back to memory. The block could be written back as the result of the execution of an instruction that causes a cache miss and the invalid address block is the target for replacement or as the result of a **dcbst** instruction.

Table 2-40. Supervisor-Level Cache Management Instruction

Name	Mnemonic	Operand Syntax
Data Cache Block Invalidate	**dcbi**	**r**A,**r**B

2.3.6.3.2 Segment Register Manipulation Instructions

The instructions listed in Table 2-41 provide access to the segment registers for the 603e. These instructions operate completely independently of the MSR[IR] and MSR[DR] bit settings. Refer to “Synchronization Requirements for Special Registers and TLBs” in Chapter 2, “Register Set,” in *The Programming Environments Manual* for serialization requirements and other recommended precautions to observe when manipulating the segment registers.

Table 2-41. Segment Register Manipulation Instructions

Name	Mnemonic	Operand Syntax
Move to Segment Register	**mtsr**	SR,**r**S
Move to Segment Register Indirect	**mtsrin**	**r**S,**r**B
Move from Segment Register	**mfsr**	**r**D,SR
Move from Segment Register Indirect	**mfsrin**	**r**D,**r**B

2.3.6.3.3 Translation Lookaside Buffer Management Instructions

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

Implementation Note—The 603e provides the ability to invalidate a TLB entry. The TLB Invalidate Entry (**tlbie**) instruction invalidates the TLB entry indexed by the EA, and operates on both the instruction and data TLBs simultaneously invalidating four TLB entries (both sets in each TLB). The index corresponds to bits 15–19 of the EA. To invalidate all entries within both TLBs, 32 **tlbie** instructions should be issued, incrementing this field by one each time.

The 603e provides two implementation-specific instructions (**tlbld** and **tlbli**) that are used by software table search operations following TLB misses to load TLB entries on-chip.

For more information on **tlbld** and **tlbli** refer to Section 2.3.8, “Implementation-Specific Instructions.”

Note that the **tlbia** instruction is not implemented on the 603e.

Refer to Chapter 5, “Memory Management” for more information about the TLB operations for the 603e. Table 2-42 lists the TLB instructions.

Table 2-42. Translation Lookaside Buffer Management Instructions

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

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

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

2.3.7 Recommended Simplified Mnemonics

To simplify assembly language programs, a set of simplified mnemonics is provided for some of the most frequently used operations (such as no-op, load immediate, load address, move register, and complement register). PowerPC compliant assemblers provide the simplified mnemonics listed in “Recommended Simplified Mnemonics” in Appendix F, “Simplified Mnemonics,” in *The Programming Environments Manual* and listed with some of the instruction descriptions in this chapter. Programs written to be portable across the various assemblers for the PowerPC architecture should not assume the existence of mnemonics not described in this document.

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

2.3.8 Implementation-Specific Instructions

This section provides a detailed look at the two 603e implementation-specific instructions—**tlbld** and **tlbli**.

tlbld — tlbld

Load Data TLB Entry — Integer Unit

tlbld **r**B

▨ Reserved

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

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

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

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

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

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

This is a supervisor-level instruction; it is also a 603e-specific instruction, and not part of the PowerPC instruction set.

Other registers altered:

- None

tlbli tlbli

Load Instruction TLB Entry Integer Unit

tlbld **r**B

▨ Reserved

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

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

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

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

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

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

This is a supervisor-level instruction; it is also a 603e-specific instruction, and not part of the PowerPC instruction set.

Other registers altered:

- None

Chapter 3
Instruction and Data Cache Operation

The PowerPC 603e microprocessor provides two 16-Kbyte, four-way set associative caches to allow the registers and execution units rapid access to instructions and data. Both the instruction and data caches are tightly coupled to the 603e's bus interface unit (BIU) to allow efficient access to the system memory controller and other bus masters. The 603e's load/store unit (LSU) is also directly coupled to the data cache to allow the efficient movement of data to and from the general-purpose and floating-point registers. (The floating-point register file is not supported on the EC603e microprocessor.)

Both the instruction and data caches have a block size of 32 bytes, and the data cache blocks can be snooped, or cast-out when the cache block is reloaded. The data cache is designed to adhere to a write-back policy, but the 603e allows control of cacheability, write-back policy, and memory coherency at the page and block level. Both caches use a least recently used (LRU) replacement policy. Burst fill operations to the caches result from cache misses, or in the case of the data cache, cache block write-back operations to memory. Note that in the PowerPC architecture, the term 'cache block', or simply 'block' when used in the context of cache implementations, refers to the unit of memory at which coherency is maintained. For the 603e, the block size is equivalent to the eight-word cache line. This value may be different for other PowerPC implementations.

The data cache is configured as 128 sets of four blocks. Each block consists of 32 bytes, two state bits, and an address tag. The two state bits implement the three-state MEI (modified/exclusive/invalid) protocol, a coherent subset of the standard four-state MESI protocol. Cache coherency is enforced by on-chip bus snooping logic. Since the 603e's data cache tags are single-ported, a simultaneous load or store and snoop access represent a resource contention. The snoop access is given first access to the tags. Load or store operations can be performed to the cache on the clock cycle immediately following a snoop access if the snoop misses; snoop hits may block the data cache for two or more cycles, depending on whether a copyback to main memory is required.

The instruction cache also consists of 128 sets of four blocks, and each block consists of 32 bytes, an address tag, and a valid bit. The instruction cache is only written as a result of a block fill operation on a cache miss. In the PID7v-603e, the instruction cache is blocked only until the critical load completes. The PID7v-603e supports instruction fetching from other instruction cache lines following the forwarding of the critical first double word of a cache line load operation. Successive instruction fetches from the cache line being loaded are forwarded, and accesses to other instruction cache lines can proceed during the cache line load operation. The instruction cache is not snooped, and cache coherency must be maintained by software. A fast hardware invalidation capability is provided to support cache maintenance.

The load/store unit provides the data transfer interface between the data cache and the GPRs and the FPRs (not supported by the EC603e microprocessor). The load/store unit provides all logic required to calculate effective addresses, handle data alignment to and from the data cache, and provides sequencing for load and store string and multiple operations. As shown in Figure 1-1, the caches provide a 64-bit interface to the instruction fetcher and load/store unit. Write operations to the data cache can be performed on a byte, half-word, word, or double-word basis.

The 603e's bus interface unit receives requests for bus operations from the instruction and data caches, and executes the operations according to the 603e bus protocol. The BIU provides address queues, prioritization and bus control logic. The BIU also captures snoop addresses for data cache, address queue, and memory reservation (**lwarx** and **stwcx**. instruction) operations. The BIU also contains a touch load address buffer used for address compares during load or store operations. All the data for the corresponding address queues (load and store data queues) is located in the data cache. The data queues are considered temporary storage for the cache and not part of the BIU.

On a cache miss, the 603e's cache blocks are loaded in four beats of 64 bits each when the 603e is configured with a 64-bit data bus; when the 603e is configured with a 32-bit bus, cache block loads are performed with eight beats of 32 bits each. The burst load is performed as critical double word first. The data cache is blocked to internal accesses until the load completes; the instruction cache allows sequential fetching during a cache block load. In the PID7v-603e, the critical double word is simultaneously written to the cache and forwarded to the requesting unit, thus minimizing stalls due to load delays. Note that the cache being filled cannot be accessed internally until the fill completes.

When address translation is enabled, the memory access is performed under the control of the page table entry used to translate the effective address. Each page table entry contains four mode control bits, W, I, M, and G, that specify the storage mode for all accesses translated using that particular page table entry. The W (write-through) and I (caching-inhibited) bits control how the processor executing the access uses its own cache. The M (memory coherence) bit specifies whether the processor executing the access must use the MEI (modified, exclusive, or invalid) cache coherence protocol to ensure all copies of the addressed memory location are kept consistent. The G (guarded memory) bit controls whether out-of-order data and instruction fetching is permitted.

The 603e maintains data cache coherency in hardware by coordinating activity between the data cache, the memory system, and the bus interface logic. As bus operations are performed on the bus by other bus masters, the 603e bus snooping logic monitors the addresses that are referenced. These addresses are compared with the addresses resident in the data cache. If there is a snoop hit, the 603e's bus snooping logic responds to the bus interface with the appropriate snoop status (for example, an $\overline{\text{ARTRY}}$). Additional snoop action may be forwarded to the cache as a result of a snoop hit in some cases (a cache push of modified data, or a cache block invalidation).

The 603e supports a fully-coherent 4-Gbyte physical memory address space. Bus snooping is used to drive the MEI three-state cache-coherency protocol that ensures the coherency of global memory with respect to the processor's cache. The MEI protocol is described in Section 3.6.1, "MEI State Definitions."

This chapter describes the organization of the 603e's on-chip instruction and data caches, the MEI cache coherency protocol, cache control instructions, various cache operations, and the interaction between the cache, load/store unit, and the bus interface unit. PID7v-603e specific information is noted where applicable.

3.1 Instruction Cache Organization and Control

The instruction fetcher accesses the instruction cache frequently in order to sustain the high throughput provided by the six-entry instruction dispatch queue.

3.1.1 Instruction Cache Organization

The organization of the instruction cache is shown in Figure 3-1. Each cache block contains eight contiguous words from memory that are loaded from an 8-word boundary (that is, bits A27–A31 of the effective addresses are zero); thus, a cache block never crosses a page boundary. Misaligned accesses across a page boundary can incur a performance penalty

Note that address bits A20–A26 provide an index to select a set. Bits A27–A31 select a byte within a block. 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. Note that the replacement algorithm is strictly an LRU algorithm; that is, the least recently used block is filled with new instructions on a cache miss.

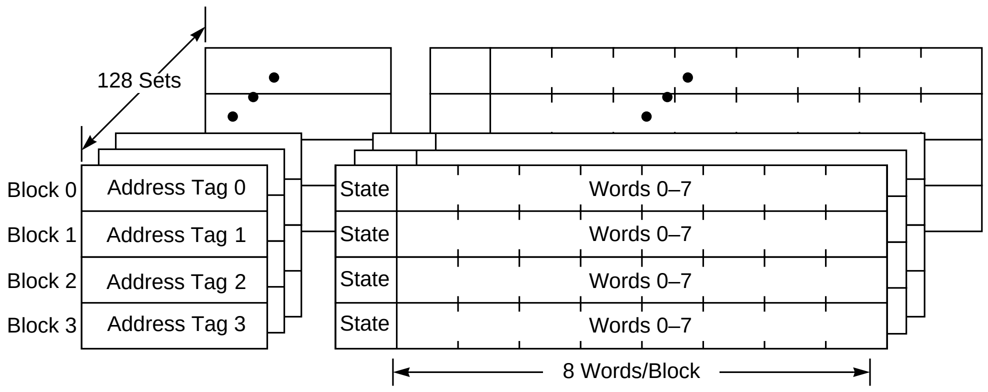


Figure 3-1. Instruction Cache Organization

3.1.2 Instruction Cache Fill Operations

The 603e's instruction cache blocks are loaded in four beats of 64 bits each, with the critical double word loaded first. The instruction cache allows sequential fetching during a cache block load. On a cache miss, the critical and following double words read from memory are simultaneously written to the instruction cache and forwarded to the dispatch queue, thus minimizing stalls due to cache fill latency. There is no snooping of the instruction cache. In the PID7v-603e, the critical double word is simultaneously written to the cache and forwarded to the requesting unit, thus minimizing stalls due to load delays.

3.1.3 Instruction Cache Control

In addition to instruction cache control instructions, the 603e provides several control bits in the HID0 register for the control of invalidating, disabling, and locking the instruction cache. In addition, the WIMG bits in the page tables also affect the cacheability of pages and whether or not the pages are considered guarded.

3.1.3.1 Instruction Cache Invalidation

While the 603e's instruction cache is automatically invalidated during a power-on or hard reset, assertion of the soft reset signal does not cause instruction cache invalidation. Software may invalidate the contents of the instruction cache using the instruction cache flash invalidate (ICFI) control bit in the HID0 register. Flash invalidation of the instruction cache is accomplished by setting and clearing the ICFI bit with two consecutive move to SPR operations to the HID0 register.

3.1.3.2 Instruction Cache Disabling

The instruction cache may be disabled through the use of the instruction cache enable (ICE) control bit in the HID0 register. When the instruction cache is in the disabled state, the cache tag state bits are ignored, and all accesses are propagated to the bus as single-beat transactions. The ICE bit is cleared during a power-on reset, causing the instruction cache to be disabled. The setting of the ICE bit must be preceded by an **isync** instruction to prevent the cache from being enabled or disabled while an instruction access is in progress.

3.1.3.3 Instruction Cache Locking

The contents of instruction cache may be locked through the use of the ILOCK control bit in the HID0 register. A locked instruction cache supplies instructions normally on a cache hit, but cache misses are treated as cache-inhibited accesses. The cache inhibited ($\overline{\text{CI}}$) signal is asserted if a cache access misses into a locked cache. The setting of the ILOCK bit in HID0 must be preceded by an **isync** instruction to prevent the instruction cache from being locked during an instruction access.

3.2 Data Cache Organization and Control

The data cache supplies data to the GPRs and FPRs (not supported on the EC603e microprocessor) by means of the load/store unit, and provides buffers for load and store bus operations. The data cache also provides storage for the cache tags required for memory coherency and performs the cache block replacement LRU function.

3.2.1 Data Cache Organization

The organization of the data cache is shown in Figure 3-2. Each cache block contains eight contiguous words from memory that are loaded from an 8-word boundary (that is, bits A27–A31 of the effective addresses are zero); thus, a cache block never crosses a page boundary. Misaligned accesses across a page boundary can incur a performance penalty.

Note that address bits A20–A26 provide an index to select a set. Bits A27–A31 select a byte within a block. 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. Note that the replacement algorithm is strictly an LRU algorithm; that is, the least recently used block is filled with new data on a cache miss.

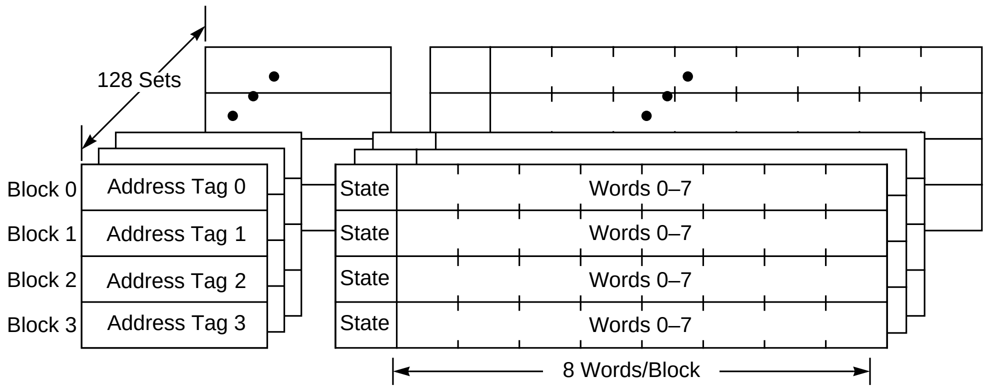


Figure 3-2. Data Cache Organization

3.2.2 Data Cache Fill Operations

The 603e’s cache blocks are loaded in four beats of 64 bits each when the 603e is configured with a 64-bit data bus; when the 603e is configured with a 32-bit bus, cache block loads are performed with eight beats of 32 bits each. The burst load is performed as critical double word first. The data cache is blocked to internal accesses until the load completes. In the PID7v-603e, the critical double word is simultaneously written to the cache and forwarded to the requesting unit, thus minimizing stalls due to load delays.

3.2.3 Data Cache Control

The 603e provides several means of data cache control through the use of the WIMG bits in the page tables, control bits in the HID0 register, and user- and supervisor-level cache control instructions. While memory page level cache control is provided by the WIMG bits, the on-chip data cache can be invalidated, disabled, locked, or broadcast by the control bits in the HID0 register described in this section. (Note that, user- and supervisor-level are referred to as problem and privileged state, respectively, in the architecture specification.)

3.2.3.1 Data Cache Invalidation

While the data cache is automatically invalidated when the 603e is powered up and during a hard reset, assertion of the soft reset signal does not cause data cache invalidation. Software may invalidate the contents of the data cache using the data cache flash invalidate (DCFI) control bit in the HID0 register. Flash invalidation of the data cache is accomplished by setting and clearing the DCFI bit in two consecutive store operations.

3.2.3.2 Data Cache Disabling

The data cache may be disabled through the use of the data cache enable (DCE) control bit in the HID0 register. When the data cache is in the disabled state, the cache tag state bits are ignored, and all accesses are propagated to the bus as single-beat transactions. The DCE bit is cleared on power-up, causing the data cache to be disabled. The setting of the DCE bit must be preceded by a **sync** instruction to prevent the cache from being enabled or disabled in the middle of a data access.

Note that while snooping is not performed when the data cache is disabled, cache operations (caused by the **dcbz**, **dcbf**, **dcbst**, and **dcbi** instructions) are not affected by disabling the cache, causing potential coherency errors. An example of this would be a **dcbf** instruction that hits a modified cache block in the disabled cache, causing a copyback to memory of potentially stale data.

Regardless of the state of HID0[DCE], load and store operations are assumed to be weakly ordered. Thus the LSU can perform load operations that occur later in the program ahead of store operations, even when the data cache is disabled. However, strongly ordered load and store operations can be enforced through the setting of the I bit (of the page WIMG bits) when address translation is enabled. Note that when address translation is disabled, the default WIMG bits cause the I bit to be cleared (accesses are assumed to be cacheable), and thus the accesses are weakly ordered. Refer to Section 3.5.2, “Caching-Inhibited Attribute (I),” for a description of the operation of the I bit and Section 5.2, “Real Addressing Mode,” for a description of the WIMG bits when address translation is disabled.

3.2.3.3 Data Cache Locking

The contents of the data cache may be locked through the use of the DLOCK control bit in the HID0 register. A locked data cache supplies data normally on a cache hit, but cache misses are treated as cache-inhibited accesses. The cache inhibited ($\overline{\text{CI}}$) signal is asserted if a cache access misses into a locked cache. The setting of the DLOCK bit in HID0 must be preceded by a **sync** instruction to prevent the data cache from being locked during a data access.

3.2.3.4 Data Cache Operations and Address Broadcasts

The execution of a **dcbz** instruction results in an address-only broadcast on the bus. Additionally, if the HID0[ABE] bit is set on a PID7v-603e processor, the execution of the **dcbf**, **dcbi**, and **dcbst** instructions will also cause an address-only broadcast. The ability of the PID7v-603e to optionally perform address-only broadcasts when executing the **dcbi**, **dcbf**, and the **dcbst** instructions allows the coherency management of an external copyback L2 cache. Note that these cache control instruction broadcasts are not snooped by the PID7v-603e.

3.2.4 Data Cache Touch Load Support

Touch load operations allow an instruction stream to prefetch data from memory prior to a cache miss. The 603e supports touch load operations through a temporary cache block buffer located between the BIU and the data cache. The cache block buffer is essentially a floating cache block that is loaded by the BIU on a touch load operation, and is then read by a load instruction that requests that data. After a touch load completes on the bus, the BIU continues to compare the touch load address with subsequent load requests from the data cache. If the load address matches the touch load address in the BIU, the data is forwarded to the data cache from the touch load buffer, the read from memory is canceled, and the touch load address buffer is invalidated.

To avoid the storage of stale data in the touch load buffer, touch load requests that are mapped as write-through or caching-inhibited by the MMU are treated as no-ops by the BIU. Also, subsequent load instructions after a touch load that are mapped as write-through or caching-inhibited do not hit in the touch load buffer, and cause the touch load buffer to be invalidated on a matching address.

While the 603e provides only a single cache block buffer, other PowerPC microprocessor implementations may provide buffering for more than one cache block. Programs written for other implementations may issue several **dcbt** or **dcbtst** instructions sequentially, reducing the performance if executed on the 603e. To improve performance in these situations, the NOOPTI bit (bit 31) in the HID0 register may be set. This causes the **dcbt** and **dcbtst** instructions to be treated as no-ops, cause no bus activity, and incur only one processor clock cycle of execution latency. The default state of the NOOPTI bit is cleared after a power-on reset operation, enabling the use of the **dcbt** and **dcbtst** instructions.

3.3 Basic Data Cache Operations

This section describes the three types of operations that can occur to the data cache, and how these operations are implemented in the 603e.

3.3.1 Data Cache Fill

A cache block is filled after a read miss or write miss (read-with-intent-to-modify) occurs in the cache. The cache block that corresponds to the missed address is updated by a burst transfer of the data from system memory. Note that if a read miss occurs in a system with multiple bus masters, and the data is modified in another cache, the modified data is first written to external memory before the cache fill occurs.

3.3.2 Data Cache Cast-Out Operation

The 603e uses an LRU replacement algorithm to determine which of the four possible cache locations should be used for a cache update on a cache miss. Adding a new block to the cache causes any modified data associated with the least recently used element to be written back, or cast out, to system memory to maintain memory coherence.

3.3.3 Cache Block Push Operation

When a cache block in the 603e is snooped and hit by another bus master and the data is modified, the cache block must be written to memory and made available to the snooping device. The cache block that is hit is said to be pushed out onto the bus. The 603e supports two kinds of push operations—normal push operations and enveloped high-priority push operations, which are described in Section 3.6.9, “Enveloped High-Priority Cache Block Push Operation.”

3.4 Data Cache Transactions on Bus

The 603e transfers data to and from the data cache in single-beat transactions of two words, or in four-beat transactions of eight words which fill a cache block.

3.4.1 Single-Beat Transactions

Single-beat bus transactions can transfer from one to eight bytes to or from the 603e. Single-beat transactions can be caused by cache write-through accesses, caching-inhibited accesses (I bit of the WIMG bits for the page is set), or accesses when the cache is disabled (HID0[DCE] bit is cleared), and can be misaligned.

3.4.2 Burst Transactions

Burst transactions on the 603e always transfer eight words of data at a time, and are aligned to a double-word boundary. The 603e transfer burst ($\overline{\text{TBST}}$) output signal indicates to the system whether the current transaction is a single-beat transaction or four-beat burst transfer. Burst transactions have an assumed address order. For cacheable read operations or cacheable, non-write-through write operations that miss the cache, the 603e presents the double-word aligned address associated with the load or store instruction that initiated the transaction.

As shown in Figure 3-3, this quad word contains the address of the load or store that missed the cache. This minimizes latency by allowing the critical code or data to be forwarded to the processor before the rest of the block is filled. For all other burst operations, however, the entire block is transferred in order (oct-word aligned). Critical-double-word-first fetching on a cache miss applies to both the data and instruction cache.

3.4.3 Access to Direct-Store Segments

The 603e does not provide support for access to direct-store segments. Operations attempting to access a direct-store segment will invoke a DSI exception. For additional information about DSI exceptions, refer to Section 4.5.3, "DSI Exception (0x00300).”

603e Cache Address
Bits (27…28)

0 0	0 1	1 0	1 1
A	B	C	D

If the address requested is in double word A, the address placed on the bus is that of double-word A, and the four data beats are ordered in the following manner:

Beat

0	1	2	3
A	B	C	D

If the address requested is in double word C, the address placed on the bus will be that of double-word C, and the four data beats are ordered in the following manner:

Beat

0	1	2	3
C	D	A	B

Figure 3-3. Double-Word Address Ordering—Critical Double Word First

3.5 Memory Management/Cache Access Mode Bits— W, I, M, and G

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

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

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

Careless specification and use of these bits may create situations where coherency paradoxes are observed by the processor. In particular, this can happen when the state of these bits is changed without appropriate precautions being taken (for example, when flushing the pages that correspond to the changed bits from the caches of all processors in the system is required, or when the address translations of aliased physical addresses (referred to as real addresses in the architecture specification) specify different values for any of the WIM bits). The 603e considers either of these cases to be a programming error which may compromise the coherency of memory. These paradoxes can occur within a single processor or across several devices, as described in Section 3.6.4.1, “Coherency in Single-Processor Systems.”

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

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

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

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

Note that for accesses performed with direct address translation (MSR[IR] = 0 or MSR[DR] = 0 for instruction or data access, respectively), the WIMG bits are automatically generated as 0b0011 (the data is write-back, caching is enabled, memory coherency is enforced, and memory is guarded).

3.5.1 Write-Through Attribute (W)

When an access is designated as write-through (W = 1), if the data is in the cache, a store operation updates the cached copy of the data. In addition, the update is written to the external memory location (as described below).

While the PowerPC architecture permits multiple store instructions to be combined for write-through accesses except when the store instructions are separated by a **sync** or **eieio** instruction, the 603e does not implement this “combined store” capability. Note that a store operation that uses the write-through attribute may cause any part of valid data in the cache to be written back to main memory.

The definition of the external memory location to be written to in addition to the on-chip cache depends on the implementation of the memory system but can be illustrated by the following examples:

- RAM—The store is sent to the RAM controller to be written into the target RAM.
- I/O device—The store is sent to the memory-mapped I/O control hardware to be written to the target register or memory location.

In systems with multilevel caching, the store must be written to at least a depth in the memory hierarchy that is seen by all processors and devices.

Accesses that correspond to W = 0 are considered write-back. For this case, although the store operation is performed to the cache, it is only made to external memory when a copy-back operation is required. Use of the write-back mode (W = 0) can improve overall performance for areas of the memory space that are seldom referenced by other masters in the system.

3.5.2 Caching-Inhibited Attribute (I)

If I = 1, the memory access is completed by referencing the location in main memory, bypassing the on-chip cache. During the access, the addressed location is not loaded into the cache nor is the location allocated in the cache. It is considered a programming error if a copy of the target location of an access to caching-inhibited memory is resident in the cache. Software must ensure that the location has not been previously loaded into the cache, or, if it has, that it has been flushed from the cache.

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. Note that the **eieio** is treated as a no-op by the 603e.

The caching-inhibited (I) bit in the 603e controls whether load and store operations are strongly or weakly ordered. If an I/O device requires load and store accesses to occur in program order, then the I bit for the page must be set.

3.5.3 Memory Coherency Attribute (M)

This attribute is provided to allow improved performance in systems where hardware-enforced coherency is relatively slow, and software is able to enforce the required coherency. When M = 0, the processor does not enforce data coherency. When M = 1, the processor enforces data coherency and the corresponding access is considered to be a global access.

When the M attribute is set, and the access is performed, the global signal is asserted to indicate that the access is global. Snooping devices affected by the access must then respond to this global access if their data is modified by asserting $\overline{\text{ARTRY}}$, and updating the memory location.

Because instruction memory does not have to be consistent with data memory, the 603e ignores the M attribute for instruction accesses.

3.5.4 Guarded Attribute (G)

When the guarded bit is set, the memory area (block or page) is designated as guarded, meaning that the processor will perform out-of-order accesses to this area of memory, only as follows:

- Out-of-order load operations from guarded memory areas are performed only if the corresponding data is resident in the cache.
- The processor prefetches from guarded areas, but only when required, and only within the memory boundary dictated by the cache block. That is, if an instruction is certain to be required for execution by the program, it is fetched and the remaining instructions in the block may be prefetched, even if the area is guarded.

This setting can be used to protect certain memory areas from read accesses made by the processor that are not dictated directly by the program. If there are areas of memory that are not fully populated (in other words, there are holes in the memory map within this area), this setting can protect the system from undesired accesses caused by out-of-order load operations or instruction prefetches that could lead to the generation of the machine check exception. Also, the guarded bit can be used to prevent out-of-order load operations or prefetches from occurring to certain peripheral devices that produce undesired results when accessed in this way.

3.5.5 W, I, and M Bit Combinations

Table 3-1 summarizes the six combinations of the WIM bits. Note that either a zero or one setting for the G bit is allowed for each of these WIM bit combinations.

Table 3-1. Combinations of W, I, and M Bits

WIM Setting	Meaning
000	Data may be cached. Loads or stores whose target hits in the cache use that entry in the cache. Memory coherency is not enforced by hardware.
001	Data may be cached. Loads or stores whose target hits in the cache use that entry in the cache. Memory coherency is enforced by hardware.
010	Caching is inhibited. The access is performed to external memory, completely bypassing the cache. Memory coherency is not enforced by hardware.
011	Caching is inhibited. The access is performed to external memory, completely bypassing the cache. Memory coherency must be enforced by external hardware (processor provides hardware indication that access is global).
100	Data may be cached. Load operations whose target hits in the cache use that entry in the cache. Stores are written to external memory. The target location of the store may be cached and is updated on a hit. Memory coherency is not enforced by hardware.
101	Data may be cached. Load operations whose target hits in the cache use that entry in the cache. Stores are written to external memory. The target location of the store may be cached and is updated on a hit. Memory coherency is enforced by hardware.

3.5.5.1 Out-of-Order Execution and Guarded Memory

Out-of-order execution occurs when the 603e performs operations in advance in case the result is needed. Typically, these operations are performed by otherwise idle resources; thus if a result is not required, it is ignored and the out-of-order operation incurs no time penalty (typically).

Supervisor-level programs designate memory as guarded on a block or page level. Memory is designated as guarded if it may not be “well-behaved” with respect to out-of-order operations.

For example, the memory area that contains a memory-mapped I/O device may be designated as guarded if an out-of-order load or instruction fetch performed to such a device might cause the device to perform unexpected or incorrect operations. Another example of memory that should be designated as guarded is the area that corresponds to the device that resides at the highest implemented physical address (as it has no successor and out-of-order sequential operations such as instruction prefetching may result in a machine check exception). In addition, areas that contain holes in the physical memory space may be designated as guarded.

3.5.5.2 Effects of Out-of-Order Data Accesses

Most data operations may be performed out-of-order, as long as the machine appears to follow a simple sequential model. However, the following out-of-order operations do not occur:

- Out-of-order loading from guarded memory (G = 1) does not occur. However, when a load or store operation is required by the program, the entire cache block(s) containing the referenced data may be loaded into the cache.
- Out-of-order store operations that alter the state of the target location do not occur.
- No errors except machine check exceptions are reported due to the out-of-order execution of an instruction until it is known that execution of the instruction is required.

Machine check exceptions resulting solely from out-of-order execution (from nonguarded memory) may be reported. When an out-of-order instruction's result is abandoned, only one side effect (other than a possible machine check) may occur—the referenced bit (R) in the corresponding page table entry (and TLB entry) can be set due to an out-of-order load operation. See Chapter 4, “Exceptions,” for more information on the machine check exception.

Thus an out-of-order load or store instruction will not access guarded memory unless one of the following conditions exist:

- The target memory item is resident in an on-chip cache. In this case, the location may be accessed from the cache or main memory.
- The target memory item is cacheable (I = 0) and it is guaranteed that the load or store is in the execution path (assuming there are no intervening exceptions). In this case, the entire cache block containing the target may be loaded into the cache.
- The target memory is cache-inhibited (I = 1), the load or store instruction is in the execution path, and it is guaranteed that no prior instructions can cause an exception.

3.5.5.3 Effects of Out-of-Order Instruction Fetches

To avoid instruction fetch delay, the processor typically fetches instructions ahead of those currently being executed. Such instruction prefetching is said to be out-of-order in that prefetched instructions may not be executed due to intervening branches or exceptions.

During instruction prefetching, no errors except machine check exceptions are reported due to the out-of-order fetching of an instruction until it is known that execution of the instruction is required.

Machine check exceptions resulting solely from out-of-order execution (from nonguarded memory) may be reported. When an out-of-order instruction's result is abandoned, only one side effect (other than a possible machine check) may occur—the referenced bit (R) in the corresponding page table entry (and TLB entry) can be set due to an out-of-order load operation. See Chapter 4, “Exceptions,” for more information on the machine check exception.

Instruction fetching from guarded memory is not permitted.

3.6 Cache Coherency—MEI Protocol

The primary objective of a coherent memory system is to provide the same image of memory to all devices using the system. Coherency allows synchronization and cooperative use of shared resources. Otherwise, multiple copies of a memory location, some containing stale values, could exist in a system resulting in errors when the stale values are used. Each potential bus master must follow rules for managing the state of its cache.

The 603e cache coherency protocol is a coherent subset of the standard MESI four-state cache protocol that omits the shared state. Since data cannot be shared, the 603e signals all cache block fills as if they were write misses (read-with-intent-to-modify), which flushes the corresponding copies of the data in all caches external to the 603e prior to the 603e’s cache block fill operation. Following the cache block load, the 603e is the exclusive owner of the data and may write to it without a bus broadcast transaction.

To maintain this coherency, all global reads observed on the bus by the 603e are snooped as if they were writes, causing the 603e to write a modified cache block back to memory and invalidate the cache block, or simply invalidate the cache block if it is unmodified. The exception to this rule occurs when a snooped transaction is a caching-inhibited read (either burst or single-beat, where TT[0–4] = X1010; see Table 7-1 for clarification), in which case the 603e does not invalidate the snooped cache block. If the cache block is modified, the block is written back to memory, and the cache block is marked exclusive unmodified. If the cache block is marked exclusive unmodified when snooped, no bus action is taken, and the cache block remains in the exclusive unmodified state. This treatment of caching-inhibited reads decreases the possibility of data thrashing by allowing noncaching devices to read data without invalidating the entry from the 603e’s data cache.

3.6.1 MEI State Definitions

The 603e’s data cache characterizes each 32-byte block it contains as being in one of three MEI states. Addresses presented to the cache are indexed into the cache directory with bits A20–A26, and the upper-order 20 bits from the physical address translation (PA0–PA19) are compared against the indexed cache directory tags. If neither of the indexed tags matches, the result is a cache miss. If a tag matches, a cache hit occurred and the directory indicates the state of the cache block through two state bits kept with the tag. The three possible states for a cache block in the cache are the modified state (M), the exclusive state (E), and the invalid state (I). The three MEI states are defined in Table 3-2.

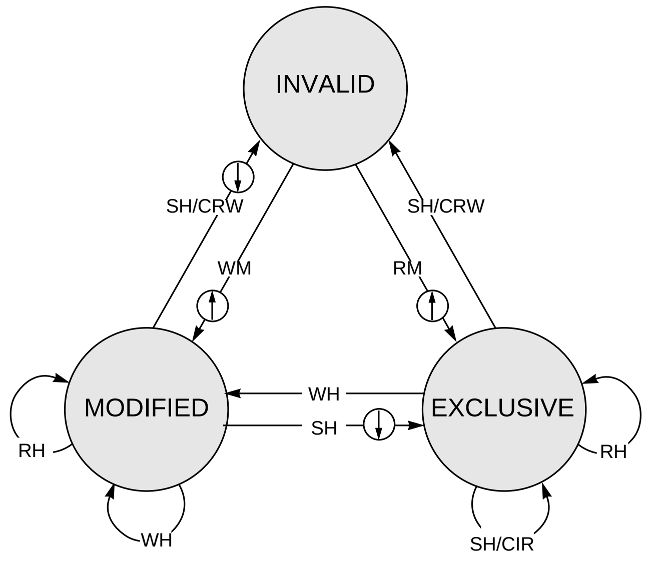
Table 3-2. MEI State Definitions

MEI State	Definition
Modified (M)	The addressed cache block is valid in the cache and only in the cache. The cache block is modified with respect to system memory—that is, the modified data in the cache block has not been written back to memory.
Exclusive (E)	The addressed block is in this cache only. The data in this cache block is consistent with system memory.
Invalid (I)	This state indicates that the addressed cache block is not resident in the cache.

3.6.2 MEI State Diagram

The 603e provides dedicated hardware to provide memory coherency by snooping bus transactions. The address retry capability of the 603e enforces the MEI protocol, as shown in Figure 3-4. Figure 3-4 assumes that the WIM bits for the page or block are set to 001; that is, write-back, caching-not-inhibited, and memory coherency enforced.

Section 3.10, “MEI State Transactions,” provides a detailed list of MEI transitions for various operations and WIM bit settings.



BUS TRANSACTIONS

SH	= Snoop Hit
RH	= Read Hit
RM	= Read Miss
WH	= Write Hit
WM	= Write Miss
SH/CRW	= Snoop Hit, Cacheable Read/Write
SH/CIR	= Snoop Hit, Cache Inhibited Read

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

3.6.3 MEI Hardware Considerations

While the 603e provides the hardware required to monitor bus traffic for coherency, the 603e data cache tags are single ported, and a simultaneous load or store and snoop access represent a resource conflict. In general, the snoop access has highest priority and is given first access to the tags. The load or store access will then occur on the clock following the snoop. The snoop is not given priority into the tags when the snoop coincides with a tag write (for example, validation after a cache block load). In these situations, the snoop is retried and must re-arbitrate before the lookup is possible.

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

Note that it is possible for a snoop to hit a modified cache block that is already in the process of being written to the copyback buffer for replacement purposes. If this happens, the 603e retries the snoop, and raises the priority of the cast-out operation to allow it to go to the bus before the cache block fill.

The global ($\overline{\text{GBL}}$) signal, asserted as part of the address attribute field during a bus transaction, enables the snooping hardware of the 603e. Address bus masters assert $\overline{\text{GBL}}$ to indicate that the current transaction is a global access (that is, an access to memory shared by more than one device). If $\overline{\text{GBL}}$ is not asserted for the transaction, that transaction is not snooped by the 603e. Note that the $\overline{\text{GBL}}$ signal is not asserted for instruction fetches, and that $\overline{\text{GBL}}$ is asserted for all data read or write operations when using direct address translation. (Note that direct address translation is referred to as the real addressing mode, not the direct-store segment, in the architecture specification.)

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 enforces coherency and can use considerable bus bandwidth if much data is shared. Therefore, available bus bandwidth can decrease as more traffic is marked global.

The 603e snoops a transaction if the transfer start ($\overline{\text{TS}}$) and $\overline{\text{GBL}}$ signals are asserted together in the same bus clock (this is a *qualified* snooping condition). No snoop update to the 603e cache occurs if the snooped transaction is not marked global. Also, because cache block cast-outs and snoop pushes do not require snooping, the $\overline{\text{GBL}}$ signal is not asserted for these operations.

When the 603e detects a qualified snoop condition, the address associated with the $\overline{\text{TS}}$ signal is compared with the cache tags. Snooping finishes if no hit is detected. If, however, the address hits in the cache, the 603e reacts according to the MEI protocol shown in Figure 3-4.

To facilitate external monitoring of the internal cache tags, the cache set entry signals (CSE[0–1]) represent in binary the cache set being replaced on read operations (including read-with-intent-to-modify operations). The CSE[0–1] signals do not apply for write operations to memory, or during non-cacheable or touch load operations. Note that these signals are valid only for 603e burst operations. Table 3-3 shows the CSE[0–1] (cache set entry) encodings.

Table 3-3. CSE[0–1] Signal Encoding

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

3.6.4 Coherency Precautions

The 603e supports a three-state coherency protocol that supports the modified, exclusive, and invalid (MEI) cache states. This protocol is a compatible subset of the MESI four-state protocol and operates coherently in systems that contain four-state caches. In addition, the 603e does not broadcast cache operations caused by cache instructions. They are intended for the management of the local cache but not for other caches in the system.

3.6.4.1 Coherency in Single-Processor Systems

The following situations concerning coherency can be encountered within a single-processor system:

- Load or store to a caching-inhibited page (WIM = 0bX1X) and a cache hit occurs

  Caching is inhibited for this page (I = 1)—Load or store operations to a caching-inhibited page that hit in the cache cause boundedly undefined results.

- Store to a page marked write-through (WIM = 0b10X) and a cache read hit to a modified cache block

  This page is marked as write-through (W = 1)—The 603e pushes the modified cache block to memory and the block remains marked modified (M).

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

3.6.5 Load and Store Coherency Summary

Table 3-4 provides a summary of memory coherency actions performed by the 603e on load operations. Noncacheable cases are not part of this table.

Table 3-4. Memory Coherency Actions on Load Operations

Cache State	Bus Operation	$\overline{\text{ARTRY}}$	Action
M	None	Don’t care	Read from cache
E	None	Don’t care	Read from cache
I	Read	Negated	Load data and mark E
I	Read	Asserted	Retry read operation

Table 3-5 provides an overview of memory coherency actions on store operations. This table does not include noncacheable or write-through cases. The read-with-intent-to-modify (RWITM) examples involve selecting a replacement class and casting-out modified data that may have resided in that replacement class.

Table 3-5. Memory Coherency Actions on Store Operations

Cache State	Bus Operation	$\overline{\text{ARTRY}}$	Action
M	None	Don't care	Modify cache
E	None	Don't care	Modify cache, mark M
I	RWITM	Negated	Load data, modify it, mark M
I	RWITM	Asserted	Retry the RWITM

3.6.6 Atomic Memory References

The Load Word and Reserve Indexed (**lwarx**) and Store Word Conditional Indexed (**stwcx.**) instructions provide an atomic update function for a single, aligned word of memory. While an **lwarx** instruction will normally be paired with an **stwcx.** instruction with the same effective address, an **stwcx.** instruction to any address will cancel the reservation. For detailed information on these instructions, refer to Chapter 2, “Programming Model,” in this book and Chapter 8, “Instruction Set,” in *The Programming Environments Manual*.

3.6.7 Cache Reaction to Specific Bus Operations

There are several bus transaction types defined for the 603e bus. The 603e must snoop these transactions and perform the appropriate action to maintain memory coherency as shown in Table 3-6. A processor may assert $\overline{\text{ARTRY}}$ for any bus transaction due to internal conflicts that prevent the appropriate snooping. The transactions in Table 3-6 correspond to the transfer type signals TT[0–4], which are described in Section 7.2.4.1, “Transfer Type (TT[0–4]).”

Table 3-6. Response to Bus Transactions

Snooped Transaction	603e Response
Clean block	No action is taken.
Flush block	No action is taken.
Write-with-flush Write-with-flush-atomic	Write-with-flush and write-with-flush-atomic operations occur after the processor issues a store or **stwcx.** instruction, respectively. • If the addressed block is in the exclusive state, the address snoop forces the state of the addressed block to invalid. • If the addressed block is in the modified state, the address snoop causes $\overline{\text{ARTRY}}$ to be asserted and initiates a push of the modified block out of the cache and changes the state of the block to invalid. • The execution of an **stwcx.** instruction cancels the reservation associated with any address.
Kill block	The kill block operation is an address-only bus transaction initiated when a **dcbz** instruction is executed; when snooped by the 603e, the addressed cache block is invalidated if in the E state, or flushed to memory and invalidated if in the M state, and any associated reservation is canceled.
Write-with-kill	In a write-with-kill operation, the processor snoops the cache for a copy of the addressed block. If one is found, an additional snoop action is initiated internally and the cache block is forced to the I state, killing modified data that may have been in the block. Any reservation associated with the block is also canceled.
Read Read-atomic	The read operation is used by most single-beat and burst read operations on the bus. All burst reads observed on the bus are snooped as if they were writes, causing the addressed cache block to be flushed. A read on the bus with the $\overline{\text{GBL}}$ signal asserted causes the following responses: • If the addressed block in the cache is invalid, the 603e takes no action. • If the addressed block in the cache is in the exclusive state, the block is invalidated. • If the addressed block in the cache is in the modified state, the block is flushed to memory and the block is invalidated. • If the snooped transaction is a caching-inhibited read, and the block in the cache is in the exclusive state, the snoop causes no bus activity and the block remains in the exclusive state. If the block is in the cache in the modified state, the 603e initiates a push of the modified block out to memory and marks the cache block as exclusive. Read atomic operations appear on the bus in response to **lwarx** instructions and generate the same snooping responses as read operations.
Read-with-intent-to-modify (RWITM) RWITM-atomic	A RWITM operation is issued to acquire exclusive use of a memory location for the purpose of modifying it. • If the addressed block is invalid, the 603e takes no action. • If the addressed block in the cache is in the exclusive state, the 603e initiates an additional snoop action to change the state of the cache block to invalid. • If the addressed block in the cache is in the modified state, the block is flushed to memory and the block is invalidated. The RWITM atomic operations appear on the bus in response to **stwcx.** instructions and are snooped like RWITM instructions.
sync	No action is taken.
TLB invalidate	No action is taken.

3.6.8 Operations Causing $\overline{\text{ARTRY}}$ Assertion

The following scenarios cause the 603e to assert the $\overline{\text{ARTRY}}$ signal:

- Snoop hits to a block in the M state (flush or clean)

  This case is a normal snoop hit and will result in $\overline{\text{ARTRY}}$ being asserted if the snooped transaction was a “flush” or “clean” request. If the snooped transaction was a “kill” request, $\overline{\text{ARTRY}}$ will not be asserted.

- Snoop attempt during the last $\overline{\text{TA}}$ of a cache line fill

  In no-$\overline{\text{DRTRY}}$ mode, during the cycle that the last $\overline{\text{TA}}$ is asserted to the 603e on a cache line fill, the tag is being written to its new state by the 603e and is not accessible. This will result in a collision being signaled by asserting $\overline{\text{ARTRY}}$. With $\overline{\text{DRTRY}}$ enabled, the cache tags are inaccessible to a snoop operation one cycle after the last $\overline{\text{TA}}$.

- Snoop hit after the first $\overline{\text{TA}}$ of a burst load operation

  After the first $\overline{\text{TA}}$ of a burst load operation, the data tags are committed to being written; snoop operations cannot be serviced until the load completes, thereby causing the assertion of $\overline{\text{ARTRY}}$.

- Snoop hits to line in the cast-out buffer

  The 603e's cast-out buffer is kept coherent with main memory, and snoop operations that hit in the cast-out buffer will cause the assertion of $\overline{\text{ARTRY}}$.

- Snoop attempt during cycles that **dcbz** instruction or load or store operation is updating the tag

  During the execution of a **dcbz** instruction or during a load or store operation that requires a cache line cast-out, the cache tags will be inaccessible during the first and last cycle of the operation.

- Snoop attempt during the cycle that a **dcbf** or **dcbst** instruction is updating the tag

  If the EA of a **dcbf** or **dcbst** instruction hits in the cache, the tag will be changed to its new state. During that clock, the tag is not accessible and snoop transactions during that cycle will cause the assertion of $\overline{\text{ARTRY}}$.

3.6.9 Enveloped High-Priority Cache Block Push Operation

In cases where the 603e has completed the address tenure of a read operation, and then detects a snoop hit to a modified cache block by another bus master, the 603e provides a high-priority push operation. If the address snooped is the same as the address of the data to be returned by the read operation, $\overline{\text{ARTRY}}$ is asserted one or more times until the data tenure of the read operation is completed. The cache block push transaction can be enveloped within the address and data tenures of a read operation. This feature prevents deadlocks in system organizations that support multiple memory-mapped buses.

More specifically, the 603e internally detects the scenario where a load request is outstanding and the processor has pipelined a write operation on top of the load. Normally, when the data bus is granted to the 603e, the resulting data bus tenure is used for the load operation. The enveloped high-priority cache block push feature defines a bus signal, the data bus write only qualifier ($\overline{\text{DBWO}}$), which, when asserted with a qualified data bus grant, indicates that the resulting data tenure should be used for the store operation instead. This signal is described in Section 8.10, "Using Data Bus Write Only." Note that the enveloped copyback operation is an internally pipelined bus operation.

3.7 Cache Control Instructions

Software must use the appropriate cache management instructions to ensure that caches are kept consistent when data is modified by the processor. When a processor alters a memory location that may be contained in an instruction cache, software must ensure that updates to memory are visible to the instruction fetching mechanism. Although the instructions to enforce coherency vary among implementations and hence operating systems should provide a system service for this function, the following sequence is typical:

1. **dcbst** (update memory)
2. **sync** (wait for update)
3. **icbi** (invalidate copy in cache)
4. **isync** (invalidate copy in own instruction buffer)

These operations are necessary because the processor does not maintain instruction memory coherent with data memory. Software is responsible for enforcing coherency of instruction caches and data memory. Since instruction fetching may bypass the data cache, changes made to items in the data cache may not be reflected in memory until after the instruction fetch completes.

The PowerPC architecture defines instructions for controlling both the instruction and data caches when they exist. The 603e interprets the cache control instructions (**icbi**, **dcbi**, **dcbt**, **dcbz**, **dcbst**) as if they pertain only to the 603e's caches. They are not intended for use in managing other caches in the system.

The **dcbz** instruction causes an address-only broadcast on the bus if the contents of the block are from a page marked global through the WIMG bits. This broadcast is performed for coherency reasons; the **dcbz** instruction is the only cache control instruction that can allocate and take new ownership of a line. Note that if the HID0[ABE] bit is set on a PID7v-603e processor, the execution of the **dcbf**, **dcbi**, and **dcbst** instructions will also cause an address-only broadcast. The **dcbz** instruction is also the only cache operation that is snooped by the 603e. The cache instructions are intended primarily for the management of the on-chip cache, and do not perform address-only broadcasts for the maintenance of other caches in the system. The ability of the PID7v-603e to optionally perform address-only broadcasts when executing the **dcbi**, **dcbf**, and the **dcbst** instructions allows the coherency management of an external copyback L2 cache.

The other instructions do not broadcast either for the purpose of invalidating or flushing other caches in the system or for managing system resources. Any bus activity caused by these instructions is the direct result of performing the operation in the 603e cache. Note that a data access exception is generated if the effective address of a **dcbi**, **dcbst**, **dcbf**, or **dcbz** instruction cannot be translated due to the lack of a TLB entry. (Note that exceptions are referred to as interrupts in the architecture specification.)

Note that in the PowerPC architecture, the term ‘cache block’, or simply ‘block’ when used in the context of cache implementations, refers to the unit of memory at which coherency is maintained. For the 603e this is the eight-word cache line. This value may be different for other PowerPC implementations. In-depth descriptions of coding these instructions is provided in Chapter 3, “Addressing Modes and Instruction Set Summary,” and Chapter 10, “Instruction Set,” in *The Programming Environments Manual*.

3.7.1 Data Cache Block Invalidate (dcbi) Instruction

If the block containing the byte addressed by the EA is in the data cache, the cache block is invalidated regardless whether the block is in the exclusive or modified state. If HID0[ABE] is set on a PID7v-603e when a **dcbi** instruction is executed, the PID7v-603e will perform an address-only bus transaction. The **dcbi** instruction can only be executed when the 603e is in the supervisor state.

3.7.2 Data Cache Block Touch (dcbt) Instruction

This instruction provides a method for improving performance through the use of software-initiated prefetch hints. The 603e performs the fetch for the cases when the address hits in the TLB or the BAT registers, and when it is a permitted load access from the addressed page. The operation is treated similarly to a byte load operation with respect to coherency.

If the address translation does not hit in the TLB or BAT mechanism, or if it does not have load access permission, the instruction is treated as a no-op.

If the cache is locked or disabled, or if the access is to a page that is marked as guarded, the **dcbt** instruction is treated as a no-op.

If the access is directed to a write-through or caching-inhibited page, the instruction is treated as a no-op.

The **dcbt** instruction never affects the referenced or changed bits in the hashed page table.

A successful **dcbt** instruction affects the state of the TLB and cache LRU bits as defined by the LRU algorithm.

The touch load buffer will be marked invalid if the contents of the touch buffer have been moved to the cache, if any data cache management instruction has been executed, if a **dcbz** instruction is executed that matches the address of the cache block in the touch buffer, or if another **dcbt** instruction is executed.

3.7.3 Data Cache Block Touch for Store (dcbtst) Instruction

The **dcbtst** instruction, like the data cache block touch instruction (**dcbt**), allows software to prefetch a cache block in anticipation of a store operation (read with intent to modify).

3.7.4 Data Cache Block Clear to Zero (dcbz) Instruction

If the block containing the byte addressed by the EA is in the data cache, all bytes are cleared.

If the block containing the byte addressed by the EA is not in the data cache and the corresponding page is caching-allowed, the block is established in the data cache without fetching the block from main memory, and all bytes of the block are cleared. If the contents of the cache block are from a page marked global through the WIM bits, an address-only bus transaction is run.

If the page containing the byte addressed by the EA is caching-inhibited or write-through, then the system alignment exception handler is invoked.

The **dcbz** instruction is treated as a store to the addressed byte with respect to address translation and protection.

3.7.5 Data Cache Block Store (dcbst) Instruction

If the block containing the byte addressed by the EA is in coherency-required mode, and a block containing the byte addressed by the EA is in the data cache of any processor and has been modified, the writing of it to main memory is initiated. On a PID7v-603e, if the cache block is unmodified, HID0[ABE] is set, and if the contents of the cache block are from a page marked global through the WIM bits, an address-only bus transaction is run.

The function of this instruction is independent of the write-through and caching-inhibited/caching-allowed modes of the block containing the byte addressed by the EA.

This instruction is treated as a load to the addressed byte with respect to address translation and protection.

3.7.6 Data Cache Block Flush (dcbf) Instruction

The action taken depends on the memory mode associated with the target, and on the state of the cache block. The list below describes the action taken for the various cases. The actions described are executed regardless of whether the page containing the addressed byte is in caching-inhibited or caching-allowed mode. The following actions occur in both coherency-required mode (WIM = 0bXX1) and coherency-not-required mode (WIM = 0bXX0).

The **dcbf** instruction causes the following cache activity:

- Unmodified block—Invalidates the block in the processor’s cache.
- Modified block—Copies the block to memory and invalidates data cache block.
- Absent block—Does nothing.

The 603e treats this instruction as a load to the addressed byte with respect to address translation and protection.

3.7.7 Enforce In-Order Execution of I/O Instruction (eieio)

As defined by the PowerPC architecture, the **eieio** instruction provides an ordering function for the effects of load and store instructions executed by a given processor. Executing **eieio** ensures that all memory accesses previously initiated by the given processor are completed with respect to main memory before any memory accesses subsequently initiated by the processor access main memory. The **eieio** instruction orders loads and stores to caching-inhibited memory only.

The **eieio** instruction is intended for use only in performing memory-mapped I/O operations. It enforces “strong” ordering of cache-inhibited memory accesses during I/O operations between the processor and I/O devices.

When executed by the 603e, the **eieio** instruction is treated as a no-op; caching-inhibited load and store operations (inhibited by the WIMG bits for the page) are performed in strict program order.

3.7.8 Instruction Cache Block Invalidate (icbi) Instruction

The execution of an **icbi** instruction causes all four cache sets indexed by the EA to be marked invalid. No cache hit is required, and no MMU translation is performed.

3.7.9 Instruction Synchronize (isync) Instruction

The **isync** instruction waits for all previous instructions to complete and then discards any previously fetched instructions, causing subsequent instructions to be fetched (or refetched) from memory and to execute in the context established by the previous instructions. This instruction has no effect on other processors or on their caches.

3.8 Bus Operations Caused by Cache Control Instructions

Table 3-7 provides an overview of the bus operations initiated by cache control instructions. The cache control, TLB management, and synchronization instructions supported by the 603e may affect or be affected by the operation of the bus. None of the instructions will actively broadcast through address-only transactions on the bus (except for **dcbz**), and no broadcasts by other masters are snooped by the 603e (except for kills). The operation of the instructions, however, may indirectly cause bus transactions to be performed, or their completion may be linked to the bus. Table 3-7 summarizes how these instructions may operate with respect to the bus.

Note that Table 3-7 assumes that the WIM bits are set to 001; that is, since the cache is operating in write-back mode, caching is permitted and coherency is enforced.

Table 3-7. Bus Operations Caused by Cache Control Instructions (WIM = 001)

Operation	Cache State	Next Cache State	Bus Operations	Comment
sync	Don't care	No change	None	Waits for memory queues to complete bus activity
icbi	Don't care	I	None	—
dcbi	Don't care	I	None	—
dcbf	I, E	I	None	—
dcbf	M	I	Write with kill	Block is pushed
dcbst	I, E	No change	None	—
dcbst	M	E	Write	Block is pushed
dcbz	I	M	Write with kill	—
dcbz	E, M	M	Kill block	Writes over modified data
dcbt	I	No change	Read	Fetched cache block is stored in touch load queue
dcbt	E, M	No change	None	—
dcbtst	I	No change	Read-with-intent-to-modify	Fetched cache block is stored in touch load queue
dcbtst	E,M	No change	None	—

Table 3-7 does not include noncacheable or write-through cases, nor does it completely describe the mechanisms for the operations described. For more information, see Section 3.10, "MEI State Transactions."

For detailed information on the cache control instructions, refer to Chapter 2, "Programming Model," in this book and Chapter 8, "Instruction Set," in *The Programming Environments Manual*. The 603e contains snooping logic to monitor the bus for these commands and the control logic required to keep the cache and the memory queues coherent. For additional details about the specific bus operations performed by the 603e, see Chapter 8, "System Interface Operation."

3.9 Bus Interface

The bus interface buffers bus requests from the instruction and data caches, and executes the requests per the 603e bus protocol. It includes address register queues, prioritization logic, and bus control logic. The bus interface also captures snoop addresses for snooping in the cache and in the address register queues, snoops for reservations, and holds the touch load address for the cache. All data storage for the address register buffers (load and store data buffers) are located in the cache section. The data buffers are considered temporary storage for the cache and not part of the bus interface.

The general functions and features of the bus interface are as follows:

- Seven address register buffers that include the following:
  - Instruction cache load address buffer
  - Data cache load address buffer
  - Data cache touch load address buffer (associated data block buffer located in cache)
  - Data cache castout/store address buffer (associated data line buffer located in cache)
  - Data cache snoop copyback address buffer (associated data line buffer located in cache)
  - Reservation address buffer for snoop monitoring
- Pipeline collision detection for data cache buffers
- Reservation address snooping for **lwarx**/**stwcx.** instructions
- One-level address pipelining
- Load ahead of store capability

A conceptual block diagram of the bus interface is shown in Figure 3-5. The address register queues hold transaction requests that the bus interface may issue on the bus independently of the other requests. The bus interface may have up to two transactions operating on the bus at any given time through the use of address pipelining.

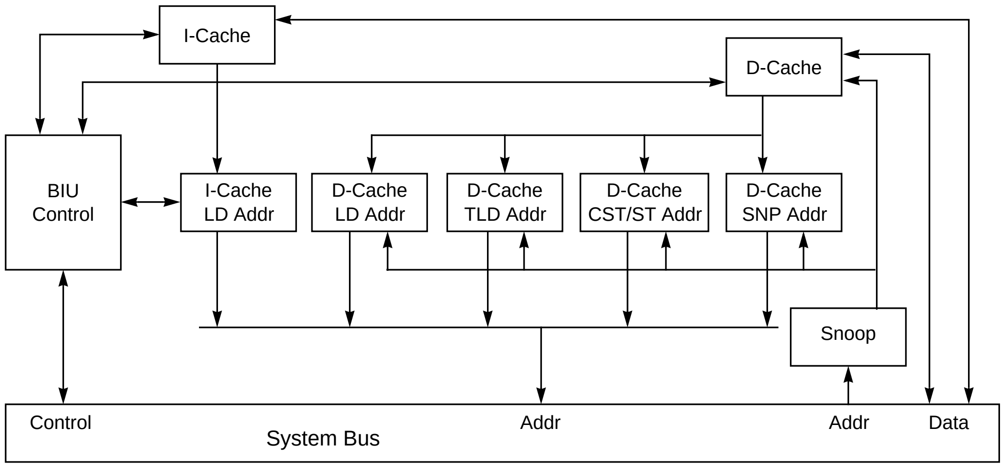


Figure 3-5. Bus Interface Address Buffers

For additional information about the 603e bus interface and the bus protocols, refer to Chapter 8, "System Interface Operation."

3.10 MEI State Transactions

Table 3-8 shows MEI state transitions for various operations. Bus operations are described in Table 3-6.

Table 3-8. MEI State Transitions

Operation	**Cache Operation**	**Bus sync**	**WIM**	**Current State**	**Next State**	**Cache Actions**	**Bus Operation**
Load (T = 0)	Read	No	x0x	I	Same	1 Cast out of modified block (as required)	Write-with-kill
						2 Pass four-beat read to memory queue	Read
Load (T = 0)	Read	No	x0x	E,M	Same	Read data from cache	—
Load (T = 0)	Read	No	x1x	I	Same	Pass single-beat read to memory queue	Read
Load (T = 0)	Read	No	x1x	E	I	CRTRY read	—
Load (T = 0)	Read	No	x1x	M	I	CRTRY read (push sector to write queue)	Write-with-kill
lwarx	Read	Acts like other reads but bus operation uses special encoding					

Table 3-8. MEI State Transitions (Continued)

Operation	Cache Operation	Bus sync	WIM	Current State	Next State	Cache Actions	Bus Operation
Store (T = 0)	Write	No	00x	I	Same	1 Cast out of modified block (if necessary)	Write-with-kill
						2 Pass RWITM to memory queue	RWITM
Store (T = 0)	Write	No	00x	E,M	M	Write data to cache	—
Store ≠ **stwcx.** (T = 0)	Write	No	10x	I	Same	Pass single-beat write to memory queue	Write-with-flush
Store ≠ **stwcx.** (T = 0)	Write	No	10x	E	Same	1 Write data to cache	—
						2 Pass single-beat write to memory queue	Write-with-flush
Store ≠ **stwcx.** (T = 0)	Write	No	10x	M	Same	1 CRTRY write	—
						2 Push block to write queue	Write-with-kill
Store (T = 0) or **stwcx.** (WIM = 10x)	Write	No	x1x	I	Same	Pass single-beat write to memory queue	Write-with-flush
Store (T = 0) or **stwcx.** (WIM = 10x)	Write	No	x1x	E	I	CRTRY write	—
Store (T = 0) or **stwcx.** (WIM = 10x)	Write	No	x1x	M	I	1 CRTRY write	—
						2 Push block to write queue	Write-with-kill
stwcx.	Conditional write	If the reserved bit is set, this operation is like other writes except the bus operation uses a special encoding.					
dcbf	Data cache block flush	No	xxx	I,E	Same	1 CRTRY **dcbf**	—
						2 Pass flush	Flush
				Same	I	3 State change only	—
dcbf	Data cache block flush	No	xxx	M	I	Push block to write queue	Write-with-kill
dcbst	Data cache block store	No	xxx	I,E	Same	1 CRTRY **dcbst**	—
						2 Pass clean	Clean
				Same	Same	3 No action	—
dcbst	Data cache block store	No	xxx	M	E	Push block to write queue	Write-with-kill
dcbz	Data cache block set to zero	No	x1x	x	x	Alignment trap	—

Table 3-8. MEI State Transitions (Continued)

Operation	Cache Operation	Bus sync	WIM	Current State	Next State	Cache Actions	Bus Operation
dcbz	Data cache block set to zero	No	10x	x	x	Alignment trap	—
dcbz	Data cache block set to zero	Yes	00x	I	Same	1 CRTRY **dcbz**	—
						2 Cast out of modified block	Write-with-kill
						3 Pass kill	Kill
				Same	M	4 Clear block	—
dcbz	Data cache block set to zero	No	00x	E,M	M	Clear block	—
dcbt	Data cache block touch	No	x1x	I	Same	Pass single-beat read to memory queue	Read
dcbt	Data cache block touch	No	x1x	E	I	CRTRY read	—
dcbt	Data cache block touch	No	x1x	M	I	1 CRTRY read	—
						2 Push block to write queue	Write-with-kill
dcbt	Data cache block touch	No	x0x	I	Same	1 Cast out of modified block (as required)	Write-with-kill
						2 Pass four-beat read to memory queue	Read
dcbt	Data cache block touch	No	x0x	E,M	Same	No action	—
Single-beat read	Reload dump 1	No	xxx	I	Same	Forward data_in	—
Four-beat read (double-word-aligned)	Reload dump	No	xxx	I	E	Write data_in to cache	—
Four-beat write (double-word-aligned)	Reload dump	No	xxx	I	M	Write data_in to cache	—
E⟶I	Snoop write or kill	No	xxx	E	I	State change only (committed)	—
M⟶I	Snoop kill	No	xxx	M	I	State change only (committed)	—
Push M⟶I	Snoop flush	No	xxx	M	I	Conditionally push	Write-with-kill
Push M⟶E	Snoop clean	No	xxx	M	E	Conditionally push	Write-with-kill

Table 3-8. MEI State Transitions (Continued)

Operation	Cache Operation	Bus sync	WIM	Current State	Next State	Cache Actions	Bus Operation
tlbie	TLB invalidate	No	xxx	x	x	1 CRTRY TLBI	—
						2 Pass TLBI	—
						3 No action	—
sync	Synchroni-zation	No	xxx	x	x	1 CRTRY **sync**	—
						2 Pass **sync**	—
						3 No action	—

Note that single-beat writes are not snooped in the write queue.

Chapter 4 Exceptions

The PowerPC exception mechanism allows the processor to change to supervisor state as a result of external signals, errors, or unusual conditions arising in the execution of instructions, and differ from the arithmetic exceptions defined by the IEEE for floating-point operations. When exceptions (referred to as interrupts in the architecture specification) occur, information about the state of the processor is saved to certain registers and the processor begins execution at an address (exception vector) predetermined for each exception. Processing of exceptions occurs in supervisor mode.

Although multiple exception conditions can map to a single exception vector, a more specific condition may be determined by examining a register associated with the exception—for example, the DSISR or the FPSCR. Additionally, certain exception conditions can be explicitly enabled or disabled by software.

The PowerPC architecture requires that exceptions be handled in program order; therefore, although a particular implementation may recognize exception conditions out of order, they are handled strictly in order with respect to the instruction stream. When an instruction-caused exception is recognized, any unexecuted instructions that appear earlier in the instruction stream, including any that have not yet entered the execute state, are required to complete before the exception is taken. Any exceptions caused by those instructions are handled first. Likewise, exceptions that are asynchronous and precise are recognized when they occur, but are not handled until the instruction currently in the completion stage successfully completes execution or generates an exception, and the completed store queue is emptied. An instruction is said to have “completed” when the results of that instruction’s execution have been committed to the registers defined by the architecture (for example, the GPRs or FPRs, rather than rename buffers). If a single instruction encounters multiple exception conditions, those exceptions are taken and handled sequentially. Likewise, exceptions that are asynchronous are recognized when they occur, but are not handled until the next instruction to complete in program order successfully completes. Throughout this chapter, the term ‘next instruction’ implies the next instruction to complete in program order.

Note that exceptions can occur while an exception handler routine is executing, and multiple exceptions can become nested. It is up to the exception handler to save the states to allow control to ultimately return to the original excepting program.

Unless a catastrophic condition causes a system reset or machine check exception, only one exception is handled at a time. If, for example, a single instruction encounters multiple exception conditions, those conditions are handled sequentially. After the exception handler handles an exception, the instruction execution continues until the next exception condition is encountered. However, in many cases there is no attempt to re-execute the instruction. This method of recognizing and handling exception conditions sequentially guarantees that exceptions are recoverable.

Exception handlers should save the information stored in SRR0 and SRR1 early to prevent the program state from being lost due to a system reset or machine check exception or to an instruction-caused exception in the exception handler, and before enabling external interrupts.

In this chapter, the following terminology is used to describe the various stages of exception processing:

Recognition	Exception recognition occurs when the condition that can cause an exception is identified by the processor.
Taken	An exception is said to be taken when control of instruction execution is passed to the exception handler; that is, the context is saved and the instruction at the appropriate vector offset is fetched and the exception handler routing is executed in supervisor mode.
Handling	Exception handling is performed by the software linked to the appropriate vector offset. Exception handling is performed at supervisor-level.

4.1 Exception Classes

The PowerPC architecture supports four types of exceptions:

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

- Synchronous, imprecise—The PowerPC architecture defines two imprecise floating-point exception modes, recoverable and nonrecoverable. Even though the PowerPC 603e provides a means to enable the imprecise modes, it implements these modes identically to the precise mode (that is, all enabled floating-point enabled exceptions are always precise on the 603e). (The EC603e microprocessor does not support floating-point operations.)
- Asynchronous, maskable—The external, system management interrupt (SMI), and decrementer exceptions are maskable asynchronous exceptions. When these exceptions occur, their handling is postponed until the next instruction, and any exceptions associated with that instruction, completes execution. If there are no instructions in the execution units, the exception is taken immediately upon determination of the correct restart address (for loading SRR0).
- Asynchronous, nonmaskable—There are two nonmaskable asynchronous exceptions: system reset and the machine check exception. These exceptions may not be recoverable, or may provide a limited degree of recoverability. All exceptions report recoverability through the MSR[RI] bit.

The 603e exception classes are shown in Table 4-1.

Table 4-1. Exception Classifications

Synchronous/Asynchronous	Precise/Imprecise	Exception Type
Asynchronous, nonmaskable	Imprecise	Machine check System reset
Asynchronous, maskable	Precise	External interrupt Decrementer System management interrupt
Synchronous	Precise	Instruction-caused exceptions

Although exceptions have other characteristics as well, such as whether they are maskable or nonmaskable, the distinctions shown in Table 4-1 define categories of exceptions that the 603e handles uniquely. Note that Table 4-1 includes no synchronous imprecise exceptions. While the PowerPC architecture supports imprecise handling of floating-point exceptions, the 603e, with the exception of the EC603e microprocessor, implements floating-point exception modes as precise exceptions. (The EC603e microprocessor does not support floating-point operations.)

Although the PowerPC architecture specifies that the recognition of the machine check exception is nonmaskable, on the 603e the stimuli that cause this exception are maskable. For example, the machine check exception is caused by the assertion of $\overline{\text{TEA}}$, $\overline{\text{APE}}$, $\overline{\text{DPE}}$, or $\overline{\text{MCP}}$. However, the $\overline{\text{MCP}}$, $\overline{\text{APE}}$, and $\overline{\text{DPE}}$ signals can be disabled by bits 0, 2, and 3 respectively in HID0. Therefore, the machine check caused by $\overline{\text{TEA}}$ is the only truly nonmaskable machine check exception.

The 603e's exceptions, and conditions that cause them, are listed in Figure 4-1. Exceptions that are specific to either the PID6-603e or PID7v-603e, or that are handled differently on the EC603e microprocessor, are indicated.

Figure 4-1. Exceptions and Conditions

Exception Type	Vector Offset (hex)	Causing Conditions
Reserved	00000	—
System reset	00100	A system reset is caused by the assertion of either $\overline{\text{SRESET}}$ or $\overline{\text{HRESET}}$.
Machine check	00200	A machine check is caused by the assertion of the $\overline{\text{TEA}}$ signal during a data bus transaction, assertion of $\overline{\text{MCP}}$, or an address or data parity error.
DSI	00300	The cause of a DSI exception can be determined by the bit settings in the DSISR, listed as follows: 1 Set if the translation of an attempted access is not found in the primary hash table entry group (HTEG), or in the rehashed secondary HTEG, or in the range of a DBAT register; otherwise cleared. 4 Set if a memory access is not permitted by the page or DBAT protection mechanism; otherwise cleared. 5 Set by an **eciwx** or **ecowx** instruction if the access is to an address that is marked as write-through, or execution of a load/store instruction that accesses a direct-store segment. 6 Set for a store operation and cleared for a load operation. 11 Set if **eciwx** or **ecowx** is used and EAR[E] is cleared.
ISI	00400	An ISI exception is caused when an instruction fetch cannot be performed for any of the following reasons: • The effective (logical) address cannot be translated. That is, there is a page fault for this portion of the translation, so an ISI exception must be taken to load the PTE (and possibly the page) into memory. • The fetch access is to a direct-store segment (indicated by SRR1[3] set). • The fetch access violates memory protection (indicated by SRR1[4] set). If the key bits (Ks and Kp) in the segment register and the PP bits in the PTE are set to prohibit read access, instructions cannot be fetched from this location.
External interrupt	00500	An external interrupt is caused when MSR[EE] = 1 and the $\overline{\text{INT}}$ signal is asserted.
Alignment	00600	An alignment exception is caused when the 603e cannot perform a memory access for any of the reasons described below: • The operand of a floating-point load or store instruction is not word-aligned. • The operand of **lmw**, **stmw**, **lwarx**, and **stwcx.** instructions are not aligned. • The operand of a single-register load or store operation is not aligned, and the 603e is in little-endian mode (PID6-603e only). • The execution of a floating-point load or store instruction to a direct-store segment. • The operand of a load, store, load multiple, store multiple, load string, or store string instruction crosses a segment boundary into a direct-store segment, or crosses a protection boundary. • Execution of a misaligned **eciwx** or **ecowx** instruction (PID7v-603e only). • The instruction is **lmw**, **stmw**, **lswi**, **lswx**, **stswi**, **stswx** and the 603e is in little-endian mode. • The operand of **dcbz** is in memory that is write-through-required or caching-inhibited.

Figure 4-1. Exceptions and Conditions (Continued)

Exception Type	Vector Offset (hex)	Causing Conditions
Program	00700	A program exception is caused by one of the following exception conditions, which correspond to bit settings in SRR1 and arise during execution of an instruction: • Floating-point enabled exception—A floating-point enabled exception condition is generated when the following condition is met: (MSR[FE0] | MSR[FE1]) & FPSCR[FEX] is 1. (Not supported by the EC603e microprocessor.) FPSCR[FEX] is set by the execution of a floating-point instruction that causes an enabled exception or by the execution of one of the “move to FPSCR” instructions that results in both an exception condition bit and its corresponding enable bit being set in the FPSCR. (Not supported by the EC603e microprocessor.) • Illegal instruction—An illegal instruction program exception is generated when execution of an instruction is attempted with an illegal opcode or illegal combination of opcode and extended opcode fields (including PowerPC instructions not implemented in the 603e), or when execution of an optional instruction not provided in the 603e is attempted (these do not include those optional instructions that are treated as no-ops). • Privileged instruction—A privileged instruction type program exception is generated when the execution of a privileged instruction is attempted and the MSR register user privilege bit, MSR[PR], is set. In the 603e, this exception is generated for **mtspr** or **mfspr** with an invalid SPR field if SPR[0] = 1 and MSR[PR] = 1. This may not be true for all PowerPC processors. • Trap—A trap type program exception is generated when any of the conditions specified in a trap instruction is met.
Floating-point unavailable	00800	A floating-point unavailable exception is caused by an attempt to execute a floating-point instruction (including floating-point load, store, and move instructions) when the floating-point available bit is disabled (MSR[FP] = 0). Note that the EC603e microprocessor takes a floating-point unavailable exception when execution of a floating-point instruction is attempted.
Decrementer	00900	The decrementer exception occurs when the most significant bit of the decrementer (DEC) register transitions from 0 to 1. Must also be enabled with the MSR[EE] bit.
Reserved	00A00–00BFF	—
System call	00C00	A system call exception occurs when a System Call (**sc**) instruction is executed.
Trace	00D00	A trace exception is taken when MSR[SE] =1 or when the currently completing instruction is a branch and MSR[BE] =1.
Reserved	00E00	The 603e does not generate an exception to this vector. Other PowerPC processors may use this vector for floating-point assist exceptions.
Reserved	00E10–00FFF	—
Instruction translation miss	01000	An instruction translation miss exception is caused when an effective address for an instruction fetch cannot be translated by the ITLB.

Figure 4-1. Exceptions and Conditions (Continued)

Exception Type	Vector Offset (hex)	Causing Conditions
Data load translation miss	01100	A data load translation miss exception is caused when an effective address for a data load operation cannot be translated by the DTLB.
Data store translation miss	01200	A data store translation miss exception is caused when an effective address for a data store operation cannot be translated by the DTLB, or where a DTLB hit occurs, and the change bit in the PTE must be set due to a data store operation.
Instruction address breakpoint	01300	An instruction address breakpoint exception occurs when the address (bits 0–29) in the IABR matches the next instruction to complete in the completion unit, and the IABR enable bit (bit 30) is set.
System management interrupt	01400	A system management interrupt is caused when MSR[EE] = 1 and the $\overline{\text{SMI}}$ input signal is asserted.
Reserved	01500–02FFF	—

Exceptions are roughly prioritized by exception class, as follows:

1. Nonmaskable, asynchronous exceptions have priority over all other exceptions—system reset and machine check exceptions (although the machine check exception condition can be disabled so the condition causes the processor to go directly into the checkstop state). These exceptions cannot be delayed, and do not wait for the completion of any precise exception handling.
2. Synchronous, precise exceptions are caused by instructions and are taken in strict program order.
3. Maskable asynchronous exceptions (external interrupt and decrementer exceptions) are delayed until higher priority exceptions are taken.

System reset and machine check exceptions may occur at any time and are not delayed even if an exception is being handled. As a result, state information for the interrupted exception may be lost; therefore, these exceptions are typically nonrecoverable.

All other exceptions have lower priority than system reset and machine check exceptions, and the exception may not be taken immediately when it is recognized.

4.1.1 Exception Priorities

The exceptions are listed in Table 4-2 in order of highest to lowest priority.

Table 4-2. Exception Priorities

Exception Category	Priority	Exception	Cause
Asynchronous	0	System reset	$\overline{\text{HRESET}}$ or power-on reset
	1	Machine check	$\overline{\text{TEA}}$, $\overline{\text{MCP}}$, $\overline{\text{APE}}$, or $\overline{\text{DPE}}$
	2	System reset	$\overline{\text{SRESET}}$
	3	System management interrupt	$\overline{\text{SMI}}$
	4	External interrupt	$\overline{\text{INT}}$
	5	Decrementer exception	Decrementer passed through 0x00000000
Instruction fetch	0	ITLB miss	Instruction TLB miss
	1	Instruction access	Instruction access exception

Table 4-2. Exception Priorities (Continued)

Exception Category	Priority	Exception	Cause
Instruction dispatch/ execution	0	IABR	Instruction address breakpoint exception
	1	Program	Program exception due to the following: • Illegal instruction • Privileged instruction • Trap
	2	System call	System call exception
	3	Floating-point unavailable	Floating-point unavailable exception due to the following: • 603e microprocessor—Floating-point unavailable exception. • EC603e microprocessor—Execution of a floating-point instruction.
	4	Program	Program exception due to a floating-point enabled exception
	5	Alignment	Alignment exception due to the following: • Floating-point not word-aligned (not applicable to the EC603e microprocessor) • **lmw**, **stmw**, **lwarx**, or **stwcx.** not word-aligned • Little-endian access is misaligned • Multiple or string access with little-endian bit set
	6	Data access	Data access exception due to a BAT page protection violation
	7	Data access	Data access exception due to the following: • **eciwx**, **ecowx**, **lwarx**, or **stwcx.** to direct-store segment (bit 5 of DSISR) • Crossing from memory segment to direct-store segment (bit 0 of DSISR) • Crossing from direct-store segment to memory segment • Any access to direct-store, SR[T] = 1 • **eciwx** or **ecowx** with EAR[E] = 0 (bit 11 of DSISR)
	8	DTLB miss	Data TLB miss exception due to: • Store miss • Load miss
	9	Alignment	Alignment exception due to a **dcbz** to a write-through or caching-inhibited page
	10	Data access	Data access exception due to TLB page protection violation
	11	DTLB miss	Data TLB miss exception due to a change bit not set on a store operation
Post-instruction execution	0	Trace	Trace exception due to the following: • MSR[SE] = 1 • MSR[BE] = 1 for branches

Exception priorities are described in detail in “Exception Priorities,” in Chapter 6, “Exceptions,” in *The Programming Environments Manual*.

4.1.2 Summary of Front-End Exception Handling

The following list of interrupt categories describes how the 603e handles exceptions up to the point of signaling the appropriate exception to occur. Note that a recoverable state is reached if the completed store queue is empty (drained, not canceled) and any instruction that is next in program order and has been signaled to complete has completed. If MSR[RI] is clear, the 603e is in a nonrecoverable state by default. Also, completion of an instruction is defined as performing all architectural register writes associated with that instruction, and then removing that instruction from the completion buffer queue.

- Asynchronous nonmaskable nonrecoverable—(System reset caused by the assertion of either $\overline{\text{HRESET}}$ or internally during power-on reset (POR)). These exceptions have highest priority and are taken immediately regardless of other pending exceptions or recoverability. A nonpredicted address is guaranteed.
- Asynchronous maskable nonrecoverable—(Machine check). A machine check exception takes priority over any other pending exception except a nonrecoverable system reset caused by the assertion of either $\overline{\text{HRESET}}$ or internally during POR. A machine check exception is taken immediately regardless of recoverability. A machine check exception can occur only if the machine check enable bit, MSR[ME], is set. If MSR[ME] is cleared, the processor goes directly into checkstop state when a machine check exception condition occurs. A nonpredicted address is guaranteed.
- Asynchronous nonmaskable recoverable—(System reset caused by the assertion of $\overline{\text{SRESET}}$). This interrupt takes priority over any other pending exceptions except nonrecoverable exceptions listed above. This exception is taken immediately when a recoverable state is reached.
- Asynchronous maskable recoverable—(System management interrupt, external interrupt, decrementer exception). Before handling this type of exception, the next instruction in program order must complete or except. If this action causes another type of exception, that exception is taken and the asynchronous maskable recoverable exception remains pending. Once an instruction can complete without causing an exception, further instruction completion is halted while the exception not taken remains pending. The exception is taken when a recoverable state is reached.
- Instruction fetch–(ITLB, ISI). When this type of exception is detected, dispatch is halted and the current instruction stream is allowed to drain. If completing any instructions in this stream causes an exception, that exception is taken and the instruction fetch exception is forgotten. Otherwise, as soon as the machine is empty and a recoverable state is reached, the instruction fetch exception is taken.

- Instruction dispatch/execution—(Program, DSI, alignment, emulation trap, system call, DTLB miss on load or store, IABR). This type of exception is determined at dispatch or execution of an instruction. The exception remains pending until all instructions in program order before the exception-causing instruction are completed. The exception is then taken without completing the exception-causing instruction. If any other exception condition is created in completing these previous instructions in the machine, that exception takes priority over the pending instruction dispatch/execution exception, which will then be forgotten.
- Post–instruction execution—(Trace). This type of exception is generated following execution and completion of an instruction while a trace mode is enabled. If executing the instruction produces conditions for another type of interrupt, that exception is taken and the post-instruction execution exception is forgotten for that instruction.

4.2 Exception Processing

When an exception is taken, the processor uses the save/restore registers, SRR0 and SRR1, to save the contents of the machine state register for user-level mode (referred to as problem mode in the architecture specification) and to identify where instruction execution should resume after the exception is handled.

When an exception occurs, SRR0 is set to point to the instruction at which instruction processing should resume when the exception handler returns control to the interrupted process. All instructions in the program flow preceding this one will have completed and no subsequent instruction will have completed. This may be the address of the instruction that caused the exception or the next one (as in the case of a system call exception). The instruction addressed can be determined from the exception type and status bits. This address is used to resume instruction processing in the interrupted process, typically when an **rfi** instruction is executed. The SRR0 register is shown in Figure 4-2.

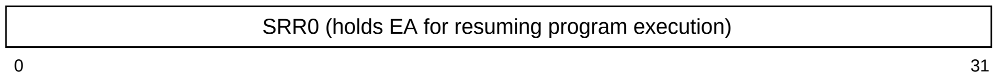


Figure 4-2. Machine Status Save/Restore Register 0

The save/restore register 1 (SRR1) is used to save machine status (the contents of the MSR) on exceptions and to restore those values when **rfi** is executed. SRR1 is shown in Figure 4-3.

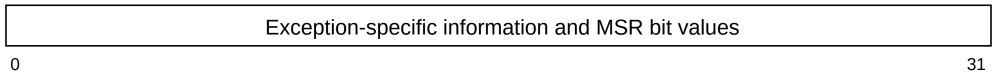


Figure 4-3. Machine Status Save/Restore Register 1

Typically, when an exception occurs, bits 0–15 of SRR1 are loaded with exception-specific information and bits 16–31 of MSR are placed into the corresponding bit positions of SRR1. The 603e loads SRR1 with specific bits for handling machine check exceptions, as shown in Table 4-3.

Table 4-3. SRR1 Bit Settings for Machine Check Exceptions

Bits	Name	Description
0	MSR[0]	Copy of MSR bit 0
1–4	—	Reserved
5–9	MSR[5–9]	Copy of MSR bits 5–9
10–11	—	Reserved
12	MCP	Machine check
13	TEA	TEA error
14	DPE	Data parity error
15	APE	Address parity error
16–31	MSR[16–31]	Copy of MSR bits16–31

The 603e loads SRR1 with specific bits for handling the three TLB miss exceptions, as shown in Table 4-4.

Table 4-4. SRR1 Bit Settings for Software Table Search Operations

Bits	Name	Description
0–3	CRF0	Copy of condition register field 0 (CR0)
4	—	Reserved
5–9	MSR[5–9]	Copy of MSR bits 5–9
10–11	—	Reserved
12	KEY	TLB miss protection key
13	I/D	Instruction/data TLB miss 0 DTLB miss 1 ITLB miss
14	WAY	Bit 14 indicates which TLB associativity set should be replaced 0 Set 0 1 Set 1
15	S/L	Store/load protection instruction 0 Load miss 1 Store miss
16–31	MSR[16–31]	Copy of MSR bits 16–31

Note that in some implementations, every instruction fetch when MSR[IR] = 1 and every instruction execution requiring address translation when MSR[DR] = 1 may modify SRR1.

The MSR is shown in Figure 4-4. When an exception occurs, MSR bits, as described in Table 4-5, are altered as determined by the exception.

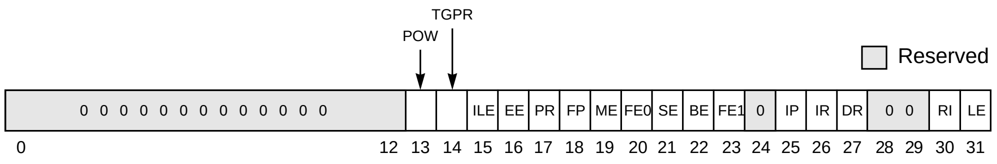


Figure 4-4. Machine State Register (MSR)

Table 4-5 shows the bit definitions for the MSR. Full function reserved bits are saved in SRR1 when an exception occurs; partial function reserved bits are not saved.

Table 4-5. MSR Bit Settings

Bit(s)	Name	Description
0	—	Reserved. Full function.
1–4	—	Reserved. Partial function.
5–9	—	Reserved. Full function.
10–12	—	Reserved. Partial function.
13	POW	Power management enable (603e-specific) 0 Disables programmable power modes (normal operation mode). 1 Enables programmable power modes (nap, doze, or sleep mode). This bit controls the programmable power modes only; it has no effect on dynamic power management (DPM). MSR[POW] may be altered with an **mtmsr** instruction only. Also, when altering the POW bit, software may alter only this bit in the MSR and no others. The **mtmsr** instruction must be followed by a context-synchronizing instruction. See Chapter 9, “Power Management,” for more information.
14	TGPR	Temporary GPR remapping (603e-specific) 0 Normal operation 1 TGPR mode. GPR0–GPR3 are remapped to TGPR0–TGPR3 for use by TLB miss routines. The contents of GPR0–GPR3 remain unchanged while MSR[TGPR] = 1. Attempts to use GPR4–GPR31 with MSR[TGPR] = 1 yield undefined results. Temporarily replacesTGPR0–TGPR3 with GPR0–GPR3 for use by TLB miss routines. When this bit is set, all instruction accesses to GPR0–GPR3 are mapped to TGPR0–TGPR3, respectively. The TGPR bit is set when either an instruction TLB miss, data read miss, or data write miss exception is taken. The TGPR bit is cleared by an **rfi** instruction.
15	ILE	Exception little-endian mode. When an exception occurs, this bit is copied into MSR[LE] to select the endian mode for the context established by the exception.
16	EE	External interrupt enable 0 The processor ignores external interrupts, system management interrupts, and decrementer interrupts. 1 The processor is enabled to take an external interrupt, system management interrupt, or decrementer interrupt.

Table 4-5. MSR Bit Settings (Continued)

Bit(s)	Name	Description
17	PR	Privilege level 0 The processor can execute both user- and supervisor-level instructions. 1 The processor can only execute user-level instructions.
18	FP	Floating-point available 0 The processor prevents dispatch of floating-point instructions, including floating-point loads, stores, and moves, default state for the EC603e microprocessor. 1 The processor can execute floating-point instructions, and can take floating-point enabled exception type program exceptions.
19	ME	Machine check enable 0 Machine check exceptions are disabled. 1 Machine check exceptions are enabled.
20	FE0	Floating-point exception mode 0 (see Table 4-6) (Not supported on the EC603e microprocessor)
21	SE	Single-step trace enable 0 The processor executes instructions normally. 1 The processor generates a trace exception upon the successful completion of the next instruction.
22	BE	Branch trace enable 0 The processor executes branch instructions normally. 1 The processor generates a trace exception upon the successful completion of a branch instruction.
23	FE1	Floating-point exception mode 1 (see Table 4-6) (Not supported on the EC603e microprocessor)
24	—	Reserved. Full function.
25	IP	Exception prefix. The setting of this bit specifies whether an exception vector offset is prepended with Fs or 0s. In the following description, *nnnnn* is the offset of the exception. See Figure 4-1. 0 Exceptions are vectored to the physical address 0x000*n_nnnn*. 1 Exceptions are vectored to the physical address 0xFFF*n_nnnn*.
26	IR	Instruction address translation 0 Instruction address translation is disabled. 1 Instruction address translation is enabled. For more information see Chapter 5, “Memory Management.”
27	DR	Data address translation 0 Data address translation is disabled. 1 Data address translation is enabled. For more information see Chapter 5, “Memory Management.”
28–29	—	Reserved. Full function.
30	RI	Recoverable exception (for system reset and machine check exceptions) 0 Exception is not recoverable. 1 Exception is recoverable.
31	LE	Little-endian mode enable 0 The processor runs in big-endian mode. 1 The processor runs in little-endian mode.

The IEEE floating-point exception mode bits (FE0 and FE1) together define whether floating-point exceptions are handled precisely, imprecisely, or whether they are taken at all. (Note that FE0 and FE1 are not supported on the EC603e microprocessor.) The possible settings and default conditions for the 603e are shown in Table 4-6. For further details, see Chapter 6, “Exceptions,” of *The Programming Environments Manual*.

Table 4-6. IEEE Floating-Point Exception Mode Bits

FE0	FE1	Mode
0	0	Floating-point exceptions disabled
0	1	Floating-point imprecise nonrecoverable*
1	0	Floating-point imprecise recoverable*
1	1	Floating-point precise mode

* Not implemented in the 603e

MSR bits are guaranteed to be written to SRR1 when the first instruction of the exception handler is encountered.

4.2.1 Enabling and Disabling Exceptions

When a condition exists that may cause an exception to be generated, it must be determined whether the exception is enabled for that condition.

- IEEE floating-point enabled exceptions (a type of program exception) are ignored when both MSR[FE0] and MSR[FE1] are cleared. If either of these bits are set, all IEEE enabled floating-point exceptions are taken and cause a program exception. (Not supported on the EC603e microprocessor.)
- Asynchronous, maskable exceptions (that is, the external, system management, and decrementer interrupts) are enabled by setting the MSR[EE] bit. When MSR[EE] = 0, recognition of these exception conditions is delayed. MSR[EE] is cleared automatically when an exception is taken, to delay recognition of conditions causing those exceptions.
- A machine check exception can occur only if the machine check enable bit, MSR[ME], is set. If MSR[ME] is cleared, the processor goes directly into checkstop state when a machine check exception condition occurs. Individual machine check exceptions can be enabled and disabled through bits in the HID0 register, which is described in Table 2-2.
- System reset exceptions cannot be masked.

4.2.2 Steps for Exception Processing

After it is determined that the exception can be taken (by confirming that any instruction-caused exceptions occurring earlier in the instruction stream have been handled, and by confirming that the exception is enabled for the exception condition), the processor does the following:

1. The machine status save/restore register 0 (SRR0) is loaded with an instruction address that depends on the type of exception. See the individual exception description for details about how this register is used for specific exceptions.
2. Bits 1–4 and 10–15 of SRR1 are loaded with information specific to the exception type.
3. Bits 5–9 and 16–31 of SRR1 are loaded with a copy of the corresponding bits of the MSR.
4. The MSR is set as described in Table 4-5. The new values take effect beginning with the fetching of the first instruction of the exception-handler routine located at the exception vector address.

   Note that MSR[IR] and MSR[DR] are cleared for all exception types; therefore, address translation is disabled for both instruction fetches and data accesses beginning with the first instruction of the exception-handler routine.
5. Instruction fetch and execution resumes, using the new MSR value, at a location specific to the exception type. The location is determined by adding the exception's vector (see Figure 4-1) to the base address determined by MSR[IP]. If IP is cleared, exceptions are vectored to the physical address 0x000*n_nnnn*. If IP is set, exceptions are vectored to the physical address 0xFFF*n_nnnn*. For a machine check exception that occurs when MSR[ME] = 0 (machine check exceptions are disabled), the processor enters the checkstop state (the machine stops executing instructions). See Section 4.5.2, “Machine Check Exception (0x00200).”

4.2.3 Setting MSR[RI]

The operating system should handle MSR[RI] as follows:

- In the machine check and system reset exceptions—If SRR1[RI] is cleared, the exception is not recoverable. If it is set, the exception is recoverable with respect to the processor.
- In each exception handler—When enough state information has been saved that a machine check or system reset exception can reconstruct the previous state, set MSR[RI].
- In each exception handler—Clear MSR[RI], set the SRR0 and SRR1 registers appropriately, and then execute **rfi**.
- Note that the RI bit being set indicates that, with respect to the processor, enough processor state data is valid for the processor to continue, but it does not guarantee that the interrupted process can resume.

4.2.4 Returning from an Exception Handler

The Return from Interrupt (**rfi**) instruction performs context synchronization by allowing previously issued instructions to complete before returning to the interrupted process. In general, execution of the **rfi** instruction ensures the following:

- All previous instructions have completed to a point where they can no longer cause an exception. If a previous instruction causes a direct-store interface error exception, the results must be determined before this instruction is executed.
- Previous instructions complete execution in the context (privilege, protection, and address translation) under which they were issued.
- The **rfi** instruction copies SRR1 bits back into the MSR.
- The instructions following this instruction execute in the context established by this instruction.

For a complete description of context synchronization, refer to Chapter 6, “Exceptions,” of *The Programming Environments Manual.*

4.3 Process Switching

The operating system should execute one of the following when processes are switched:

- The **sync** instruction, which orders the effects of instruction execution. All instructions previously initiated appear to have completed before the **sync** instruction completes, and no subsequent instructions appear to be initiated until the **sync** instruction completes. For an example showing use of the **sync** instruction, see Chapter 2, “PowerPC Register Set,” of *The Programming Environments Manual.*
- The **isync** instruction, which waits for all previous instructions to complete and then discards any fetched instructions, causing subsequent instructions to be fetched (or refetched) from memory and to execute in the context (privilege, translation, protection, etc.) established by the previous instructions.
- The **stwcx.** instruction, to clear any outstanding reservations, which ensures that an **lwarx** instruction in the old process is not paired with an **stwcx.** instruction in the new process.

The operating system should set the MSR[RI] bit as described in Section 4.2.3, “Setting MSR[RI].”

4.4 Exception Latencies

Latencies for taking various exceptions depend on the state of the machine when the exception conditions occur. This latency may be as short as one cycle, in which case an exception is signaled in the cycle following the appearance of the exception condition. The latencies are as follows:

- Hard reset and machine check—In most cases, a hard reset or machine check exception will have a single-cycle latency. A two-to-three-cycle delay may occur only when a predicted instruction is next to complete, and the branch guess that forced this instruction to be predicted was resolved to be incorrect.
- Soft reset—The latency of a soft reset exception is affected by recoverability. The time to reach a recoverable state may depend on the time needed to complete or except an instruction at the point of completion, the time needed to drain the completed store queue, and the time waiting for a correct empty state so that a valid MSR[IP] may be saved. For lower-priority externally-generated interrupts, a delay may be incurred waiting for another interrupt, generated while reaching a recoverable state, to be serviced.

Further delays are possible for other types of exceptions depending on the number and type of instructions that must be completed before those exceptions may be serviced. See Section 4.1.2, “Summary of Front-End Exception Handling,” to determine possible maximum latencies for different exceptions.

4.5 Exception Definitions

Table 4-7 shows all the types of exceptions that can occur with the 603e and the MSR bit settings when the processor transitions to supervisor mode. The state of these bits prior to the exception is typically stored in SRR1.

Table 4-7. MSR Setting Due to Exception

Exception Type	MSR Bit															
	POW	TGPR	ILE	EE	PR	FP[1]	ME	FE0[2]	SE	BE	FE1[2]	IP	IR	DR	RI	LE
System reset	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	ILE
Machine check	0	0	—	0	0	0	0	0	0	0	0	—	0	0	0	ILE
DSI	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	ILE
ISI	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	ILE
External	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	ILE
Alignment	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	ILE
Program	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	ILE
Floating-point unavailable[3]	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	ILE

Table 4-7. MSR Setting Due to Exception (Continued)

Exception Type	MSR Bit															
	POW	TGPR	ILE	EE	PR	FP[1]	ME	FE0[2]	SE	BE	FE1[2]	IP	IR	DR	RI	LE
Decrementer	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	ILE
System call	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	ILE
Trace exception	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	ILE
ITLB miss	0	1	—	0	0	0	—	0	0	0	0	—	0	0	0	ILE
DTLB miss on load	0	1	—	0	0	0	—	0	0	0	0	—	0	0	0	ILE
DTLB miss on store	0	1	—	0	0	0	—	0	0	0	0	—	0	0	0	ILE
Instruction address breakpoint	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	ILE
System management interrupt	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	ILE

0	Bit is cleared
1	Bit is set
ILE	Bit is copied from the ILE bit in the MSR.
—	Bit is not altered

Reserved bits are read as if written as 0.

Notes:

1. The floating-point available bit is always set to 0 on the EC603e microprocessor.
2. FE0 and FE1 are not supported on the EC603e microprocessor.
3. On the EC603e microprocessor, the floating-point unavailable exception is caused by the execution of a floating-point instruction.

4.5.1 Reset Exceptions (0x00100)

The system reset exception is a nonmaskable, asynchronous exception signaled to the 603e either through the assertion of the reset signals ($\overline{\text{SRESET}}$ or $\overline{\text{HRESET}}$) or internally during the power-on reset (POR) process. The assertion of the soft reset signal, $\overline{\text{SRESET}}$, as described in Section 7.2.9.6.2, “Soft Reset (SRESET)—Input” causes the soft reset exception to be taken and the physical base address of the handler is determined by the MSR[IP] bit. The assertion of the hard reset signal, $\overline{\text{HRESET}}$, as described in Section 7.2.9.6.1, “Hard Reset (HRESET)—Input” causes the hard reset exception to be taken and the physical address of the handler is always 0xFFF0_0100.

4.5.1.1 Hard Reset and Power-On Reset

As described in 4.1.2, “Summary of Front-End Exception Handling,” the hard reset exception is a nonrecoverable, nonmaskable asynchronous exception (maskable interrupt). When $\overline{\text{HRESET}}$ is asserted or at power-on reset (POR), the 603e immediately branches to 0xFFF0_0100 without attempting to reach a recoverable state. A hard reset has the highest priority of any exception. It is always nonrecoverable. Table 4-8 shows the state of the machine just before it fetches the first instruction of the system reset handler after a hard reset.

The $\overline{\text{HRESET}}$ signal can be asserted for the following reasons:

- System power-on reset
- System reset from a panel switch
- An action required by the ESP utility

For information on the $\overline{\text{HRESET}}$ signal, see Section 7.2.9.6.1, “Hard Reset (HRESET)—Input.”

Table 4-8. Settings Caused by Hard Reset

Register	Setting	Register	Setting
GPRs	Unknown	PVR	0003000*n*
FPRs*	Unknown	HID0	00000000
FPSCR*	00000000	HID1	00000000
CR	All 0s	DMISS and IMISS	All 0s
SRs	Unknown	DCMP and ICMP	All 0s
MSR	00000040	RPA	All 0s
XER	00000000	IABR	All 0s
TBU	00000000	DSISR	00000000
TBL	00000000	DAR	00000000
LR	00000000	DEC	FFFFFFFF
CTR	00000000	HASH1	00000000
SDR1	00000000	HASH2	00000000
SRR0	00000000	TLBs	Unknown
SRR1	00000000	Cache	All cache blocks invalidated
SPRGs	00000000	BATs	Unknown
Tag directory	All 0s. (However, LRU bits are initialized so each side of the cache has a unique LRU value.)		

Note: Not supported on the EC603e microprocessor.

The following is also true after a hard reset operation:

- External checkstops are enabled.
- The on-chip test interface has given control of the I/Os to the rest of the chip for functional use.
- Since the reset exception has data and instruction translation disabled (MSR[DR] and MSR[IR] both cleared), the chip operates in real addressing mode as described in Section 5.2, "Real Addressing Mode."

4.5.1.2 Soft Reset

As described in Section 4.1.2, "Summary of Front-End Exception Handling," the soft reset exception is a type of system reset exception that is recoverable, nonmaskable, and asynchronous. 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.

Unlike a hard reset, the latches are not initialized and the instruction cache is disabled. The $\overline{\text{SRESET}}$ signal must be asserted for at least two bus clock cycles. After the $\overline{\text{SRESET}}$ signal is negated, the 603e vectors to the system reset routine at 0x0000_0100 if MSR[IP] is cleared or 0xFFF0_0100 if MSR[IP] is set. A soft reset is recoverable provided that attaining the recoverable state does not cause a machine check exception. This interrupt case is third in priority, following hard reset and machine check.

When a soft reset occurs, registers are set as shown in Table 4-9.

Table 4-9. Soft Reset Exception—Register Settings

Register	Setting Description
SRR0	Set to the effective address of the instruction that the processor would have attempted to complete next if no exception conditions were present.
SRR1	0–15 Cleared 16–31 Loaded from bits 16–31 of the MSR. Note that if the processor state is corrupted to the extent that execution cannot be reliably restarted, SRR1[30] is cleared.
MSR	POW 0 EE 0 FE0[2] 0 IR 0 TGPR 0 PR 0 SE 0 DR 0 ILE — FP[1] 0 BE 0 RI 0 IP — ME — FE1[2] 0 LE Set to value of ILE

Notes:

1. The floating-point available bit is always set to 0 on the EC603e microprocessor.
2. FE0 and FE1 are not supported on the EC603e microprocessor.

4.5.2 Machine Check Exception (0x00200)

The 603e conditionally initiates a machine check exception after detecting the assertion of the $\overline{\text{TEA}}$ or $\overline{\text{MCP}}$ signals on the 603e 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.

Note that the 603e 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 canceled 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 the MSR[ME] bit 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.

Machine check exceptions are only enabled when MSR[ME] = 1; this is described in Section 4.5.2.1, “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 4.5.2.2, “Checkstop State (MSR[ME] = 0).”

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

When a machine check exception is taken, registers are updated as shown in Table 4-10.

Table 4-10. Machine Check Exception—Register Settings

Register	Setting Description
SRR0	Set to the address of the next instruction that would have been completed in the interrupted instruction stream. Neither this instruction nor any others beyond it will have been completed. All preceding instructions will have been completed.
SRR1	0–11 Cleared 12 $\overline{\text{MCP}}$—Machine check signal caused exception 13 $\overline{\text{TEA}}$—Transfer error acknowledge signal caused exception 14 $\overline{\text{DPE}}$—Data parity error signal caused exception 15 $\overline{\text{APE}}$—Address parity error signal caused exception 16–31 Loaded from MSR[16–31].
MSR	POW 0 · EE 0 · FE0[2] 0 · IR 0 TGPR 0 · PR 0 · SE 0 · DR 0 ILE — · FP[1] 0 · BE 0 · RI 0 IP — · ME — · FE1[2] 0 · LE Set to value of ILE 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 $\overline{\text{TEA}}$ assertion. Otherwise, subsequent $\overline{\text{TEA}}$ assertions cause the processor to automatically enter the checkstop state.

Notes:

1. The floating-point available bit is always cleared to 0 on the EC603e microprocessor.
2. FE0 and FE1 are not supported on the EC603e microprocessor.

When a machine check exception is taken, instruction execution for the handler begins at offset 0x00200 from the physical base address indicated by MSR[IP].

In order to return to the main program, the exception handler should do the following:

1. SRR0 and SRR1 should be given the values to be used by the **rfi** instruction.
2. Execute **rfi**.

4.5.2.2 Checkstop State (MSR[ME] = 0)

When the 603e enters the 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.
- An extended transfer protocol error occurs.

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 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.

4.5.3 DSI Exception (0x00300)

A DSI exception occurs when no higher priority exception exists and a data memory access cannot be performed. The condition that caused the DSI exception can be determined by reading the DSISR register, a supervisor-level SPR (SPR18) that can be read by using the **mfspr** instruction. Bit settings are provided in Table 4-11. Table 4-11 also indicates which memory element is saved to the DAR. DSI exceptions can occur for any of the following reasons:

- The instruction is not supported for the type of memory addressed.
- Any access to a direct-store segment (SR[T] = 1).
- The access violates memory protection. Access is not permitted by the key (Ks and Kp) and PP bits, which are set in the segment register and PTE for page protection and in the BATs for block protection.

Note that the OEA specifies an additional case that may cause a DSI exception—when an effective address for a load, store, or cache operation cannot be translated by the TLBs. On the 603e, this condition causes a TLB miss exception instead.

These scenarios are common among all PowerPC processors. The following additional scenarios can cause a DSI exception in the 603e:

- A bus error indicates crossing from a direct-store segment to a memory segment.
- The execution of any load/store instruction to a direct-store segment, SR[T] = 1.
- A data access crosses from a memory segment (SR[T] = 0) into a direct-store segment (SR[T] = 1).

DSI exceptions can be generated by load/store instructions, and the cache control instructions (**dcbi**, **dcbz**, **dcbst**, and **dcbf**).

The 603e supports the crossing of page boundaries. However, if the second page has a translation error or protection violation associated with it, the 603e will take the DSI exception in the middle of the instruction. In this case, the data address register (DAR) always points to a byte address in the first word of the offending page.

If an **stwcx.** instruction has an effective address for which a normal store operation would cause a DSI exception, the 603e will take the DSI exception without checking for the reservation.

If the XER indicates that the byte count for an **lswi** or **stswi** instruction is zero, a DSI exception does not occur, regardless of the effective address.

The condition that caused the exception is defined in the DSISR. These conditions also use the data address register (DAR) as shown in Table 4-11.

Table 4-11. DSI Exception—Register Settings

Register	Setting Description
SRR0	Set to the effective address of the instruction that caused the exception.
SRR1	0–15 Cleared 16–31 Loaded with bits 16–31 of the MSR
MSR	POW 0 EE 0 FE0[2] 0 IR 0 TGPR 0 PR 0 SE 0 DR 0 ILE — FP[1] 0 BE 0 RI 0 IP — ME — FE1[2] 0 LE Set to value of ILE
DSISR	0 Set if a load or store instruction results in a direct-store error exception. 1 Set by the data TLB miss exception handler if the translation of an attempted access is not found in the primary hash table entry group (HTEG), or in the rehashed secondary HTEG, or in the range of a DBAT register; otherwise cleared. 2–3 Cleared 4 Set if a memory access is not permitted by the page or BAT protection mechanism; otherwise cleared. 5 Set if the **lwarx** or **stwcx.** instruction is attempted to direct-store space. 6 Set for a store operation and cleared for a load operation. 7–31 Cleared
DAR	Set to the effective address of a memory element as described in the following list: • A byte in the first word accessed in the page that caused the DSI exception, for a byte, half word, or word memory access. • A byte in the first word accessed in the BAT area that caused the DSI exception for a byte, half word, or word access to a BAT area. • A byte in the block that caused the exception for **icbi**, **dcbz**, **dcbst**, **dcbf**, or **dcbi** instructions. • Any EA in the memory range addressed (for direct-store exceptions).

Notes:

1. The floating-point available bit is always cleared to 0 on the EC603e microprocessor.
2. FE0 and FE1 are not supported on the EC603e microprocessor.

When a DSI exception is taken, instruction execution for the handler begins at offset 0x00300 from the physical base address indicated by MSR[IP].

The architecture permits certain instructions to be partially executed when they cause a DSI exception. These are as follows:

- Load multiple or load string instructions—Some registers in the range of registers to be loaded may have been loaded.
- Store multiple or store string instructions—Some bytes of memory in the range addressed may have been updated.

In these cases, the number of registers and amount of memory altered are instruction- and boundary-dependent. However, memory protection is not violated. Furthermore, if some of the data accessed is in direct-store space (SR[T] = 1) and the instruction is not supported for direct-store accesses, the locations in direct-store space are not accessed.

For update forms, the update register (**r**A) is not altered.

4.5.4 ISI Exception (0x00400)

The ISI exception is implemented as it is defined by the PowerPC architecture. An ISI exception occurs when no higher priority exception exists and an attempt to fetch the next instruction fails for any of the following reasons:

- If an instruction TLB miss fails to find the desired PTE, then a page fault is synthesized. The ITLB miss handler branches to the ISI exception handler to retrieve the translation from a storage device.
- An attempt is made to fetch an instruction from a direct-store segment while instruction translation is enabled (MSR[IR] = 1).
- An attempt is made to fetch an instruction from no-execute memory.
- An attempt is made to fetch an instruction from guarded memory when MSR[IR] = 1.
- The fetch access violates memory protection.

Register settings for this exception are described in Chapter 6, “Exceptions,” in *The Programming Environments Manual.*

When an ISI exception is taken, instruction execution for the handler begins at offset 0x00400 from the physical base address indicated by MSR[IP].

4.5.5 External Interrupt (0x00500)

An external interrupt is signaled to the 603e by the assertion of the $\overline{\text{INT}}$ signal as described in Section 7.2.9.1, “Interrupt (INT)—Input.” The interrupt may not be recognized if a higher priority exception occurs simultaneously or if the MSR[EE] bit is cleared when $\overline{\text{INT}}$ is asserted.

After the $\overline{\text{INT}}$ is detected (and provided that MSR[EE] is set), the 603e generates a recoverable halt to instruction completion. The 603e requires the next instruction in program order to complete or except, block completion of any following instructions, and allow the completed store queue to drain. If any other exceptions are encountered in this process, they are taken first and the external interrupt is delayed until a recoverable halt is achieved. At this time the 603e saves the state information and takes the external interrupt as defined in the PowerPC architecture.

The register settings for the external interrupt are shown in Table 4-12.

Table 4-12. External Interrupt—Register Settings

Register	Setting
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	0–15 Cleared 16–31 Loaded from bits 16–31 of the MSR
MSR	POW 0 EE 0 FE0[2] 0 IR 0 TGPR 0 PR 0 SE 0 DR 0 ILE — FP[1] 0 BE 0 RI 0 IP — ME — FE1[2] 0 LE Set to value of ILE

Notes:

1. The floating-point available bit is always cleared to 0 on the EC603e microprocessor.
2. FE0 and FE1 are not supported on the EC603e microprocessor.

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

The 603e only recognizes the interrupt condition ($\overline{\text{INT}}$ asserted) if the MSR[EE] bit is set; it ignores the interrupt condition if the MSR[EE] bit is cleared. To guarantee that the external interrupt is taken, the $\overline{\text{INT}}$ signal must be held active until the 603e takes the interrupt. If the $\overline{\text{INT}}$ signal is negated before the interrupt is taken, the 603e is not guaranteed to take an external interrupt. The interrupt handler must send a command to the device that asserted $\overline{\text{INT}}$, acknowledging the interrupt and instructing the device to negate $\overline{\text{INT}}$.

4.5.6 Alignment Exception (0x00600)

This section describes conditions that can cause alignment exceptions in the 603e. Similar to DSI exceptions, alignment exceptions use the SRR0 and SRR1 to save the machine state and the DSISR to determine the source of the exception. The 603e will initiate an alignment exception when it detects any of the following conditions:

- The operand of a floating-point load or store operation is not word-aligned. (Not supported on the EC603e microprocessor.)
- The operand of an **lmw**, **stmw**, **lwarx**, or **stwcx.** instruction is not word-aligned.
- A little-endian access (MSR[LE] = 1) is misaligned.
- A multiple or string access is attempted with the MSR[LE] bit set.
- The operand of a **dcbz** instruction is in a page that is write-through or caching-inhibited.

The register settings for alignment exceptions are shown in Table 4-12.

Table 4-13. Alignment Interrupt—Register Settings

Register	Setting
SRR0	Set to the effective address of the instruction that caused the exception.
SRR1	0–15 Cleared 16–31 Loaded from bits 16–31 of the MSR
MSR	POW 0 · EE 0 · FE0[2] 0 · IR 0 TGPR 0 · PR 0 · SE 0 · DR 0 ILE — · FP[1] 0 · BE 0 · RI 0 IP — · ME — · FE1[2] 0 · LE Set to value of ILE
DSISR	0–11 Cleared 12–13 Cleared. (Note that these bits can be set by several 64-bit PowerPC instructions that are not supported in the 603e.) 14 Cleared 15–16 For instructions that use register indirect with index addressing—set to bits 29–30 of the instruction. For instructions that use register indirect with immediate index addressing—cleared. 17 For instructions that use register indirect with index addressing—set to bit 25 of the instruction. For instructions that use register indirect with immediate index addressing— Set to bit 5 of the instruction 18–21 For instructions that use register indirect with index addressing—set to bits 21–24 of the instruction. For instructions that use register indirect with immediate index addressing—set to bits 1–4 of the instruction. 22–26 Set to bits 6–10 (identifying either the source or destination) of the instruction. Undefined for **dcbz**. 27–31 Set to bits 11–15 of the instruction (**r**A) Set to either bits 11–15 of the instruction or to any register number not in the range of registers loaded by a valid form instruction, for **lmw**, **lswi**, and **lswx** instructions. Otherwise undefined.
DAR	Set to the EA of the data access as computed by the instruction causing the alignment exception.

Notes:

1. The floating-point available bit is always cleared to 0 on the EC603e microprocessor.
2. FE0 and FE1 are not supported on the EC603e microprocessor.

The architecture does not support the use of an unaligned EA by **lwarx** or **stwcx.** instructions. If one of these instructions specifies an unaligned EA, the exception handler should not emulate the instruction, but should treat the occurrence as a programming error.

4.5.6.1 Integer Alignment Exceptions

The 603e is optimized for load and store operations that are aligned on natural boundaries. Operations that are not naturally aligned may suffer performance degradation, depending on the type of operation, the boundaries crossed, and the mode that the processor is in during execution. More specifically, these operations may either cause an alignment exception or they may cause the processor to break the memory access into multiple, smaller accesses with respect to the cache and the memory subsystem.

The 603e can initiate an alignment exception for the access shown in Table 4-14. In this case, the appropriate range check is performed before the instruction begins execution. As a result, if an alignment exception is taken, it is guaranteed that no portion of the instruction has been executed.

Table 4-14. Access Types

MSR[DR]	SR[T]	Access Type
1	0	Page-address translation access

4.5.6.1.1 Page Address Translation Access

A page-address translation access occurs when MSR[DR] is set, SR[T] is cleared and there is not a match in the BAT. Note the following points:

- The following is true for all loads and stores except strings/multiples:
  - — Byte operands never cause an alignment exception.
  - — Half-word operands can cause an alignment exception if the EA ends in 0xFFF.
  - — Word operands can cause an alignment exception if the EA ends in 0xFFD–FFF.
  - — Double-word operands cause an alignment exception if the EA ends in 0xFF9–FFF.
- The **dcbz** instruction causes an alignment exception if the access is to a page or block with the W (write-through) or I (cache-inhibit) bit set in the TLB or BAT, respectively.

A misaligned memory access that does not cause an alignment exception will not perform as well as an aligned access of the same type. The resulting performance degradation due to misaligned accesses depends on how well each individual access behaves with respect to the memory hierarchy. At a minimum, additional cache access cycles are required that can delay other processor resources from using the cache. More dramatically, for an access to a noncacheable page, each discrete access involves individual processor bus operations that reduce the effective bandwidth of that bus.

Finally, note that when the 603e is in page address translation mode, there is no special handling for accesses that fall into BAT regions.

4.5.6.2 Floating-Point Alignment Exceptions

The 603e implements the alignment exception as it is defined in the PowerPC architecture. For information on bit settings and how exception conditions are detected, refer to *The Programming Environments Manual*.

Note that the PowerPC architecture allows individual processors to determine whether an exception is required to handle various alignment conditions. The 603e initiates an alignment exception when it detects any of the following conditions:

- The operand of a floating-point load or store operation is not word-aligned.
- The operand of a **dcbz** instruction is in a page that is write-through or caching-inhibited for a virtual mode access.
- The operand of an **lmw**, **stmw**, **lwarx**, or **stwcx**. instruction is not word-aligned. Note that unlike other alignment exceptions, which store the address as computed by the instruction in the DAR, alignment exceptions for load or store multiple instructions store that address value + 4 into the DAR.
- A little-endian access is misaligned.
- A multiple access is attempted while the little-endian, MSR[LE], bit is set.

4.5.7 Program Exception (0x00700)

The 603e implements the program exception as it is defined by the PowerPC architecture (OEA). A program exception occurs when no higher priority exception exists and one or more of the exception conditions defined in the OEA occur.

When a program exception is taken, instruction execution for the handler begins at offset 0x00700 from the physical base address indicated by MSR[IP]. The exception conditions are as follows:

- Floating-point enabled exception—These exceptions correspond to IEEE-defined exception conditions, such as overflows, and divide by zeros that may occur during the execution of a floating-point arithmetic instruction. As a group, these exceptions are enabled by the FE0 and FE1 bits in the in the MSR. Individual conditions are enabled by specific bits in the FPSCR. For general information about this exception, see *The Programming Environments Manual*. For more information about how these exceptions are implemented in the 603e, see Section 4.5.7.1, “IEEE Floating-Point Exception Program Exceptions.”

  Note: The floating-point enabled exception is not supported on the EC603e microprocessor.
- Illegal instruction—An illegal instruction program exception is generated when execution of an instruction is attempted with an illegal opcode or illegal combination of opcode and extended opcode fields (including PowerPC instructions not implemented in the 603e). These do not include those optional instructions treated as no-ops.
- Privileged instruction—A privileged instruction type program exception is generated when the execution of a privileged instruction is attempted and the MSR register user privilege bit, MSR[PR], is set. In the 603e, this exception is generated for **mtspr** or **mfspr** with an invalid SPR field if SPR[0] = 1 and MSR[PR] = 1. This may not be true for all PowerPC processors.

- Trap—A trap type program exception is generated when any of the conditions specified in a trap instruction is met.

4.5.7.1 IEEE Floating-Point Exception Program Exceptions

Floating-point exceptions (not supported on the EC603e microprocessor) are signaled by condition bits set in the floating-point status and control register (FPSCR). They can cause the system floating-point enabled exception handler to be invoked. The 603e handles all floating-point exceptions precisely. The 603e implements the FPSCR as it is defined by the PowerPC architecture; for more information about the FPSCR, see *The Programming Environments Manual*.

Floating-point operations that change exception sticky bits in the FPSCR may suffer a performance penalty. When an exception is disabled in the FPSCR and MSR[FE] = 0, updates to the FPSCR exception sticky bits are serialized at the completion stage. This serialization may result in a one- or two-cycle execution delay. The penalty is incurred only when the exception bit is changed and not on subsequent operations with the same exception. See Chapter 6, “Instruction Timing,” for a full description of completion serialization.

When an exception is enabled in the FPSCR, the instruction traps to the emulation trap exception vector without updating the FPSCR or the target FPR. The emulation trap exception handler is required to complete the instruction. The emulation trap exception handler is invoked regardless of the FE setting in the MSR.

The two IEEE floating-point imprecise modes, defined by the PowerPC architecture when MSR[FE0] ≠ MSR[FE1], are treated as precise exceptions (that is, MSR[FE0] = MSR[FE1] = 1). This is regardless of the setting of MSR[NI].

For the highest and most predictable floating-point performance, all exceptions should be disabled in the FPSCR and MSR. For more information about the program exception, see *The Programming Environments Manual*.

4.5.7.2 Illegal, Reserved, and Unimplemented Instructions Program Exceptions

In accordance with the PowerPC architecture, the 603e considers all instructions defined for 64-bit implementations and unimplemented optional instructions, such as **fsqrt**, **eciwx**, and **ecowx** as illegal and takes a program exception when one of these instructions is encountered. Likewise, if a supervisor-level instruction is encountered when the processor is in user-level mode, a privileged instruction-type program exception is taken.

The 603e implements some instructions, such as double-precision floating-point and load/store string instructions in software. These instructions take the 603e-specific emulation trap exception (0x01600) rather than a program exception.

4.5.8 Floating-Point Unavailable Exception (0x00800)

A floating-point unavailable exception occurs when no higher priority exception exists, an attempt is made to execute a floating-point instruction (including floating-point load, store, and move instructions), and the floating-point available bit in the MSR is disabled (MSR[FP] = 0); note that on the EC603e microprocessor, the MSR[FP] is always cleared to 0. Register settings for this exception are described in Chapter 6, “Exceptions,” in *The Programming Environments Manual*

When a floating-point unavailable exception is taken, instruction execution for the handler begins at offset 0x00800 from the physical base address indicated by MSR[IP].

4.5.9 Decrementer Exception (0x00900)

The 603e implements the decrementer interrupt exception as it is defined in the PowerPC architecture. A decrementer exception request is made when the decrementer counts down through zero. The request is held until there are no higher priority exceptions and MSR[EE] = 1. At this point the decrementer exception is taken. If multiple decrementer exception requests are received before the first can be reported, only one exception is reported. The occurrence of a decrementer exception cancels the request. Register settings for this exception are described in Chapter 6, “Exceptions,” in *The Programming Environments Manual.*

When a decrementer exception is taken, instruction execution for the handler begins at offset 0x00900 from the physical base address indicated by MSR[IP].

4.5.10 System Call Exception (0x00C00)

The 603e implements the system call exception as it is defined by the PowerPC architecture. A system call exception request is made when a system call (**sc**) instruction is completed. If no higher priority exception exists, the system call exception is taken, with SRR0 being set to the EA of the instruction following the **sc** instruction. Register settings for this exception are described in Chapter 6, “Exceptions,” in *The Programming Environments Manual.*

When a system call exception is taken, instruction execution for the handler begins at offset 0x00C00 from the physical base address indicated by MSR[IP].

4.5.11 Trace Exception (0x00D00)

The trace exception is taken under one of the following conditions:

- When MSR[SE] is set, a single-step instruction trace exception is taken when no higher priority exception exists and any instruction (other than **rfi** or **isync**) is successfully completed. Note that other PowerPC processors will take the trace exception on **isync** instructions (when MSR[SE] is set); the 603e does not take the trace exception on **isync** instructions. Single-step instruction trace mode is described in Section 4.5.11.1, "Single-Step Instruction Trace Mode."
- When MSR[BE] is set, the branch trace exception is taken after each branch instruction is completed.
- The 603e deviates from the architecture by not taking trace exceptions on **isync** instructions. Single-step instruction trace mode is described in Section 4.5.11.2, "Branch Trace Mode."

Successful completion implies that the instruction caused no other exceptions. A trace exception is never taken for an **sc** instruction or for a trap instruction that takes a trap exception.

MSR[SE] and MSR[BE] are cleared when the trace exception is taken. In the normal use of this function, MSR[SE] and MSR[BE] are restored when the exception handler returns to the interrupted program using an **rfi** instruction.

Register settings for the trace mode are described in Table 4-15.

Table 4-15. Trace Exception—Register Settings

Register	Setting Description							
SRR0	Set to the address of the instruction following the one for which the trace exception was generated.							
SRR1	0–15 Cleared 16–31 Loaded from bits 16–31 of the MSR							
MSR	POW	0	EE	0	FE0[2]	0	IR	0
	TGPR	0	PR	0	SE	0	DR	0
	ILE	—	FP[1]	0	BE	0	RI	0
	IP	—	ME	—	FE1[2]	0	LE	Set to value of ILE

Notes:

1. The floating-point available bit is always cleared to 0 on the EC603e microprocessor.
2. FE0 and FE1 are not supported on the EC603e microprocessor.

Note that a trace or instruction address breakpoint exception condition generates a soft stop instead of an exception if soft stop has been enabled by the JTAG/COP logic. If trace and breakpoint conditions occur simultaneously, the breakpoint conditions receive higher priority.

When a trace exception is taken, instruction execution for the handler begins as offset 0x00D00 from the base address indicated by MSR[IP].

4.5.11.1 Single-Step Instruction Trace Mode

The single-step instruction trace mode is enabled by setting MSR[SE]. Encountering the single-step breakpoint causes one of the following actions:

- Trap to address vector 0x00D00
- Soft stop (wait for quiescence)

The default single-step trace action traps after an instruction execution and completion. The soft stop option, in which the 603e stops in a restartable state after an instruction execution and completion, can be enabled only through the COP function. The ESP, which interfaces to the COP, can restart the 603e after a soft stop. Refer to the section on JTAG/COP and Section 8.9, “IEEE 1149.1-Compliant Interface,” for more information.

4.5.11.2 Branch Trace Mode

The branch trace mode is enabled by setting MSR[BE]. Encountering the branch trace breakpoint causes one of the following actions:

- Trap to interrupt vector 0x00D00
- Soft stop
- Hard stop

The default branch trace action is to trap after the completion of any branch instruction whenever MSR[BE] is set. However, if soft stop is enabled through the COP interface, the 603e stops in a restartable state. If hard stop is enabled through the COP interface, the 603e stops immediately without waiting to reach a restartable state. Therefore, the 603e is not guaranteed to be restartable after a hard stop. For more information, see Section 8.9, “IEEE 1149.1-Compliant Interface.”

4.5.12 Instruction TLB Miss Exception (0x01000)

When the effective address for an instruction load, store, or cache operation cannot be translated by the ITLBs, an instruction TLB miss exception is generated. Register settings for the instruction and data TLB miss exceptions are described in Table 4-16.

Table 4-16. Instruction and Data TLB Miss Exceptions—Register Settings

Register	Setting Description
SRR0	Set to the address of the next instruction to be executed in the program for which the TLB miss exception was generated.
SRR1	0–3 Loaded from Condition register CR0 field
	4–12 Cleared
	13 0 = data TLB miss
	1 = instruction TLB miss
	14 0 = replace TLB associativity set 0
	1 = replace TLB associativity set 1
	15 0 = data TLB miss on store (or C = 0)
	1 = data TLB miss on load
	16–31 Loaded from bits 16–31 of the MSR
MSR	POW 0 EE 0 FE0[2] 0 IR 0
	TGPR 1 PR 0 SE 0 DR 0
	ILE — FP[1] 0 BE 0 RI 0
	IP — ME — FE1[2] 0 LE Set to value of ILE

Notes:

1. The floating-point available bit is always cleared to 0 on the EC603e microprocessor.
2. FE0 and FE1 are not supported on the EC603e microprocessor.

If the instruction TLB miss exception handler fails to find the desired PTE, then a page fault must be synthesized. The handler must restore the machine state and turn off the GPRs before invoking the ISI exception (0x00400).

Software table search operations are discussed in Chapter 5, “Memory Management.”

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

4.5.13 Data TLB Miss on Load Exception (0x01100)

When the effective address for a data load or cache operation cannot be translated by the DTLBs, a data TLB miss on load exception is generated. Register settings for the instruction and data TLB miss exceptions are described in Table 4-16.

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

Software table search operations are discussed in Chapter 5, “Memory Management.”

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

4.5.14 Data TLB Miss on Store Exception (0x01200)

When the effective address for a data store or cache operation cannot be translated by the DTLBs, a data TLB miss on store exception is generated. The data TLB miss on store exception is also taken when the changed bit (C = 0) for a DTLB entry needs to be updated for a store operation. Register settings for the instruction and data TLB miss exceptions are described in Table 4-16.

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

Software table search operations are discussed in Chapter 5, “Memory Management.”

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

4.5.15 Instruction Address Breakpoint Exception (0x01300)

The instruction address breakpoint is controlled by the IABR special purpose register. IABR[0–29] holds an effective address to which each instruction is compared. The exception is enabled by setting IABR[30]. Note that the 603e ignores the translation enable bit (IABR[31]). The exception is taken when an instruction breakpoint address matches on the next instruction to complete. The instruction tagged with the match is not completed before the instruction address breakpoint exception is taken.

The breakpoint action can be one of the following:

- Trap to interrupt vector 0x01300 (default)
- Soft stop

The bit settings for when an instruction address breakpoint exception is taken are shown in Table 4-17.

Table 4-17. Instruction Address Breakpoint Exception—Register Settings

Register	Setting Description
SRR0	Set to the address of the next instruction to be executed in the program for which the TLB miss exception was generated.
SRR1	0–15 Cleared 16–31 Loaded from bits 16–31 of the MSR
MSR	POW 0 · EE 0 · FE0[2] 0 · IR 0 TGPR 0 · PR 0 · SE 0 · DR 0 ILE — · FP[1] 0 · BE 0 · RI 0 IP — · ME — · FE1[2] 0 · LE Set to value of ILE

Notes:

1. The floating-point available bit is always cleared to 0 on the EC603e microprocessor.
2. FE0 and FE1 are not supported on the EC603e microprocessor.

The default breakpoint action is to trap before the execution of the matching instruction.

The soft stop feature can be enabled only through the COP interface. With soft stop enabled, the 603e stops in a restartable state, while with hard stop enabled, the 603e stops immediately without attempting to reach a restartable state. Upon restarting from a soft stop, the matching instructions are executed and completed unless it generates an exception. For soft stops, the next ten instructions that could have passed the IABR check can be monitored only by single-stepping the processor. When soft stops are used, the address compare must be separated by at least 10 instructions.

If soft stop is enabled, only one soft stop is generated before completion of an instruction with an IABR match, regardless of whether a soft stop is generated before that instruction for any other reason, such as trace mode on for the preceding instruction or a COP soft stop request.

Table 4-18 shows the priority of actions taken when more than one mode is enabled for the same instruction.

Table 4-18. Breakpoint Action for Multiple Modes Enabled for the Same Address

IABR[IE]	MSR[BE]	MSR[SE]	First Action	Next Action	Comments
1	1	0	Instruction address	Trace (branch)	Enabling both modes is useful only if both trace and address breakpoint interrupts are needed.
1	0	1	Instruction address breakpoint	Trace (single-step)	Enabling both modes is useful only if different breakpoint actions are required.
0	1	1	Trace (branch)	None	The action for branch trace and single-step trace is the same. Enabling both trace modes is redundant except for hard stop on branches.
1	1	1	Instruction address breakpoint	Trace	Enabling all modes is redundant. This entry is for clarification only.

Note that a trace or instruction address breakpoint exception condition generates a soft stop instead of an exception if soft stop has been enabled by the JTAG/COP logic. If trace and breakpoint conditions occur simultaneously, the breakpoint conditions receive higher priority.

The 603e requires that an **mtspr** instruction that updates the IABR be followed by a context-synchronizing instruction. If the **mtspr** instruction enables the instruction address breakpoint exception, the context-synchronizing instruction cannot generate a breakpoint response. The 603e also cannot block a breakpoint response on the context-synchronizing instruction if the breakpoint was disabled by the **mtspr** instruction. See “Synchronization Requirements for Special Registers and TLBs” in Chapter 2, “Register Set,” in *The Programming Environments Manual*” for more information on this requirement.

4.5.16 System Management Interrupt (0x01400)

The system management interrupt behaves like an external interrupt except for the signal asserted and the vector taken. A system management interrupt is signaled to the 603e 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 $\overline{\text{SMI}}$ is detected (and provided that MSR[EE] is set), the 603e generates a recoverable halt to instruction completion. The 603e requires the next instruction in program order to complete or except, block completion of any following instructions, and allow the completed store queue to drain. If any higher priority exceptions are encountered in this process, they are taken first and the system management interrupt is delayed until a recoverable halt is achieved. At this time the 603e saves state information and takes the system management interrupt.

The register settings for the external interrupt exception are shown in Table 4-19.

Table 4-19. System Management Interrupt—Register Settings

Register	Setting Description
SRR0	Set to the effective address of the instruction that the processor would have attempted to complete next if no interrupt conditions were present.
SRR1	0–15 Cleared 16–31 Loaded from bits 16–31 of the MSR
MSR	POW 0 EE 0 FE0[2] 0 IR 0 TGPR 0 PR 0 SE 0 DR 0 ILE — FP[1] 0 BE 0 RI 0 IP — ME — FE1[2] 0 LE Set to value of ILE

Notes:

1. The floating-point available bit is always cleared to 0 on the EC603e microprocessor.
2. FE0 and FE1 are not supported on the EC603e microprocessor.

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 603e recognizes the interrupt condition ($\overline{\text{SMI}}$ asserted) only if the MSR[EE] bit is set; and ignores the interrupt condition otherwise. To guarantee that the external interrupt is taken, the $\overline{\text{SMI}}$ signal must be held active until the 603e takes the interrupt. If the $\overline{\text{SMI}}$ signal is negated before the interrupt is taken, the 603e 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 5
Memory Management

This chapter describes the PowerPC 603e microprocessor's implementation of the memory management unit (MMU) specifications provided by the PowerPC operating environment architecture (OEA) for PowerPC processors. The 603e MMU implementation is very similar to that of the PowerPC 603 microprocessor except that the 603e implements an extra key bit in the SRR1 register that simplifies the table search software. In addition, because the 603e does not support direct-store bus accesses, it causes a DSI exception when a direct-store segment is encountered. Refer to Appendix C, "PowerPC 603 Processor System Design and Programming Considerations," for a complete description of the differences applicable to the PowerPC 603 microprocessor.

The primary function of the MMU in a PowerPC processor is the translation of logical (effective) addresses to physical addresses (referred to as real addresses in the architecture specification) for memory accesses, and I/O accesses (I/O accesses are assumed to be memory-mapped). In addition, the MMU provides access protection on a segment, block, or page basis. This chapter describes the specific hardware used to implement the MMU model of the OEA in the 603e. Refer to Chapter 7, "Memory Management," in *The Programming Environments Manual* for a complete description of the conceptual model.

Two general types of accesses generated by PowerPC processors require address translation—instruction accesses, and data accesses to memory generated by load and store instructions. Generally, the address translation mechanism is defined in terms of segment descriptors and page tables used by PowerPC processors to locate the effective-to-physical address mapping for instruction and data accesses. The segment information translates the effective address to an interim virtual address, and the page table information translates the virtual address to a physical address.

The segment descriptors, used to generate the interim virtual addresses, are stored as on-chip segment registers on 32-bit implementations (such as the 603e). In addition, two translation lookaside buffers (TLBs) are implemented on the 603e to keep recently-used page address translations on-chip. Although the PowerPC OEA describes one MMU (conceptually), the 603e hardware maintains separate TLBs and table search resources for instruction and data accesses that can be accessed independently (and simultaneously). Therefore, the 603e is described as having two MMUs, one for instruction accesses (IMMU) and one for data accesses (DMMU).

The block address translation (BAT) mechanism is a software-controlled array that stores the available block address translations on-chip. BAT array entries are implemented as pairs of BAT registers that are accessible as supervisor-level special-purpose registers (SPRs). There are separate instruction and data BAT mechanisms, and in the 603e, they reside in the instruction and data MMUs respectively.

The MMUs, together with the exception processing mechanism, provide the necessary support for the operating system to implement a paged virtual memory environment and for enforcing protection of designated memory areas. Exception processing is described in Chapter 4, “Exceptions.” Section 4.2, “Exception Processing,” describes the MSR, which controls some of the critical functionality of the MMUs.

5.1 MMU Features

The 603e implements the memory management specification of the PowerPC OEA for 32-bit implementations. Thus, it provides 4 Gbytes of effective address space accessible to supervisor and user programs with a 4-Kbyte page size and 256-Mbyte segment size. In addition, the MMUs of 32-bit PowerPC processors use an interim virtual address (52 bits) and hashed page tables in the generation of 32-bit physical addresses. PowerPC processors also have a block address translation (BAT) mechanism for mapping large blocks of memory. Block sizes range from 128 Kbyte to 256 Mbyte and are software-programmable.

The 603e completely implements all features required by the MMU specifications of the PowerPC architecture (OEA) for 32-bit implementations. Table 5-1 summarizes all 603e MMU features including the architectural features of PowerPC MMUs (defined by the OEA) for 32-bit processors and the implementation-specific features provided by the 603e.

Table 5-1. MMU Features Summary

Feature Category	Architecturally Defined/ 603e-Specific	Feature
Address ranges	Architecturally defined	2^{32} bytes of effective address
		2^{52} bytes of virtual address
		2^{32} bytes of physical address
Page size	Architecturally defined	4 Kbytes
Segment size	Architecturally defined	256 Mbytes
Block address translation	Architecturally defined	Range of 128 Kbyte–256 Mbytes sizes
		Implemented with IBAT and DBAT registers in BAT array
Memory protection	Architecturally defined	Segments selectable as no-execute
		Pages selectable as user/supervisor and read-only
		Blocks selectable as user/supervisor and read-only
Page history	Architecturally defined	Referenced and changed bits defined and maintained

Table 5-1. MMU Features Summary (Continued)

Feature Category	Architecturally Defined/ 603e-Specific	Feature
Page address translation	Architecturally defined	Translations stored as PTEs in hashed page tables in memory
		Page table size determined by mask in SDR1 register
TLBs	Architecturally defined	Instructions for maintaining optional TLBs (**tlbie** instruction in 603e)
	603e-specific	64-entry, two-way set associative ITLB 64-entry, two-way set associative DTLB
Segment descriptors	Architecturally defined	Stored as segment registers on-chip
Page table search support	603e-specific	Three MMU exceptions defined: ITLB miss exception, DTLB miss on load exception, and DTLB miss on store (or C = 0) exception; MMU-related bits set in SRR1 for these exceptions
		IMISS and DMISS registers (missed effective address) HASH1 and HASH2 registers (PTEG addr) ICMP and DCMP registers (for comparing PTEs) RPA register (for loading TLBs)
		tlbli rB instruction for loading ITLB entries **tlbld r**B instruction for loading DTLB entries
		Shadow registers for GPR0–GPR3 (can use **r0**–**r3** in table search handler without corruption of **r0**–**r3** in context that was previously executing)

5.1.1 Memory Addressing

A program references memory using the effective (logical) address computed by the processor when it executes a load, store, or cache instruction, and when it fetches the next instruction. The effective address is translated to a physical address according to the procedures described in Chapter 7, "Memory Management," in *The Programming Environments Manual*, augmented with information in this chapter. The memory subsystem uses the physical address for the access.

For a complete discussion of effective address calculation, see Section 2.3.2.3, "Effective Address Calculation."

5.1.2 MMU Organization

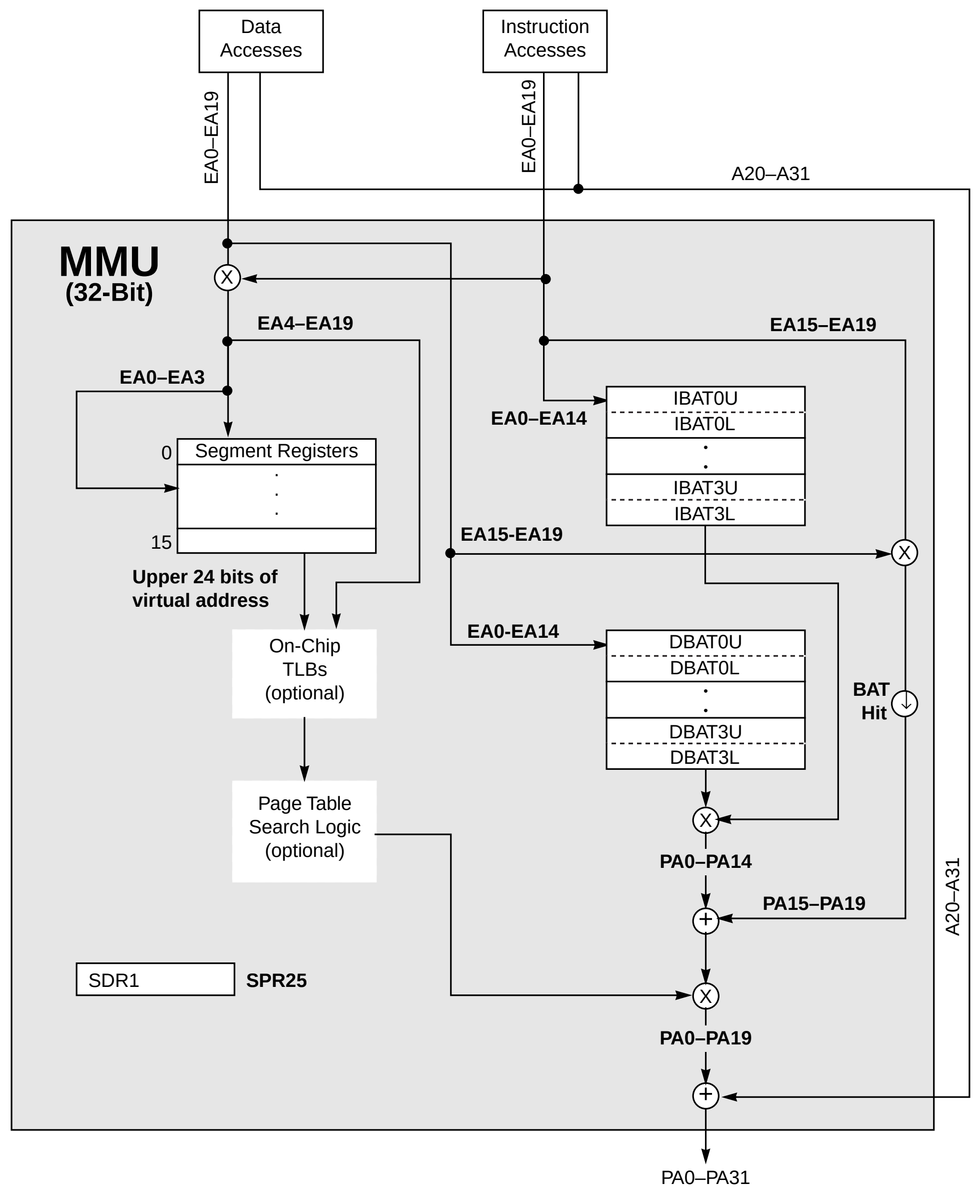
Figure 5-1 shows the conceptual organization of a PowerPC MMU in a 32-bit implementation; note that it does not describe the specific hardware used to implement the memory management function for a particular processor. Processors may optionally implement on-chip TLBs and may optionally support the automatic search of the page tables for PTEs. In addition, other hardware features (invisible to the system software) not depicted in the figure may be implemented.

Figure 5-2 and Figure 5-3 show the conceptual organization of the 603e instruction and data MMUs, respectively. The instruction addresses shown in Figure 5-2 are generated by the processor for sequential instruction fetches and addresses that correspond to a change of program flow. Data addresses shown in Figure 5-3 are generated by load and store instructions and by cache instructions.

As shown in the figures, after an address is generated, the higher-order bits of the effective address, EA0–EA19 (or a smaller set of address bits, EA0–EA*n*, in the cases of blocks), are translated into physical address bits PA0–PA19. The lower-order address bits, A20–A31 are untranslated and therefore identical for both effective and physical addresses. After translating the address, the MMUs pass the resulting 32-bit physical address to the memory subsystem.

In addition to the higher-order address bits, the MMUs automatically keep an indicator of whether each access was generated as an instruction or data access and a supervisor/user indicator that reflects the state of the PR bit of the MSR when the effective address was generated. In addition, for data accesses, there is an indicator of whether the access is for a load or a store operation. This information is then used by the MMUs to appropriately direct the address translation and to enforce the protection hierarchy programmed by the operating system. Section 4.2, “Exception Processing,” describes the MSR, which controls some of the critical functionality of the MMUs.

The figures show the way in which the A20–A26 address bits index into the on-chip instruction and data caches to select a cache set. The remaining physical address bits are then compared with the tag fields (comprised of bits PA0–PA19) of the four selected cache blocks to determine if a cache hit has occurred. In the case of a cache miss, the instruction or data access is then forwarded to the bus interface unit which then initiates an external memory access.



Optional

Figure 5-1. MMU Conceptual Block Diagram—32-Bit Implementations

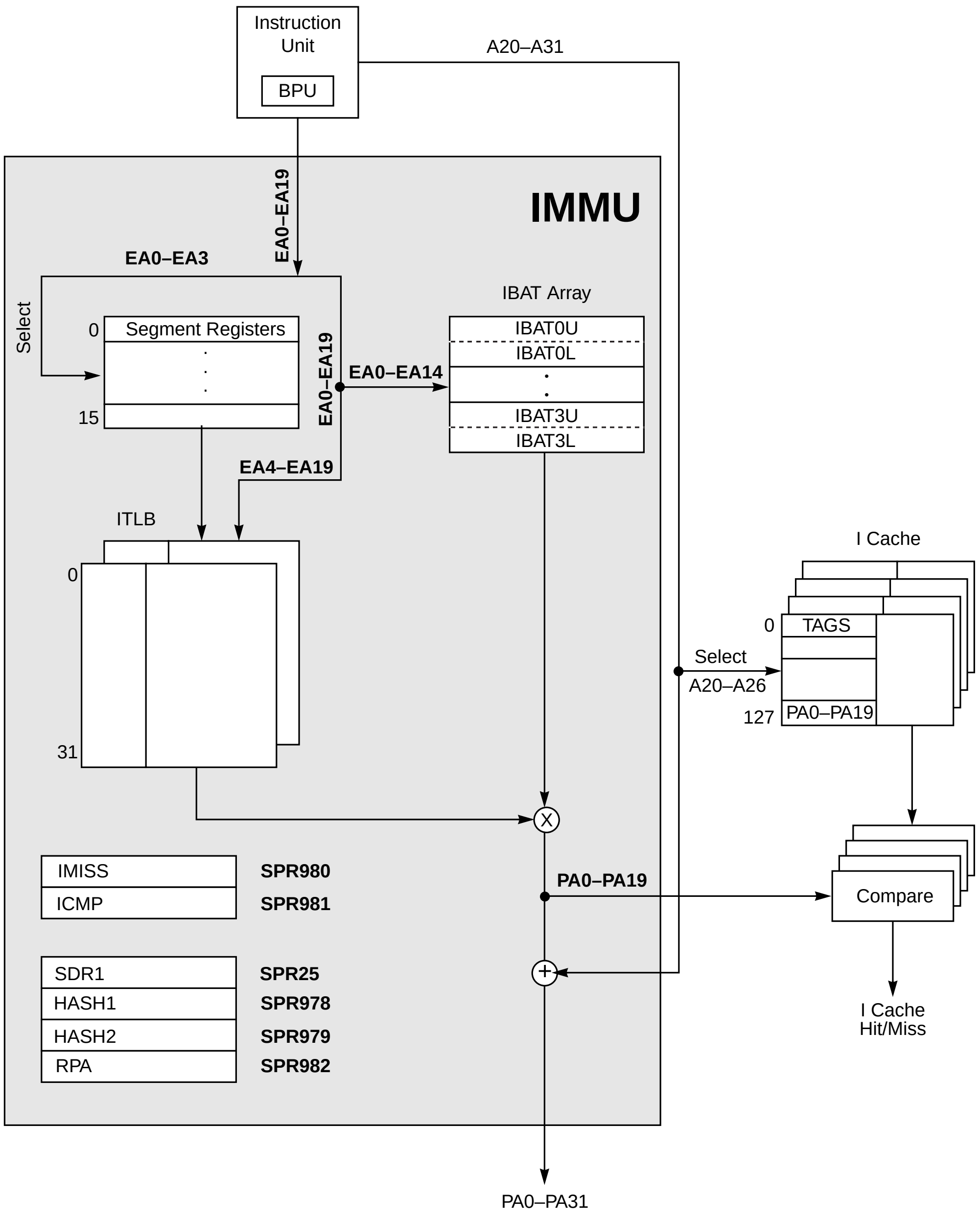


Figure 5-2. IMMU Block Diagram

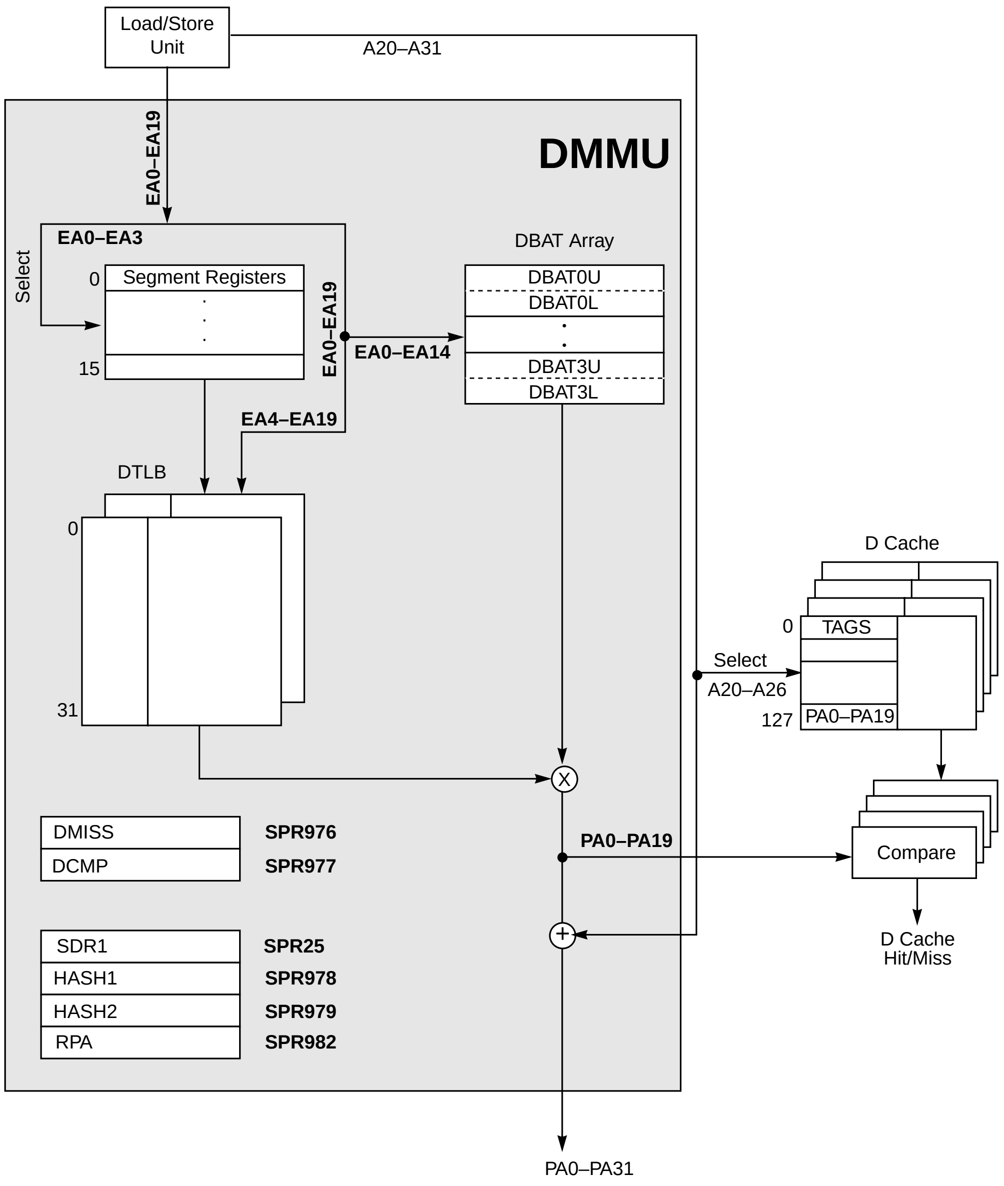


Figure 5-3. DMMU Block Diagram

5.1.3 Address Translation Mechanisms

PowerPC processors support the following four types of address translation:

- Page address translation—translates the page frame address for a 4-Kbyte page size
- Block address translation—translates the block number for blocks that range in size from 128 Kbyte to 256 Mbyte
- Direct-store interface address translation—used to generate direct-store interface accesses on the external bus; not implemented in the 603e.
- Real addressing mode translation—when address translation is disabled, the physical address is identical to the effective address.

Figure 5-4 shows the three implemented address translation mechanisms provided by the 603e MMUs. The segment descriptors shown in the figure control the page address translation mechanism. When an access uses page address translation, the appropriate segment descriptor is required. In 32-bit implementations, one of the 16 on-chip segment registers (which contain segment descriptors) is selected by the four highest-order effective address bits.

A control bit in the corresponding segment descriptor then determines if the access is to memory (memory-mapped) or to the direct-store interface space (selected when the direct-store translation control bit (T bit) in the corresponding segment descriptor is set). Note that the direct-store interface is present only for compatibility with existing I/O devices that use this interface. When an access is determined to be to the direct-store interface space, the 603e takes a DSI exception as described in Section 4.5.3, “DSI Exception (0x00300)” if it is a data access, and takes an ISI exception as described in Section 4.5.4, “ISI Exception (0x00400)” if it is an instruction access.

For memory accesses translated by a segment descriptor, the interim virtual address is generated using the information in the segment descriptor. Page address translation corresponds to the conversion of this virtual address into the 32-bit physical address used by the memory subsystem. In most cases, the physical address for the page resides in an on-chip TLB and is available for quick access. However, if the page address translation misses in an on-chip TLB, the MMU causes a search of the page tables in memory (using the virtual address information and a hashing function) to locate the required physical address. When this occurs, the 603e vectors to exception handlers that search the page tables with software.

Block address translation occurs in parallel with page address translation and is similar to page address translation; however, fewer higher-order effective address bits are translated into physical address bits (more lower-order address bits (at least 17) are untranslated to form the offset into a block). Also, instead of segment descriptors and a TLB, block address translations use the on-chip BAT registers as a BAT array. If an effective address matches the corresponding field of a BAT register, the information in the BAT register is used to generate the physical address; in this case, the results of the page translation (occurring in parallel) are ignored (even if the segment corresponds to the direct-store interface space).

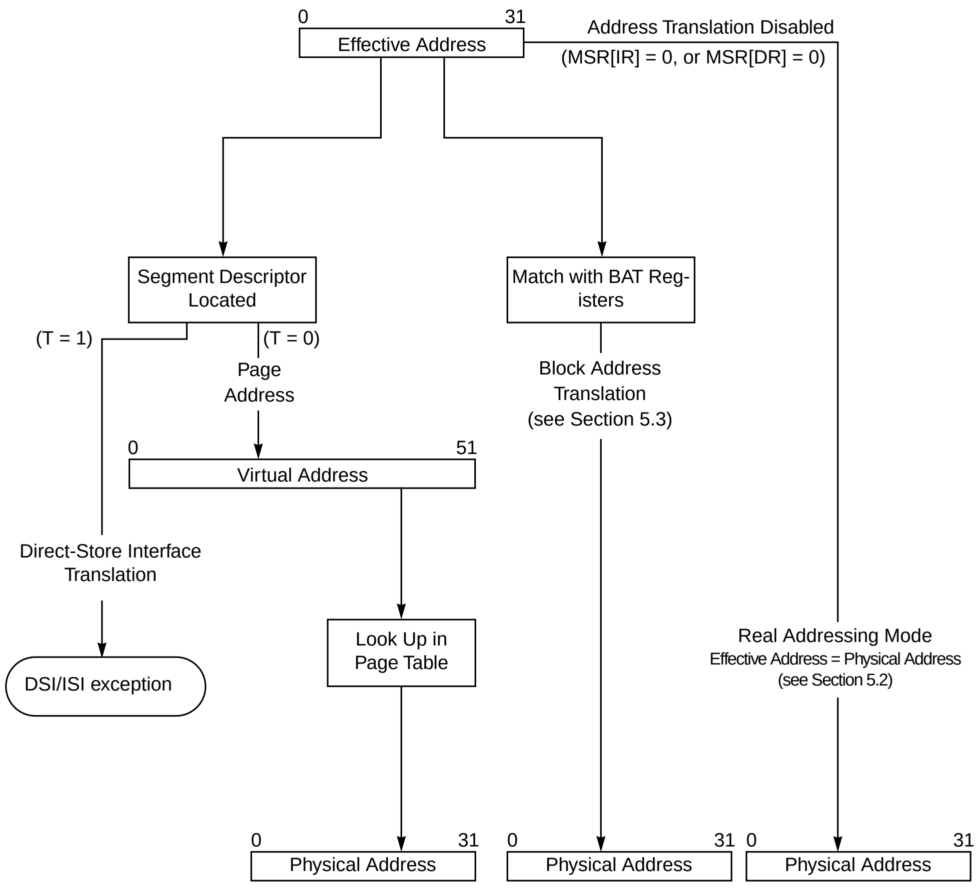


Figure 5-4. Address Translation Types

Real addressing mode translation occurs when address translation is disabled; in this case the physical address generated is identical to the effective address. Instruction and data address translation is enabled with the MSR[IR] and MSR[DR] bits, respectively. Thus when the processor generates an access, and the corresponding address translation enable bit in MSR (MSR[IR] for instruction accesses and MSR[DR] for data accesses) is cleared, the resulting physical address is identical to the effective address and all other translation mechanisms are ignored.

5.1.4 Memory Protection Facilities

In addition to the translation of effective addresses to physical addresses, the MMUs provide access protection of supervisor areas from user access and can designate areas of memory as read-only as well as no-execute or guarded. Table 5-2 shows the eight protection options supported by the MMUs for pages.

Table 5-2. Access Protection Options for Pages

Option	User Read		User Write	Supervisor Read		Supervisor Write
	I-Fetch	Data		I-Fetch	Data	
Supervisor-only	—	—	—	√	√	√
Supervisor-only-no-execute	—	—	—	—	√	√
Supervisor-write-only	√	√	—	√	√	√
Supervisor-write-only-no-execute	—	√	—	—	√	√
Both user/supervisor	√	√	√	√	√	√
Both user/supervisor-no-execute	—	√	√	—	√	√
Both read-only	√	√	—	√	√	—
Both read-only-no-execute	—	√	—	—	√	—

√ access permitted
— protection violation

The operating system programs whether instructions can be fetched from an area of memory by appropriately using the no-execute option provided in the segment descriptor. Each of the remaining options is enforced based on a combination of information in the segment descriptor and the page table entry. Thus, the supervisor-only option allows only read and write operations generated while the processor is operating in supervisor mode (corresponding to MSR[PR] = 0) to access the page. User accesses that map into a supervisor-only page cause an exception to be taken.

Finally, there is a facility in the VEA and OEA that allows pages or blocks to be designated as guarded preventing out-of order accesses that may cause undesired side effects. For example, areas of the memory map that are used to control I/O devices can be marked as guarded so that accesses (for example, instruction prefetches) do not occur unless they are explicitly required by the program.

For more information on memory protection, see “Memory Protection Facilities,” in Chapter 7, “Memory Management,” in the *The Programming Environments Manual*.

5.1.5 Page History Information

The MMUs of PowerPC processors also define referenced (R) and changed (C) bits in the page address translation mechanism that can be used as history information relevant to the page. This information can then be used by the operating system to determine which areas of memory to write back to disk when new pages must be allocated in main memory. While these bits are initially programmed by the operating system into the page table, the architecture specifies that the R and C bits may be maintained either by the processor hardware (automatically) or by some software-assist mechanism that updates these bits when required. The software table search routines used by the 603e set the R bit when a PTE is accessed; the 603e causes an exception (to vector to the software table search routines) when the C bit in the corresponding TLB entry requires updating.

5.1.6 General Flow of MMU Address Translation

The following sections describe the general flow used by PowerPC processors to translate effective addresses to virtual and then physical addresses.

5.1.6.1 Real Addressing Mode and Block Address Translation Selection

When an instruction or data access is generated and the corresponding instruction or data translation is disabled (MSR[IR] = 0 or MSR[DR] = 0), real addressing mode translation is used (physical address equals effective address) and the access continues to the memory subsystem as described in Section 5.2, "Real Addressing Mode."

Figure 5-5 shows the flow used by the MMUs in determining whether to select real addressing mode, block address translation or to use the segment descriptor to select page address translation.

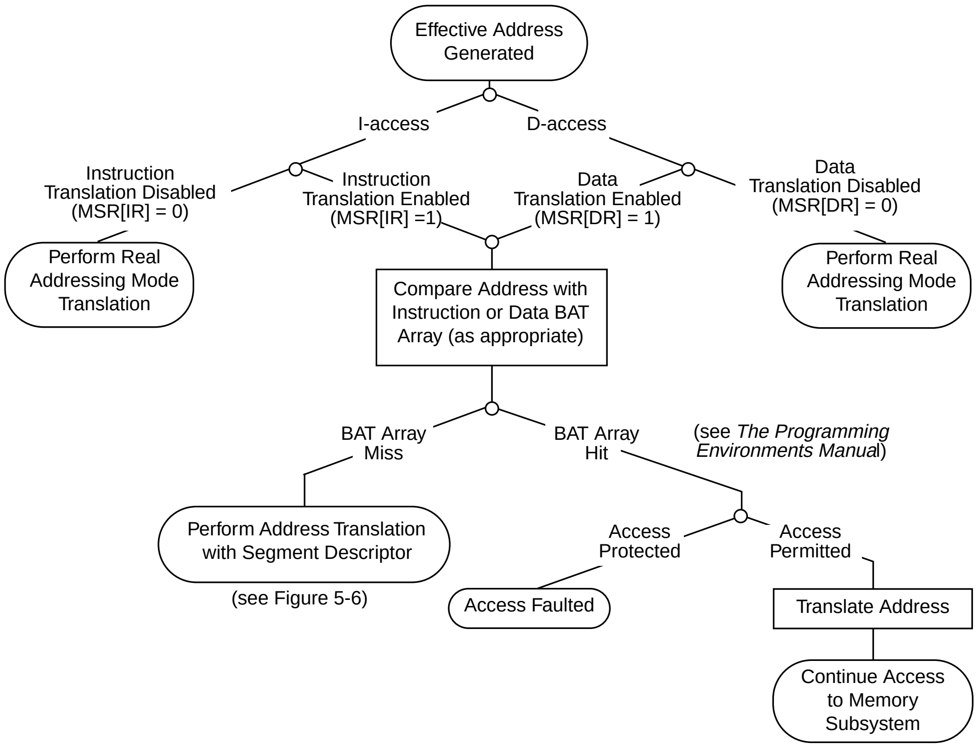

Effective Address Generated
I-access
D-access
Instruction Translation Disabled (MSR[IR] = 0)
Instruction Translation Enabled (MSR[IR] =1)
Data Translation Enabled (MSR[DR] = 1)
Data Translation Disabled (MSR[DR] = 0)
Perform Real Addressing Mode Translation
Compare Address with Instruction or Data BAT Array (as appropriate)
Perform Real Addressing Mode Translation
BAT Array Miss
BAT Array Hit
(see The Programming Environments Manual)
Perform Address Translation with Segment Descriptor
(see Figure 5-6)
Access Protected
Access Permitted
Access Faulted
Translate Address
Continue Access to Memory Subsystem

Figure 5-5. General Flow of Address Translation (Real Addressing Mode and Block)

Note that if the BAT array search results in a hit, the access is qualified with the appropriate protection bits. If the access violates the protection mechanism, an exception (ISI or DSI exception) is generated.

5.1.6.2 Page Address Translation Selection

If address translation is enabled (real addressing mode not selected) and the effective address information does not match with a BAT array entry, then the segment descriptor must be located. Once the segment descriptor is located, the T bit in the segment descriptor selects whether the translation is to a page or to a direct-store interface segment as shown in Figure 5-6. Note that the 603e does not implement the direct-store interface, and accesses to these segments cause a DSI exception. In addition, Figure 5-6 also shows the way in which the no-execute protection is enforced; if the N bit in the segment descriptor is set and the access is an instruction fetch, the access is faulted as described in Chapter 7, “Memory Management,” in *The Programming Environments Manual*. Note that the figure shows the flow for these cases as described by the PowerPC OEA, and so the TLB references are shown as optional. Since the 603e implements TLBs, these branches are valid, and described in more detail throughout this chapter.

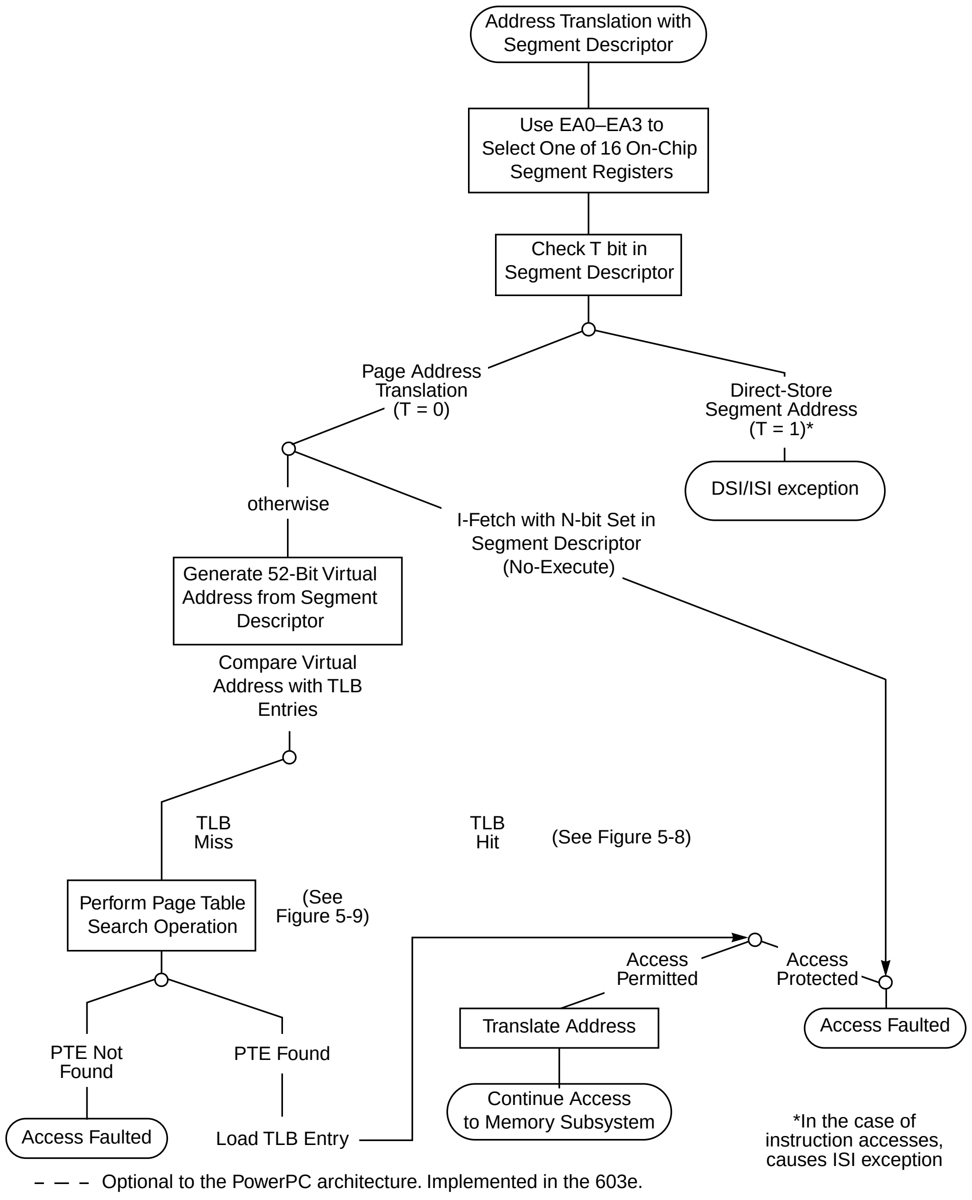

Address Translation with Segment Descriptor
Use EA0–EA3 to Select One of 16 On-Chip Segment Registers
Check T bit in Segment Descriptor
Page Address Translation (T = 0)
Direct-Store Segment Address (T = 1)*
DSI/ISI exception
otherwise
I-Fetch with N-bit Set in Segment Descriptor (No-Execute)
Generate 52-Bit Virtual Address from Segment Descriptor
Compare Virtual Address with TLB Entries
TLB Miss
TLB Hit
(See Figure 5-8)
Perform Page Table Search Operation
(See Figure 5-9)
Access Permitted
Access Protected
Translate Address
Access Faulted
PTE Not Found
PTE Found
Continue Access to Memory Subsystem
Access Faulted
Load TLB Entry
*In the case of instruction accesses, causes ISI exception
– — – Optional to the PowerPC architecture. Implemented in the 603e.

Figure 5-6. General Flow of Page and Direct-Store Interface Address Translation

If the T bit in the corresponding segment descriptor is zero, page address translation is selected. The information in the segment descriptor is then used to generate the 52-bit virtual address. The virtual address is then used to identify the page address translation information (stored as page table entries (PTEs) in a page table in memory). For increased performance, the 603e has two TLBs to store recently-used PTEs on-chip.

If an access hits in the appropriate TLB, the page translation occurs and the physical address bits are forwarded to the memory subsystem. If the required PTE is not resident, the MMU requires a search of the page table. In this case, the 603e traps to one of three exception handlers for the system software to perform the page table search. If the PTE is successfully matched, a new TLB entry is created and the page translation is once again attempted. This time, the TLB is guaranteed to hit. Once the PTE is located, the access is qualified with the appropriate protection bits. If the access is a protection violation (not allowed), an exception (instruction access or data access) is generated.

If the PTE is not found by the table search operation, a page fault condition exists, and the TLB miss exception handlers synthesize either an ISI or DSI exception to handle the page fault.

5.1.7 MMU Exceptions Summary

In order to complete any memory access, the effective address must be translated to a physical address. In the 603e, an MMU exception condition occurs if this translation fails for one of the following reasons:

- Page fault—there is no valid entry in the page table for the page specified by the effective address (and segment descriptor) and there is no valid BAT translation.
- An address translation is found but the access is not allowed by the memory protection mechanism.

Additionally, because the 603e relies on software to perform table search operations, the processor also takes an exception when:

- There is a miss in the corresponding (instruction or data) TLB.
- The page table requires an update to the changed (C) bit.

The state saved by the processor for each of these exceptions contains information that identifies the address of the failing instruction. Refer to Chapter 4, “Exceptions,” for a more detailed description of exception processing.

Because a page fault condition (PTE not found in the page tables in memory) is detected by the software that performs the table search operation (and not the 603e hardware), it does not cause 603e exception in the strictest sense in that exception processing as described in Chapter 4, “Exceptions,” does not occur. However, in order to maintain architectural compatibility with software written for other PowerPC devices, the software that detects this condition should synthesize an exception by setting the appropriate bits in the DSISR or SRR1 and branching to the ISI or DSI exception handler. Refer to Section 5.5.2, “Implementation-Specific Table Search Operation,” for more information and examples of this exception software. The remainder of this chapter assumes that the table search software emulates this exception and refers to this condition as an exception.

The translation exception conditions defined by the OEA for 32-bit implementations cause either the ISI or the DSI exception to be taken as shown in Table 5-3.

Table 5-3. Translation Exception Conditions

Condition	Description	Exception
Page fault (no PTE found)	No matching PTE found in page tables (and no matching BAT array entry)	I access: ISI exception* SRR1[1] = 1
		D access: DSI exception* DSISR[1] =1
Block protection violation	Conditions described for block in “Block Memory Protection” in Chapter 7, “Memory Management,” in *The Programming Environments Manual.*“	I access: ISI exception SRR1[4] = 1
		D access: DSI exception DSISR[4] =1
Page protection violation	Conditions described for page in “Page Memory Protection” in Chapter 7, “Memory Management,” in *The Programming Environments Manual.*	I access: ISI exception** SRR1[4] = 1
		D access: DSI exception** DSISR[4] =1
No-execute protection violation	Attempt to fetch instruction when SR[N] = 1	ISI exception SRR1[3] = 1
Instruction fetch from direct-store segment	Attempt to fetch instruction when SR[T] = 1	ISI exception SRR1[3] =1
Data access to direct-store segment (including floating-point accesses) **Note**: this is a 603e-specific condition	Attempt to perform load or store (including floating-point load or store***) when SR[T] = 1	DSI exception DSISR[5] =1
Instruction fetch from guarded memory with MSR[IR] = 1	Attempt to fetch instruction when MSR[IR] = 1 and either matching xBAT[G] = 1, or no matching BAT entry and PTE[G] = 1	ISI exception SRR1[3] =1

* The 603e hardware does not vector to these exceptions automatically. It is assumed that the software that performs the table search operations vectors to these exceptions and sets the appropriate bits when a page fault condition occurs.

**The table search software can also vector to these exception conditions

***The EC603e microprocessor does not support the floating-point unit.

In addition to the translation exceptions, there are other MMU-related conditions (some of them defined as implementation-specific and therefore, not required by the architecture) that can cause an exception to occur in the 603e. These exception conditions map to the processor exception as shown in Table 5-4. For example, the 603e also defines three exception conditions to support software table searching. The only exception conditions that occur when MSR[DR] = 0 are the conditions that cause the alignment exception for data accesses. For more detailed information about the conditions that cause the alignment exception (in particular for string/multiple instructions), see Section 4.5.6, "Alignment Exception (0x00600)."

Note that some exception conditions depend upon whether the memory area is set up as write-though (W = 1) or cache-inhibited (I = 1). These bits are described fully in "Memory/Cache Access Attributes," in Chapter 5, "Cache Model and Memory Coherency," of *The Programming Environments Manual*. Refer to Chapter 4, "Exceptions," and to Chapter 6, "Exceptions," in *The Programming Environments Manual* for a complete description of the SRR1 and DSISR bit settings for these exceptions.

Table 5-4. Other MMU Exception Conditions

Condition	Description	Exception

TLB miss for an instruction fetch	No matching entry found in ITLB	Instruction TLB miss exception
		SRR1[13] = 1
		MSR[14] = 1

TLB miss for a data access	No matching entry found in DTLB for data access	Load: data TLB miss on load exception MSR[14] = 1

		Store: data TLB miss on store exception
		SRR1[15] =1
		MSR[14] = 1

Store operation and C = 0	Matching DLTB entry has C = 0 and access is a store	Data TLB miss on store exception
		SRR1[15] =1
		MSR[14] = 1

dcbz with W = 1 or I = 1	**dcbz** instruction to write-through or cache-inhibited segment or block	Alignment exception (not required by architecture for this condition)
dcbz when the data cache is locked	The **dcbz** instruction takes an alignment exception if the data cache is locked (HID0 bits 18 and 19) when it is executed.	Alignment exception
lwarx or **stwcx.** with W = 1	Reservation instruction to write-through segment or block	DSI exception DSISR[5] = 1
lwarx, **stwcx.**, **eciwx**, or **ecowx** instruction to direct-store segment	Reservation instruction or external control instruction when SR[T] =1	DSI exception DSISR[5] = 1
Floating-point load or store to direct-store segment*	FP memory access when SR[T] = 1	See data access to direct-store segment in Table 5-3.

Table 5-4. Other MMU Exception Conditions (Continued)

Condition	Description	Exception
Load or store that results in a direct-store error	Does not occur in 603e	Does not apply
eciwx or **ecowx** attempted when external control facility disabled	**eciwx** or **ecowx** attempted with EAR[E] = 0	DSI exception DSISR[11] = 1
lmw, **stmw**, **lswi**, **lswx**, **stswi**, or **stswx** instruction attempted in little-endian mode	**lmw**, **stmw**, **lswi**, **lswx**, **stswi**, or **stswx** instruction attempted while MSR[LE] = 1	Alignment exception
Operand misalignment	Translation enabled and operand is misaligned as described in Chapter 4, "Exceptions."	Alignment exception (some of these cases are implementation-specific)

*The EC603e microprocessor does not support the floating-point unit.

5.1.8 MMU Instructions and Register Summary

The MMU instructions and registers provide the operating system with the ability to set up the block address translation areas and the page tables in memory.

Note that because the implementation of TLBs is optional, the instructions that refer to these structures are also optional. However, because these structures serve as caches of the page table, the architecture specifies a software protocol for maintaining coherency between these caches and the tables in memory whenever changes are made to the tables in memory. When the tables in memory are changed, the operating system purges these caches of the corresponding entries, allowing the translation caching mechanism to refetch from the tables when the corresponding entries are required.

Note that the 603e implements all TLB-related instructions except **tlbia**, which is treated as an illegal instruction. The 603e also uses some implementation-specific instructions to load two on-chip TLBs.

Because the MMU specification for PowerPC processors is so flexible, it is recommended that the software that uses these instructions and registers be "encapsulated" into subroutines to minimize the impact of migrating across the family of implementations.

Table 5-5 summarizes 603e instructions that specifically control the MMU. For more detailed information about the instructions, refer to Chapter 2, "Programming Model," in this book and Chapter 8, "Instruction Set," in *The Programming Environments Manual.*

Table 5-5. Instruction Summary—MMU Control

Instruction	Description
mtsr SR,**r**S	Move to Segment Register SR[SR#]⟵ **r**S
mtsrin **r**S,**r**B	Move to Segment Register Indirect SR[**r**B[0–3]]⟵**r**S
mfsr **r**D,SR	Move from Segment Register **r**D⟵SR[SR#]
mfsrin **r**D,**r**B	Move from Segment Register Indirect **r**D⟵SR[**r**B[0–3]]
tlbie **r**B*	TLB Invalidate Entry For effective address specified by **r**B, TLB[V]⟵0 The **tlbie** instruction invalidates both TLB entries indexed by the EA, and operates on both the instruction and data TLBs simultaneously invalidating four TLB entries. The index corresponds to bits 15–19 of the EA.
tlbsync*	TLB Synchronize Synchronizes the execution of all other **tlbie** instructions in the system. In the 603e, when the $\overline{\text{TLBISYNC}}$ signal is negated, instruction execution may continue or resume after the completion of a **tlbsync** instruction. When the $\overline{\text{TLBISYNC}}$ signal is asserted, instruction execution stops after the completion of a **tlbsync** instruction.
tlbli (603e-specific)	Load Instruction TLB Entry Loads the contents of the ICMP and RPA registers into the ITLB
tlbld (603e-specific)	Load Data TLB Entry Loads the contents of the DCMP and RPA registers into the DTLB

*These instructions are defined by the PowerPC architecture, but are optional.

Table 5-6 summarizes the registers that the operating system uses to program the 603e MMUs. These registers are accessible to supervisor-level software only. These registers are described in Chapter 2, “Register Set,” in *The Programming Environments Manual*. For 603e-specific registers, see Chapter 2, “Programming Model,” of this book.

Table 5-6. MMU Registers

Register	Description
Segment registers (SR0–SR15)	The sixteen 32-bit segment registers are present only in 32-bit implementations of the PowerPC architecture. The fields in the segment register are interpreted differently depending on the value of bit 0. The segment registers are accessed by the **mtsr**, **mtsrin**, **mfsr**, and **mfsrin** instructions.
BAT registers (IBAT0U–IBAT3U, IBAT0L–IBAT3L, DBAT0U–DBAT3U, and DBAT0L–DBAT3L)	There are 16 BAT registers, organized as four pairs of instruction BAT registers (IBAT0U–IBAT3U paired with IBAT0L–IBAT3L) and four pairs of data BAT registers (DBAT0U–DBAT3U paired with DBAT0L–DBAT3L). The BAT registers are defined as 32-bit registers in 32-bit implementations. These are special-purpose registers that are accessed by the **mtspr** and **mfspr** instructions.
SDR1	The SDR1 register specifies the variable used in accessing the page tables in memory. SDR1 is defined as a 32-bit register for 32-bit implementations. This is a special-purpose register that is accessed by the **mtspr** and **mfspr** instructions.

Table 5-6. MMU Registers (Continued)

Register	Description
Instruction TLB miss address and data TLB miss address registers (IMISS and DMISS)	When a TLB miss exception occurs, the IMISS or DMISS register contains the 32-bit effective address of the instruction or data access, respectively, that caused the miss. Note that the 603e always loads a big-endian address into the DMISS register. These registers are 603e-specific.
Primary and secondary hash address registers (HASH1 and HASH2)	The HASH1 and HASH2 registers contain the primary and secondary PTEG addresses that correspond to the address causing a TLB miss. These PTEG addresses are automatically derived by the 603e by performing the primary and secondary hashing function on the contents of IMISS or DMISS, for an ITLB or DTLB miss exception, respectively. These registers are 603e-specific.
Instruction and data PTE compare registers (ICMP and DCMP)	The ICMP and DCMP registers contain the word to be compared with the first word of a PTE in the table search software routine to determine if a PTE contains the address translation for the instruction or data access. The contents of ICMP and DCMP are automatically derived by the 603e when a TLB miss exception occurs. These registers are 603e-specific.
Required physical address register (RPA)	The system software loads a TLB entry by loading the second word of the matching PTE entry into the RPA register and then executing the **tlbli** or **tlbld** instruction (for loading the ITLB or DTLB, respectively). This register is 603e-specific.

Note that the 603e contains other features that do not specifically control the 603e MMU but are implemented to increase performance and flexibility. These are:

- Complete set of shadow segment registers for the instruction MMU. These registers are invisible to the programming model, as described in Section 5.4.3, “TLB Description.”
- Temporary GPR0–GPR3. These registers are available as **r0**–**r3** when MSR[TGPR] is set. The 603e automatically sets MSR[TGPR] whenever one of the three TLB miss exceptions occurs, allowing these exception handlers to have four registers that are used as scratchpad space, without having to save or restore this part of the machine state that existed when the exception occurred. Note that MSR[TGPR] is restored to the value in SRR1 when the **rfi** instruction is executed. Refer to Section 5.5.2, “Implementation-Specific Table Search Operation,” for code examples that take advantage of these registers.

In addition, the 603e also automatically saves the values of CR[CR0] of the executing context to SRR1[0–3] whenever one of the three TLB miss exceptions occurs. Thus, the exception handler can set CR[CR0] bits and branch accordingly in the exception handler routine, without having to save the existing CR[CR0] bits. However, the exception handler must restore these bits to CR[CR0] before executing the **rfi** instruction. There are also four other bits saved in SRR1 whenever a TLB miss exception occurs that give information about whether the access was an instruction or data access; and if it was a data access, whether it was for a load or a store instruction. Also these bits give some information related to the protection attributes for the access, and which set in the TLB will be replaced when the next TLB entry is loaded. Refer to Section 5.5.2.1, “Resources for Table Search Operations,” for more information on these bits and their use.

5.2 Real Addressing Mode

If address translation is disabled (MSR[IR] = 0 or MSR[DR] = 0) for a particular access, the effective address is treated as the physical address and is passed directly to the memory subsystem as described in Chapter 7, “Memory Management,” in *The Programming Environments Manual*.

Note that the default WIMG bits (0b0011) cause data accesses to be considered cacheable (I = 0) and thus load and store accesses are weakly ordered. This is the case, even if the data cache is disabled in the HID0 register (as it is out of hard reset). If I/O devices require load and store accesses to occur in strict program order (strongly ordered), translation must be enabled so that the corresponding I bit can be set. Also, for instruction accesses, the default memory access mode bits (WIMG) are 0b0001. That is, instruction accesses are considered cacheable (I = 0), and the memory is guarded. Again, instruction cache accesses are considered cacheable even if the instruction cache is disabled in the HID0 register (as it is out of hard reset). The W and M bits have no effect on the instruction cache.

For information on the synchronization requirements for changes to MSR[IR] and MSR[DR], refer to “Synchronization Requirements for Special Registers and for Lookaside Buffers” in Chapter 2, “PowerPC Register Set,” in *The Programming Environments Manual.*

5.3 Block Address Translation

The block address translation (BAT) mechanism in the OEA provides a way to map ranges of effective addresses larger than a single page into contiguous areas of physical memory. Such areas can be used for data that is not subject to normal virtual memory handling (paging), such as a memory-mapped display buffer or an extremely large array of numerical data.

The software model for block address translation in the 603e is described in Chapter 7, “Memory Management,” in *The Programming Environments Manual* for 32-bit implementations.

Implementation Note—The 603e BAT registers are not initialized by the hardware after the power-up or reset sequence. Consequently, all valid bits in both instruction and data BAT areas must be cleared before setting any BAT area for the first time. This is true regardless of whether address translation is enabled. Also, software must avoid overlapping blocks while updating a BAT area or areas. Even if translation is disabled, multiple BAT area hits are treated as programming errors and can corrupt the BAT registers and produce unpredictable results.

5.4 Memory Segment Model

The 603e adheres to the memory segment model as defined in Chapter 7, “Memory Management,” in *The Programming Environments Manual* for 32-bit implementations. Memory in the PowerPC OEA is divided into 256-Mbyte segments. This segmented memory model provides a way to map 4-Kbyte pages of effective addresses to 4-Kbyte pages in physical memory (page address translation), while providing the programming flexibility afforded by a large virtual address space (52 bits).

The segment/page address translation mechanism may be superseded by the block address translation (BAT) mechanism described in Section 5.3, “Block Address Translation.” If not, the translation proceeds in the following two steps:

1. from effective address to the virtual address (which never exists as a specific entity but can be considered to be the concatenation of the virtual page number and the byte offset within a page), and
2. from virtual address to physical address.

This section highlights those areas of the memory segment model defined by the OEA that are specific to the 603e.

5.4.1 Page History Recording

Referenced (R) and changed (C) bits reside in each PTE to keep history information about the page. They are maintained by a combination of the 603e hardware and the table search software. The operating system uses this information to determine which areas of memory to write back to disk when new pages must be allocated in main memory. Referenced and changed recording is performed only for accesses made with page address translation and not for translations made with the BAT mechanism or for accesses that correspond to direct-store interface (T = 1) segments. Furthermore, R and C bits are maintained only for accesses made while address translation is enabled (MSR[IR] = 1 or MSR[DR] = 1).

In the 603e, the referenced and changed bits are updated as follows:

- For TLB hits, the C bit is updated according to Table 5-7.
- For TLB misses, when a table search operation is in progress to locate a PTE, the R and C bits are updated (set, if required) to reflect the status of the page based on this access.

Table 5-7. Table Search Operations to Update History Bits—TLB Hit Case

R and C Bits in TLB entry	Processor Action
00	Combination doesn't occur
01	Combination doesn't occur
10	Read: No special action Write: Table search operation required to update C. Causes a data TLB miss on store exception
11	No special action for read or write

Table 5-7 shows that the status of the C bit in the TLB entry (in the case of a TLB hit) is what causes the processor to update the C bit in the PTE (the R bit is assumed to be set in the page tables if there is a TLB hit). Therefore, when software clears the R and C bits in the page tables in memory, it must invalidate the TLB entries associated with the pages whose referenced and changed bits were cleared.

The 603e causes the R bit to be set for the execution of the **dcbt** or **dcbtst** instruction to that page (by causing a TLB miss exception to load the TLB entry in the case of a TLB miss). However, neither of these instructions cause the C bit to be set.

The update of the referenced and changed bits is performed by PowerPC processors as if address translation were disabled (real addressing mode translation). Additionally, these updates should be performed with single-beat read and byte write transactions on the bus.

5.4.1.1 Referenced Bit

The referenced (R) bit of a page is located in the PTE in the page table. Every time a page is referenced (with a read or write access) and the R bit is zero, the R bit is then set in the page table. The OEA specifies that the referenced bit may be set immediately, or the setting may be delayed until the memory access is determined to be successful. Because the reference to a page is what causes a PTE to be loaded into the TLB, the referenced bit in all 603e TLB entries is effectively always set. The processor never automatically clears the referenced bit.

The referenced bit is only a hint to the operating system about the activity of a page. At times, the referenced bit may be set although the access was not logically required by the program or even if the access was prevented by memory protection. Examples of this in PowerPC systems include the following:

- Fetching of instructions not subsequently executed
- Accesses generated by an **lswx** or **stswx** instruction with a zero length
- Accesses generated by a **stwcx.** instruction when no store is performed because a reservation does not exist
- Accesses that cause exceptions and are not completed

5.4.1.2 Changed Bit

The changed bit of a page is located both in the PTE in the page table and in the copy of the PTE loaded into the TLB (if a TLB is implemented, as in the 603e). Whenever a data store instruction is executed successfully, if the TLB search (for page address translation) results in a hit, the changed bit in the matching TLB entry is checked. If it is already set, the processor does not change the C bit. If the TLB changed bit is 0, it is set and a table search operation is performed to also set the C bit in the corresponding PTE in the page table. The 603e causes a data TLB miss on store exception for this case so that the software can perform the table search operation for setting the C bit. Refer to Section 5.5.2, “Implementation-Specific Table Search Operation,” for an example code sequence that handles these conditions.

The changed bit (in both the TLB and the PTE in the page tables) is set only when a store operation is allowed by the page memory protection mechanism and all conditional branches occurring earlier in the program have been resolved (such that the store is guaranteed to be in the execution path). Furthermore, the following conditions may cause the C bit to be set:

- The execution of an **stwcx.** instruction is allowed by the memory protection mechanism but a store operation is not performed because no reservation exists.
- The execution of an **stswx** instruction is allowed by the memory protection mechanism but a store operation is not performed because the specified length is zero.
- The store operation is not performed because an exception occurs before the store is performed.

Again, note that although the execution of the **dcbt** and **dcbtst** instructions may cause the R bit to be set, they never cause the C bit to be set.

5.4.1.3 Scenarios for Referenced and Changed Bit Recording

This section provides a summary of the model (defined by the OEA) that is used by PowerPC processors for maintaining the referenced and changed bits. In some scenarios, the bits are guaranteed to be set by the processor, in some scenarios, the architecture allows that the bits may be set (not absolutely required), and in some scenarios, the bits are guaranteed to not be set.

In implementations that do not maintain the R and C bits in hardware (such as the 603e), software assistance is required. For these processors, the information in this section still applies, except that the software performing the updates is constrained to the rules described (that is, must set bits shown as guaranteed to be set and must not set bits shown as guaranteed to not be set).

Table 5-8 defines a prioritized list of the R and C bit settings for all scenarios. The entries in the table are prioritized from top to bottom, such that a matching scenario occurring closer to the top of the table takes precedence over a matching scenario closer to the bottom of the table. For example, if an **stwcx.** instruction causes a protection violation and there is no reservation, the C bit is not altered, as shown for the protection violation case. Note that in the table, load operations include those generated by load instructions, by the **eciwx** instruction, and by the cache management instructions that are treated as a load with respect to address translation. Similarly, store operations include those operations generated by store instructions, by the **ecowx** instruction, and by the cache management instructions that are treated as a store with respect to address translation. In the columns for the 603e, the combination of the 603e itself and the software used to search the page tables (described in Section 5.5.2, "Implementation-Specific Table Search Operation") is assumed.

Table 5-8. Model for Guaranteed R and C Bit Settings

Priority	Scenario	R Bit Set		C Bit Set	
		OEA	603e	OEA	603e
1	No-execute protection violation	No	No	No	No
2	Page protection violation	Maybe	Yes	No	No
3	Out-of-order instruction fetch or load operation	Maybe	No	No	No
4	Out-of-order store operation for instructions that will cause no other kind of precise exception (in the absence of system-caused, imprecise, or floating-point assist exceptions)[1]	Maybe[2]	No	No	No
5	All other out-of-order store operations	Maybe[2]	No	Maybe[2]	No
6	Zero-length load (**lswx**)	Maybe	Yes	No	No
7	Zero-length store (**stswx**)	Maybe[2]	Yes	Maybe[2]	Yes
8	Store conditional (**stwcx.**) that does not store	Maybe[2]	Yes	Maybe[2]	Yes
9	In-order instruction fetch	Yes[3]	Yes	No	No
10	Load instruction or **eciwx**	Yes	Yes	No	No
11	Store instruction, **ecowx** or **dcbz** instruction	Yes	Yes	Yes	Yes
12	**dcbt**, **dcbtst**, **dcbst**, or **dcbf** instruction	Maybe	Yes	No	No
13	**icbi** instruction	Maybe[2]	No	No[2]	No
14	**dcbi** instruction	Maybe[2]	Yes	Maybe[2]	Yes

[1] The EC603e microprocessor does not support the floating-point unit.
[2] If C is set, R is guaranteed to also be set
[3]This includes the case in which the instruction was fetched out-of-order and R was not set (does not apply for 603e).

For more information, see “Page History Recording” in Chapter 7, “Memory Management,” of *The Programming Environments Manual*.

5.4.2 Page Memory Protection

The 603e implements page memory protection as it is defined in Chapter 7, “Memory Management,” in *The Programming Environments Manual*.

5.4.3 TLB Description

This section describes the hardware resources provided in the 603e to facilitate the page address translation process. Note that the hardware implementation of the MMU is not specified by the architecture, and while this description applies to the 603e, it does not necessarily apply to other PowerPC processors.

5.4.3.1 TLB Organization

Because the 603e has two MMUs (IMMU and DMMU) that operate in parallel, some of the MMU resources are shared, and some are actually duplicated (shadowed) in each MMU to maximize performance. Figure 5-7 shows the relationships between these resources within both the IMMU and DMMU and how the various portions of the effective address are used in the address translation process.

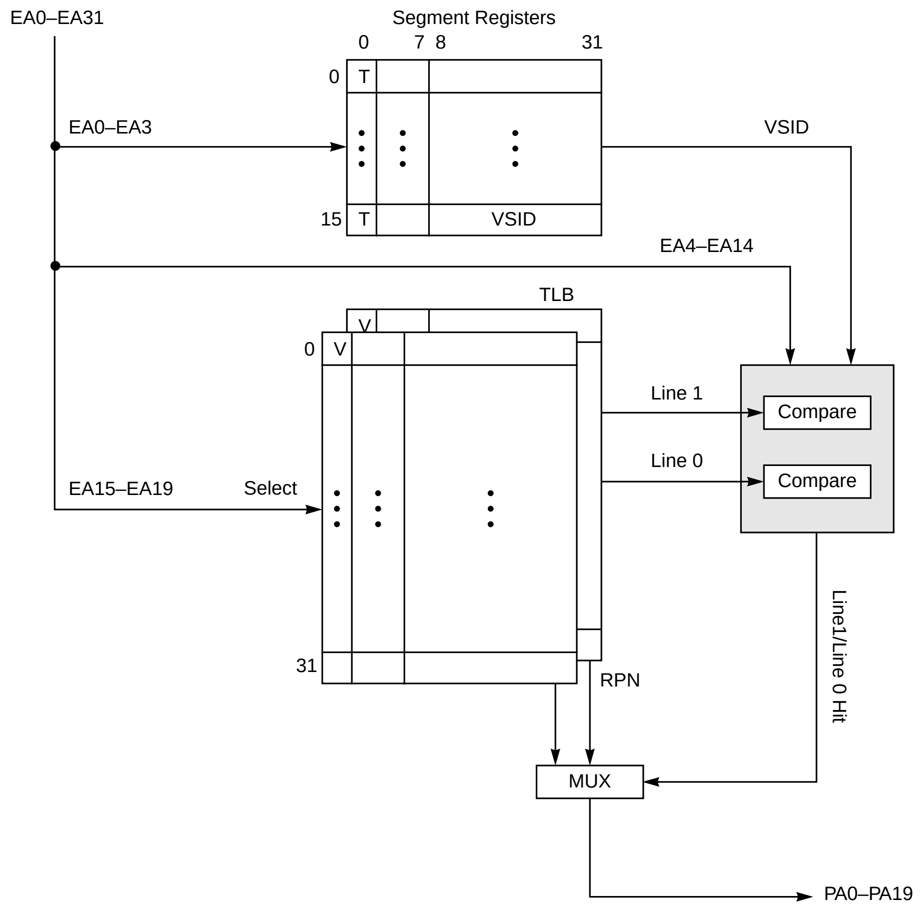


Figure 5-7. Segment Register and TLB Organization

While both MMUs can be accessed simultaneously (both sets of segment registers and TLBs can be accessed in the same clock), when there is an exception condition, only one exception is reported at a time. ITLB miss exceptions are reported when there are no more instructions to be dispatched or retired (the pipeline is empty), and DTLB miss conditions are reported when the load or store instruction is ready to be retired. Refer to Chapter 6, “Instruction Timing,” for more detailed information about the internal pipelines and the reporting of exceptions.

As TLB entries are on-chip copies of PTEs in the page tables in memory, they are similar in structure. TLB entries consist of two words; the high-order word contains the VSID and API fields of the high-order word of the PTE and the low-order word contains the RPN, the C bit, the WIMG bits and the PP bits (as in the low-order word of the PTE). In order to uniquely identify a TLB entry as the required PTE, the TLB also contains five more bits of the page index, EA10–EA14 (in addition to the API bits of the PTE).

When an instruction or data access occurs, the effective address is routed to the appropriate MMU. EA0–EA3 select one of the 16 segment registers and the remaining effective address bits and the virtual address from the segment register is passed to the TLB. EA15–EA19 then select two entries in the TLB; the valid bit is checked and EA10–EA14, the VSID, and API fields (EA4–EA9) for the access are then compared with the corresponding values in the TLB entries. If one of the entries hits, the PP bits are checked for a protection violation, and the C bit is checked. If these bits do not cause an exception, the RPN value is passed to the memory subsystem and the WIMG bits are then used as attributes for the access.

Although address translation is disabled on a reset condition, the valid bits of the BAT array and TLB entries are not automatically cleared. Thus TLB entries must be explicitly cleared by the system software (with the **tlbie** instruction) before the valid entries are loaded and address translation is enabled. Also, note that the segment registers do not have a valid bit, and so they should also be initialized before translation is enabled.

5.4.3.2 TLB Entry Invalidation

For the PowerPC processors, such as the 603e, that implement TLB structures to maintain on-chip copies of the PTEs that are resident in physical memory, the optional **tlbie** instruction provides a way to invalidate the TLB entries. Note that the execution of the **tlbie** instruction in the 603e invalidates four entries—both the ITLB entries indexed by EA15–EA19 and both the indexed entries of the DTLB.

The architecture allows **tlbie** to optionally enable a TLB invalidate signaling mechanism in hardware so that other processors also invalidate their resident copies of the matching PTE. The 603e does not signal the TLB invalidation to other processors nor does it perform any action when a TLB invalidation is performed by another processor.

The **tlbsync** instruction causes instruction execution to stop if the $\overline{\text{TLBISYNC}}$ signal is also asserted. If $\overline{\text{TLBISYNC}}$ is negated, instruction execution may continue or resume after the completion of a **tlbsync** instruction. Section 8.8.2, “TLBISYNC Input,” describes the TLB synchronization mechanism in further detail.

The **tlbia** instruction is not implemented on the 603e and when its opcode is encountered, an illegal instruction program exception is generated. To invalidate all entries of both TLBs, 32 **tlbie** instructions must be executed, incrementing the value in EA15–EA19 by one each time. See Chapter 8, “Instruction Set,” in *The Programming Environments Manual* for detailed information about the **tlbie** instruction.

5.4.4 Page Address Translation Summary

Figure 5-8 provides the detailed flow for the page address translation mechanism. The figure includes the checking of the N bit in the segment descriptor and then expands on the “TLB Hit” branch of Figure 5-6. The detailed flow for the “TLB Miss” branch of Figure 5-6 is described in Section 5.5.1, “Page Table Search Operation—Conceptual Flow.” Note that as in the case of block address translation, if the **dcbz** instruction is attempted to be executed either in write-through mode or as cache-inhibited (W = 1 or I = 1), the alignment exception is generated. The checking of memory protection violation conditions for page address translation is described in Chapter 7, “Memory Management,” in *The Programming Environments Manual* for 32-bit implementations.

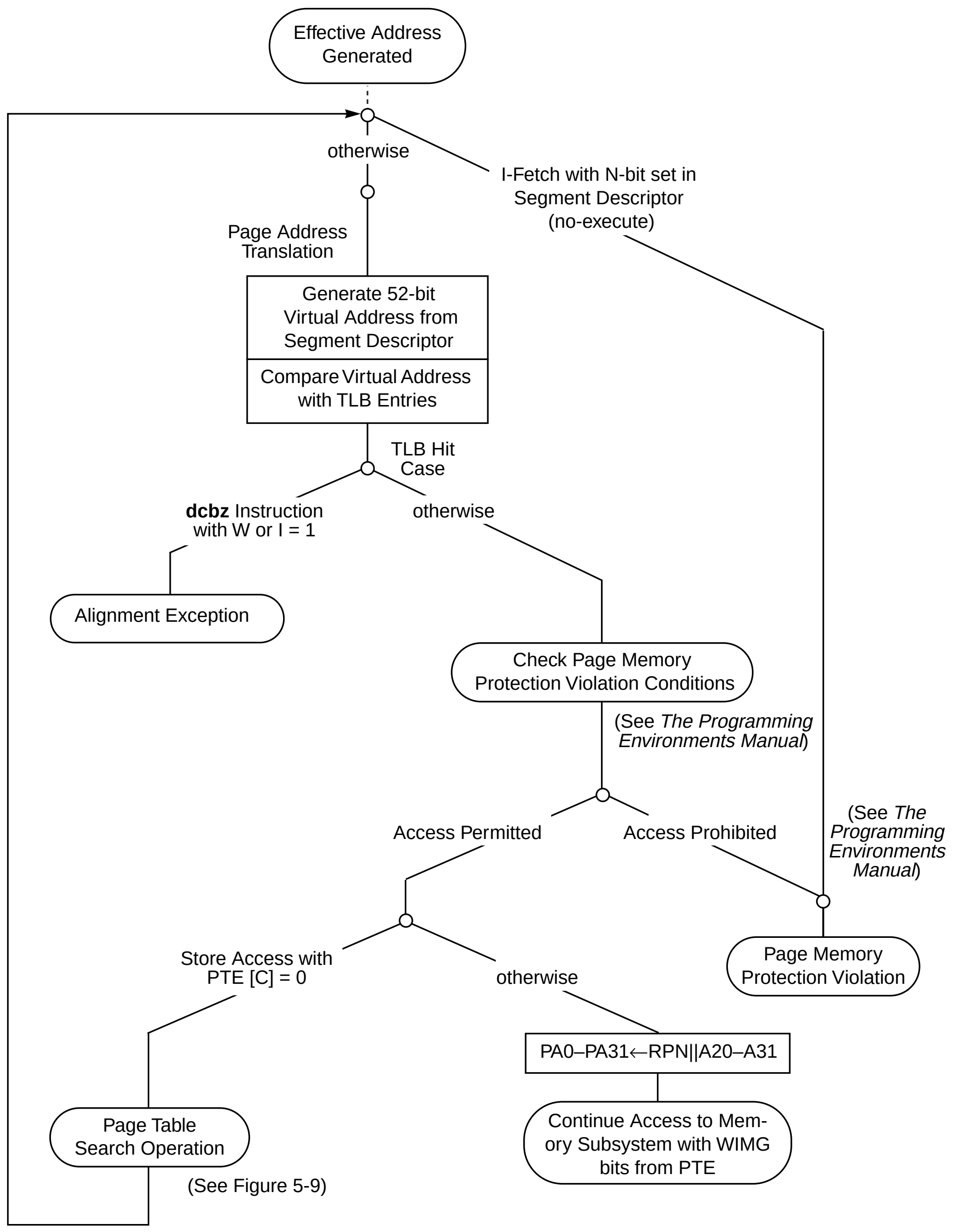

Effective Address Generated
otherwise
I-Fetch with N-bit set in Segment Descriptor (no-execute)
Page Address Translation
Generate 52-bit Virtual Address from Segment Descriptor
Compare Virtual Address with TLB Entries
TLB Hit Case
dcbz Instruction with W or I = 1
otherwise
Alignment Exception
Check Page Memory Protection Violation Conditions
(See *The Programming Environments Manual*)
Access Permitted
Access Prohibited
(See *The Programming Environments Manual*)
Store Access with PTE [C] = 0
otherwise
Page Memory Protection Violation
PA0–PA31←RPN||A20–A31
Page Table Search Operation
Continue Access to Memory Subsystem with WIMG bits from PTE
(See Figure 5-9)

Figure 5-8. Page Address Translation Flow for 32-Bit Implementations—TLB Hit

5.5 Page Table Search Operation

As stated earlier, the operating system must synthesize the table search algorithm for setting up the tables. In the case of the 603e, the TLB miss exception handlers also use this algorithm (with the assistance of some hardware-generated values) to load TLB entries when TLB misses occur as described in Section 5.5.2, "Implementation-Specific Table Search Operation."

5.5.1 Page Table Search Operation—Conceptual Flow

The table search process for a PowerPC processor varies slightly for 64- and 32-bit implementations. The main differences are the address ranges and PTE formats specified. An outline of the page table search process performed by a 32-bit implementation (such as the 603e) is as follows:

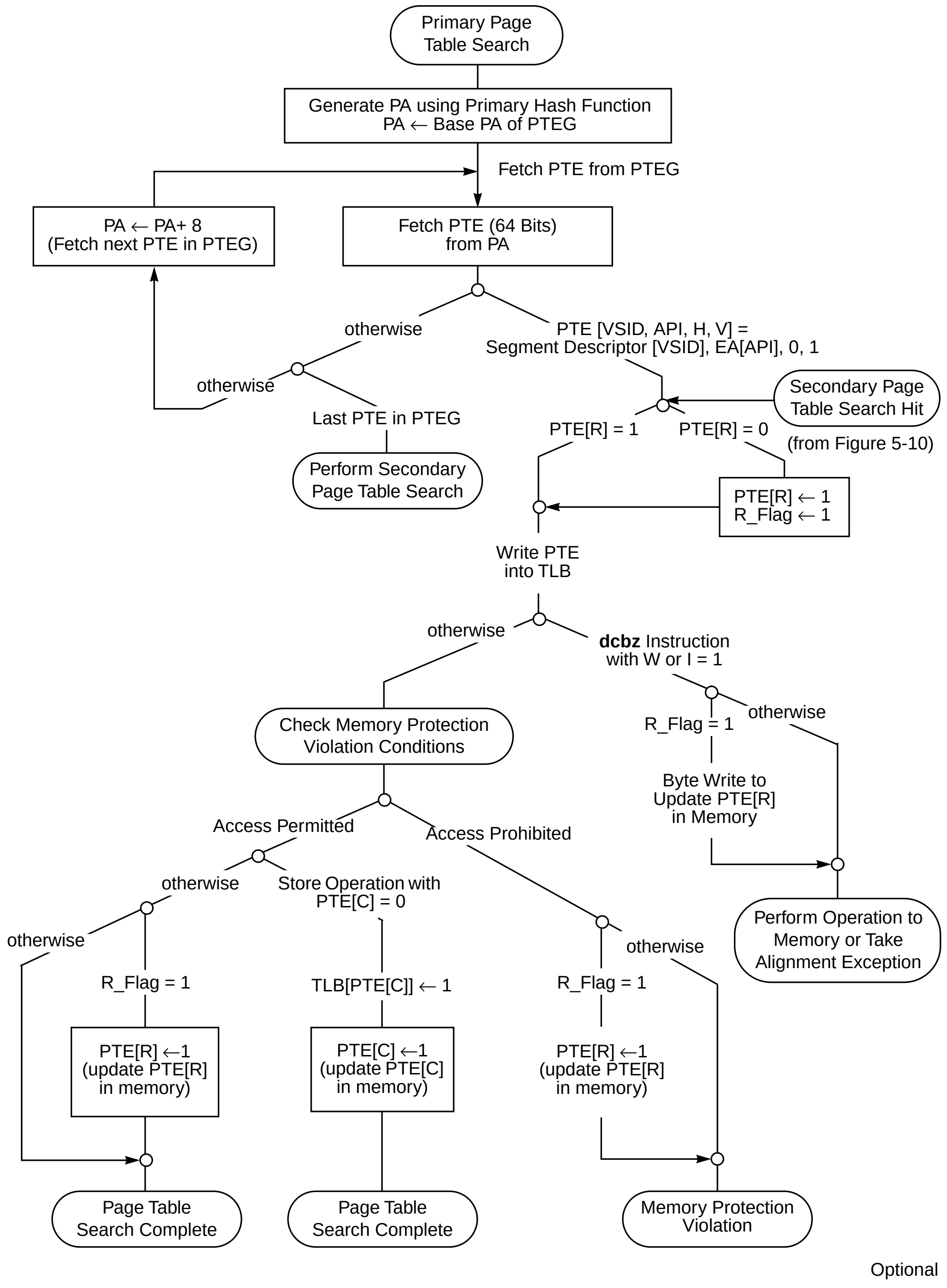
1. The 32-bit physical address of the primary PTEG is generated as described in Chapter 7, "Memory Management," in *The Programming Environments Manual* for 32-bit implementations.
2. The first PTE (PTE0) in the primary PTEG is read from memory. PTE reads should occur with an implied WIM memory/cache mode control bit setting of 0b001. Therefore, they are considered cacheable and burst in from memory and placed in the cache.
3. The PTE in the selected PTEG is tested for a match with the virtual page number (VPN) of the access. The VPN is the VSID concatenated with the page index field of the virtual address. For a match to occur, the following must be true:
   — PTE[H] = 0
   — PTE[V] = 1
   — PTE[VSID] = VA[0–23]
   — PTE[API] = VA[24–29]
4. If a match is not found, step 3 is repeated for each of the other seven PTEs in the primary PTEG. If a match is found, the table search process continues as described in step 8. If a match is not found within the eight PTEs of the primary PTEG, the address of the secondary PTEG is generated.
5. The first PTE (PTE0) in the secondary PTEG is read from memory. Again, because PTE reads typically have a WIM bit combination of 0b001, an entire cache line is burst into the on-chip cache.
6. The PTE in the selected secondary PTEG is tested for a match with the virtual page number (VPN) of the access. For a match to occur, the following must be true:
   — PTE[H] = 1
   — PTE[V] = 1
   — PTE[VSID] = VA[0–23]
   — PTE[API] = VA[24–29]

7. If a match is not found, step 6 is repeated for each of the other seven PTEs in the secondary PTEG.
8. If a match is found, the PTE is written into the on-chip TLB (if implemented, as in the 603e) and the R bit is updated in the PTE in memory (if necessary). If there is no memory protection violation, the C bit is also updated in memory and the table search is complete.
9. If a match is not found within the eight PTEs of the secondary PTEG, the search fails, and a page fault exception condition occurs (either an ISI exception or a DSI exception). Note that the software routines that implement this algorithm for the 603e must synthesize this condition by appropriately setting the bits in SRR1 (or DSISR) and branching to the ISI or DSI handler routine.

Reads from memory for table search operations should be performed as global (but not exclusive), cacheable operations, and can be loaded into the on-chip cache.

Figure 5-9 and Figure 5-10 provide conceptual flow diagrams of primary and secondary page table search operations, respectively as described in the OEA for 32-bit processors. Recall that the architecture allows for implementations to perform the page table search operations automatically (in hardware) or software assist may be required, as is the case with the 603e. Also, the elements in the figure that apply to TLBs are shown as optional because TLBs are not required by the architecture.

Figure 5-9 shows the case of a **dcbz** instruction that is executed with W = 1 or I = 1, and that the R bit may be updated in memory (if required) before the operation is performed or the alignment exception occurs. The R bit may also be updated in the case of a memory protection violation.



Primary Page Table Search
Generate PA using Primary Hash Function
PA ← Base PA of PTEG
Fetch PTE from PTEG
PA ← PA+ 8
(Fetch next PTE in PTEG)
Fetch PTE (64 Bits) from PA
otherwise
PTE [VSID, API, H, V] = Segment Descriptor [VSID], EA[API], 0, 1
otherwise
Last PTE in PTEG
Perform Secondary Page Table Search
Secondary Page Table Search Hit
(from Figure 5-10)
PTE[R] = 1
PTE[R] = 0
PTE[R] ← 1
R_Flag ← 1
Write PTE into TLB
otherwise
dcbz Instruction with W or I = 1
R_Flag = 1
otherwise
Byte Write to Update PTE[R] in Memory
Check Memory Protection Violation Conditions
Perform Operation to Memory or Take Alignment Exception
Access Permitted
Access Prohibited
otherwise
Store Operation with PTE[C] = 0
otherwise
R_Flag = 1
TLB[PTE[C]] ← 1
R_Flag = 1
otherwise
PTE[R] ←1 (update PTE[R] in memory)
PTE[C] ←1 (update PTE[C] in memory)
PTE[R] ←1 (update PTE[R] in memory)
Page Table Search Complete
Page Table Search Complete
Memory Protection Violation
Optional

Figure 5-9. Primary Page Table Search—Conceptual Flow

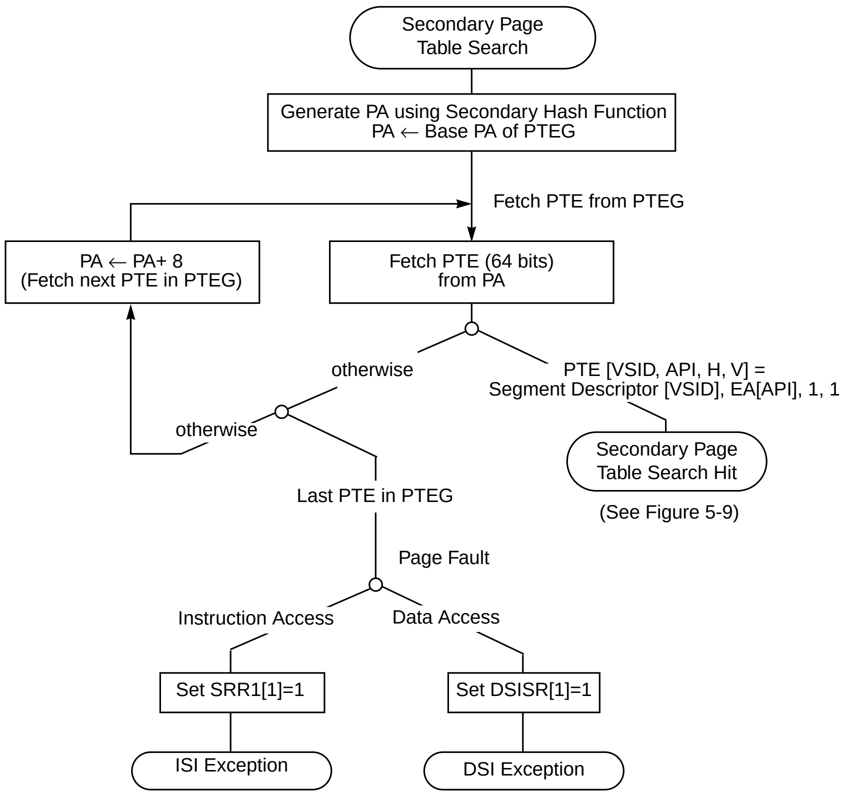


Figure 5-10. Secondary Page Table Search Flow—Conceptual Flow

5.5.2 Implementation-Specific Table Search Operation

The 603e has a set of implementation-specific registers, exceptions, and instructions that facilitate very efficient software searching of the page tables in memory. This section describes those resources and provides three example code sequences that can be used in a 603e system for an efficient search of the translation tables in software. These three code sequences can be used as handlers for the three exceptions requiring access to the PTEs in the page tables in memory—instruction TLB miss, data TLB miss on load, and data TLB miss on store exceptions.

5.5.2.1 Resources for Table Search Operations

In addition to setting up the translation page tables in memory, the system software must assist the processor in loading PTEs into the on-chip TLBs. When a required TLB entry is not found in the appropriate TLB, the processor vectors to one of the three TLB miss exception handlers so that the software can perform a table search operation and load the TLB. When this occurs, the processor automatically saves information about the access and the executing context. Table 5-9 provides a summary of the implementation-specific exceptions, registers, and instructions, that can be used by the TLB miss exception handler software in 603e systems. Refer to Chapter 4, “Exceptions,” for more information about exception processing.

Table 5-9. Implementation-Specific Resources for Table Search Operations

Resource	Name	Description
Exceptions	Instruction TLB miss exception (vector offset 0x1000)	No matching entry found in ITLB
	Data TLB miss on load exception (vector offset 0x1100)	No matching entry found in DTLB for a load data access
	Data TLB miss on store exception—also caused when changed bit must be updated (vector offset 0x1200)	No matching entry found in DTLB for a store data access or matching DLTB entry has C = 0 and access is a store.
Registers	IMISS and DMISS	When a TLB miss exception occurs, the IMISS or DMISS register contains the 32-bit effective address of the instruction or data access that caused the miss exception.
	ICMP and DCMP	The ICMP and DCMP registers contain the word to be compared with the first word of a PTE in the table search software routine to determine if a PTE contains the address translation for the instruction or data access. The contents of ICMP and DCMP are automatically derived by the 603e when a TLB miss exception occurs.
	HASH1 and HASH2	The HASH1 and HASH2 registers contain the primary and secondary PTEG addresses that correspond to the address causing a TLB miss. These PTEG addresses are automatically derived by the 603e by performing the primary and secondary hashing function on the contents of IMISS or DMISS, for an ITLB or DTLB miss exception, respectively
	RPA	The system software loads a TLB entry by loading the second word of the matching PTE entry into the RPA register and then executing the **tlbli** or **tlbld** instruction (for loading the ITLB or DTLB, respectively).

Table 5-9. Implementation-Specific Resources for Table Search Operations

Resource	Name	Description
Instructions	**tlbli r**B	Loads the contents of the ICMP and RPA registers into the ITLB entry selected by <ea> and SRR1[WAY]
	tlbld rB	Loads the contents of the DCMP and RPA registers into the DTLB entry selected by <ea> and SRR1[WAY]

In addition, the 603e contains the following features that do not specifically control the 603e MMU but that are implemented to increase performance and flexibility in the software table search routines whenever one of the three TLB miss exceptions occurs:

- Temporary GPR0–GPR3. These registers are available as **r0**–**r3** when MSR[TGPR] is set. The 603e automatically sets MSR[TGPR] for these cases, allowing these exception handlers to have four registers that are used as scratchpad space, without having to save or restore this part of the machine state that existed when the exception occurred. Note that MSR[TGPR] is cleared when the **rfi** instruction is executed because the old MSR value (with MSR[TGPR] = 0) saved in SRR1 is restored. Refer to Section 5.5.2.2, "Software Table Search Operation," for code examples that take advantage of these registers.
- The 603e also automatically saves the values of CR[CR0] of the executing context to SRR1[0–3]. Thus, the exception handler can set CR[CR0] bits and branch accordingly in the exception handler routine, without having to save the existing CR[CR0] bits. However, the exception handler must restore these bits to CR[CR0] before executing the **rfi** instruction.
- Also saved in SRR1 are two bits identifying the type of miss (SRR1[D/I] identifies instruction or data, and SRR1[L/S] identifies a load or store). Additionally, SRR1[WAY] identifies the associativity class of the TLB entry selected for replacement by the LRU algorithm. The software can change this value, effectively overriding the replacement algorithm. Finally, the SRR1[KEY] bit is used by the table search software to determine if there is a protection violation associated with the access (useful on data write misses for determining if the C bit should be updated in the table). Table 5-10 summarizes the SRR1 bits updated whenever one of the three TLB miss exceptions occurs.

Table 5-10. Implementation-Specific SRR1 Bits

Bit Number	Name	Function
0–3	CRF0	Condition register field 0 bits
12	KEY	Key for TLB miss (either Ks or Kp from segment register, depending on whether the access is a user or supervisor access)
13	D/I	Set if instruction TLB miss
14	WAY	Next TLB set to be replaced (set per LRU)
15	S/L	Set if data TLB miss was for a load instruction

The key bit saved in SRR1 is derived as shown in Figure 5-11.

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

Figure 5-11. Derivation of Key Bit for SRR1

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

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

The DMISS and IMISS registers have the same format as shown in Figure 5-12. They are loaded automatically upon a data or instruction TLB miss. The DMISS and IMISS contain the effective page address of the access which caused the TLB miss exception. The contents are used by the processor when calculating the values of HASH1 and HASH2, and by the **tlbld** and **tlbli** instructions when loading a new TLB entry. Note that the 603e always loads a big-endian address into the DMISS register. These registers are read-only to the software.

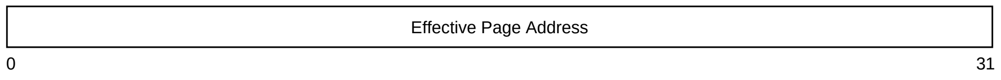


Figure 5-12. DMISS and IMISS Registers

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

The DCMP and ICMP registers are shown in Figure 5-13. These registers contain the first word in the required PTE. The contents are constructed automatically from the contents of the segment registers and the effective address (DMISS or IMISS) when a TLB miss exception occurs. Each PTE read from the tables in memory during the table search process should be compared with this value to determine whether or not the PTE is a match. Upon execution of a **tlbld** or **tlbli** instruction, the contents of the DCMP or ICMP register is loaded into the first word of the selected TLB entry.

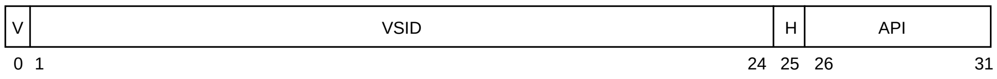


Figure 5-13. DCMP and ICMP Registers

Table 5-11 describes the bit settings for the DCMP and ICMP registers.

Table 5-11. DCMP and ICMP Bit Settings

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

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

HASH1 and HASH2 contain the physical addresses of the primary and secondary PTEGs for the access that caused the TLB miss exception. Only bits 7–25 differ between them. For convenience, the processor automatically constructs the full physical address by routing bits 0–6 of SDR1 into HASH1 and HASH2 and clearing the lower six bits. These registers are read-only and are constructed from the contents of the DMISS or IMISS register. The format for the HASH1 and HASH2 registers is shown in Figure 5-14.

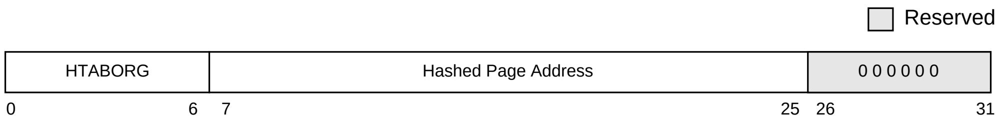


Figure 5-14. HASH1 and HASH2 Registers

Table 5-12 describes the bit settings of the HASH1 and HASH2 registers.

Table 5-12. HASH1 and HASH2 Bit Settings

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

5.5.2.1.4 Required Physical Address Register (RPA)

The RPA is shown in Figure 5-15. During a page table search operation, the software must load the RPA with the second word of the correct PTE. When the **tlbld** or **tlbli** instruction is executed, data from the IMISS and ICMP (or DMISS and DCMP) and the RPA registers is merged and loaded into the selected TLB entry. The TLB entry is selected by the effective address of the access (loaded by the table search software from the DMISS or IMISS register) and the SRR1[WAY] bit.

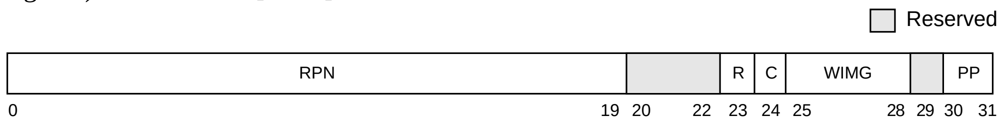


Figure 5-15. Required Physical Address (RPA) Register

Table 5-13 describes the bit settings of the RPA register.

Table 5-13. RPA Bit Settings

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

5.5.2.2 Software Table Search Operation

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

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

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

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

This section provides a flow diagram outlining some example software that can be used to handle the three TLB miss exceptions, as well as some example assembly language that implements that flow.

5.5.2.2.1 Flow for Example Exception Handlers

Figure 5-16 shows the flow for the example TLB miss exception handlers. The flow shown is common for the three exception handlers, except that the IMISS and ICMP registers are used for the instruction TLB miss exception while the DMISS and DCMP registers are used for the two data TLB miss exceptions. Also, for the cases of store instructions that cause either a TLB miss or require a table search operation to update the C bit, the flow shows that the C bit is set in both the TLB entry and the PTE in memory. Finally, in the case of a page fault (no PTE found in the table search operation), the setup for the ISI or DSI exception is slightly different for these two cases.

Figure 5-17 shows the flow for checking the R and C bits and setting them appropriately, Figure 5-18 shows the flow for synthesizing a page fault exception when no PTE is found. Figure 5-19 shows the flow for managing the cases of a TLB miss on an instruction access to guarded memory, and a TLB miss when C = 0 and a protection violation exists. The set up for these protection violation exceptions is very similar to that of page fault conditions (as shown in Figure 5-18) except that different bits in SRR1 (and DSISR) are set.

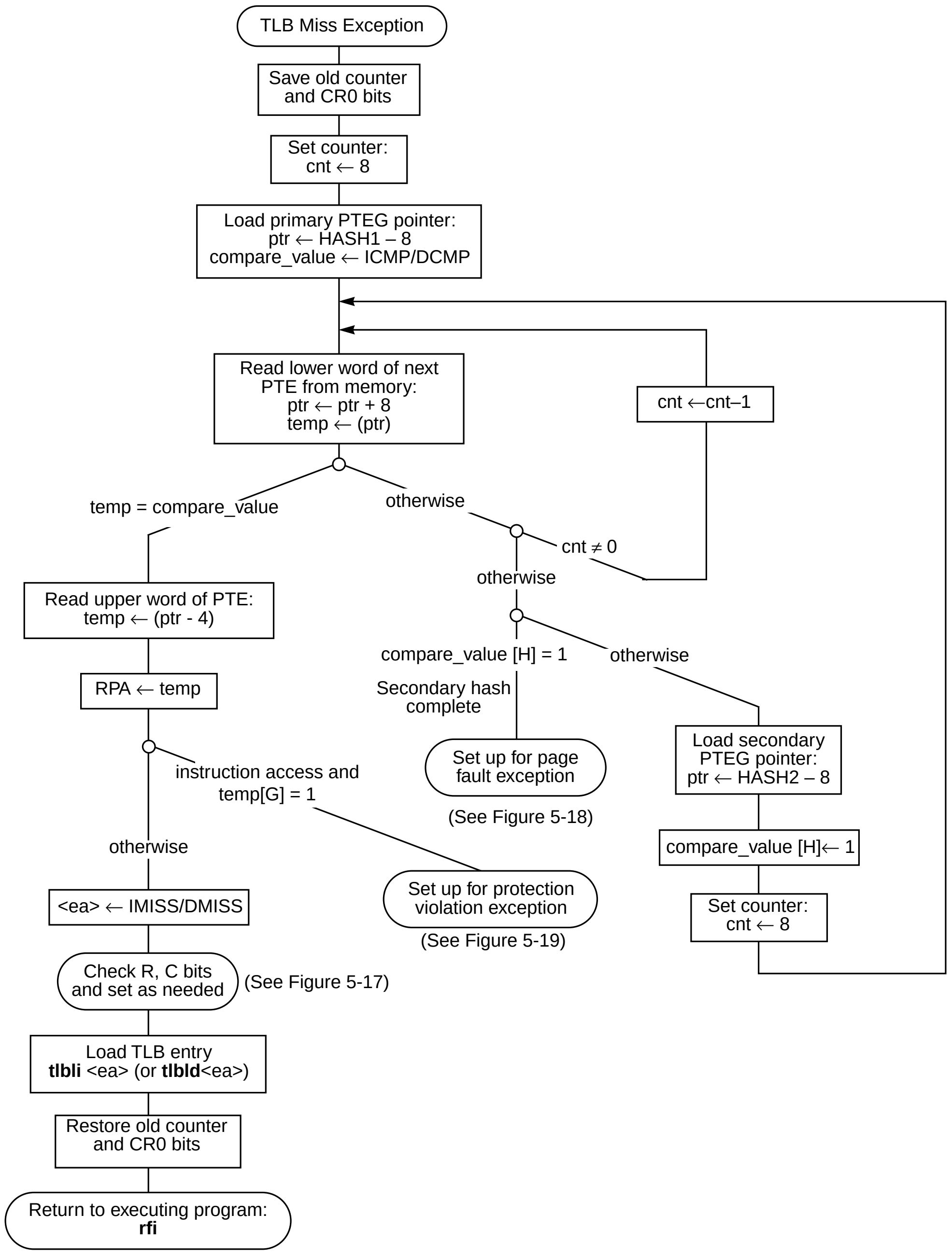
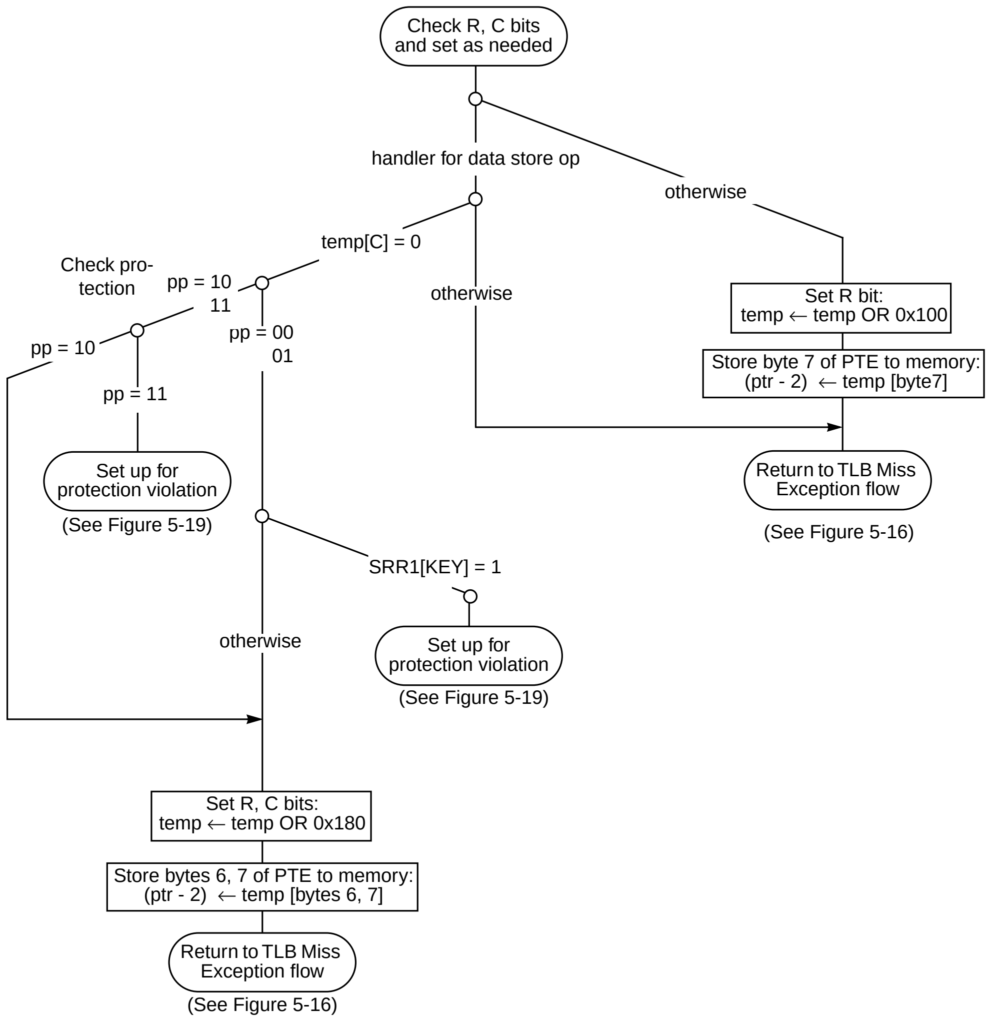

TLB Miss Exception
Save old counter and CR0 bits
Set counter: cnt ← 8
Load primary PTEG pointer: ptr ← HASH1 – 8 compare_value ← ICMP/DCMP
Read lower word of next PTE from memory: ptr ← ptr + 8 temp ← (ptr)
cnt ←cnt–1
temp = compare_value
otherwise
cnt ≠ 0
otherwise
Read upper word of PTE: temp ← (ptr - 4)
compare_value [H] = 1
otherwise
RPA ← temp
Secondary hash complete
Load secondary PTEG pointer: ptr ← HASH2 – 8
instruction access and temp[G] = 1
Set up for page fault exception
(See Figure 5-18)
compare_value [H]← 1
otherwise
Set up for protection violation exception
(See Figure 5-19)
Set counter: cnt ← 8
<ea> ← IMISS/DMISS
Check R, C bits and set as needed
(See Figure 5-17)
Load TLB entry tlbli <ea> (or tlbld<ea>)
Restore old counter and CR0 bits
Return to executing program: rfi

Figure 5-16. Flow for Example Software Table Search Operation



Check R, C bits and set as needed
handler for data store op
otherwise
temp[C] = 0
otherwise
Check protection
pp = 10 11
pp = 00 01
pp = 10
pp = 11
Set up for protection violation
(See Figure 5-19)
Set R bit: temp ← temp OR 0x100
Store byte 7 of PTE to memory: (ptr - 2) ← temp [byte7]
Return to TLB Miss Exception flow
(See Figure 5-16)
SRR1[KEY] = 1
otherwise
Set up for protection violation
(See Figure 5-19)
Set R, C bits: temp ← temp OR 0x180
Store bytes 6, 7 of PTE to memory: (ptr - 2) ← temp [bytes 6, 7]
Return to TLB Miss Exception flow
(See Figure 5-16)

Figure 5-17. Check and Set R and C Bit Flow

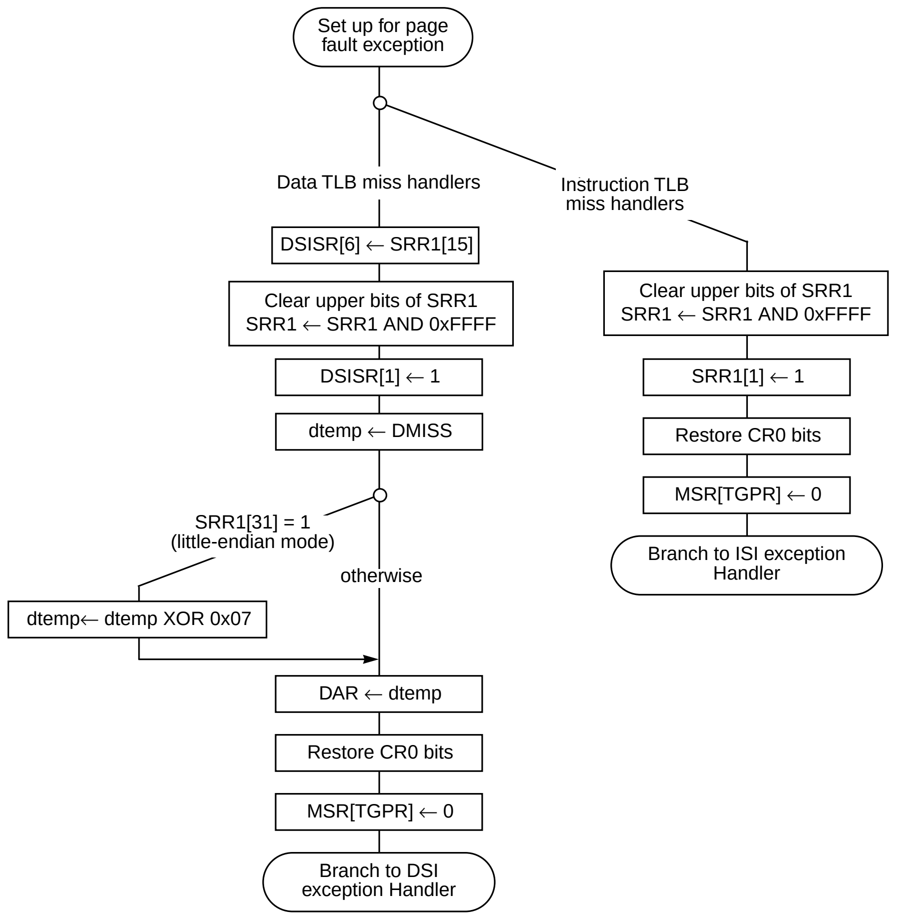

Set up for page fault exception
Data TLB miss handlers
Instruction TLB miss handlers
DSISR[6] ← SRR1[15]
Clear upper bits of SRR1 SRR1 ← SRR1 AND 0xFFFF
DSISR[1] ← 1
dtemp ← DMISS
SRR1[31] = 1 (little-endian mode)
otherwise
dtemp← dtemp XOR 0x07
DAR ← dtemp
Restore CR0 bits
MSR[TGPR] ← 0
Branch to DSI exception Handler
Clear upper bits of SRR1 SRR1 ← SRR1 AND 0xFFFF
SRR1[1] ← 1
Restore CR0 bits
MSR[TGPR] ← 0
Branch to ISI exception Handler

Figure 5-18. Page Fault Setup Flow

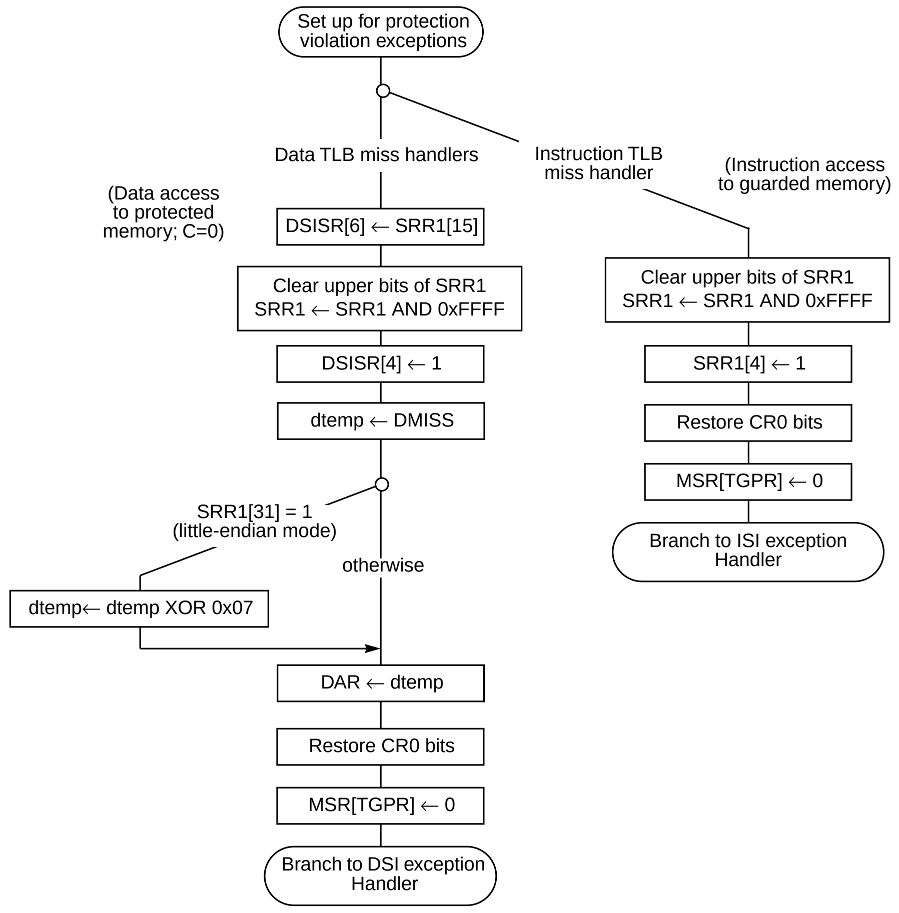


Figure 5-19. Setup for Protection Violation Exceptions

5.5.2.2.2 Code for Example Exception Handlers

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

```
# TLB software load for 603e
#
# New Instructions:
#           tlbld                   - write the dtlb with the pte in rpa reg
#           tlbli                   - write the itlb with the pte in rpa reg
# New SPRs
#           dmiss                   - address of dstream miss
#           imiss                   - address of istream miss
#           hash1                   - address primary hash PTEG address
#           hash2                   - returns secondary hash PTEG address
#           iCmp                    - returns the primary istream compare value
#           dCmp                    - returns the primary dstream compare value
#           rpa                     - the second word of pte used by tlblx
#
# gpr r0..r3 are shadowed
#
# there are three flows.
#           tlbDataMiss- tlb miss on data load
#           tlbCeq0     - tlb miss on data store or store with tlb change bit == 0
#           tlbInstrMiss- tlb miss on instruction fetch
#+
# place labels for rel branches
#-
#.machine PPC_603e
.set        r0, 0
.set        r1, 1
.set        r2, 2
.set        r3, 3
.set        dMiss, 1010
.set        dCmp,   1011
.set        hash1, 1012
.set        hash2, 1013
.set        iMiss, 1014
.set        iCmp,   1015
.set        rpa, 1010
.set        c0, 0
.set        dar, 19
.set        dsisr, 18
.set        srr0, 26
.set        srr1, 27
.
.csect tlbmiss[PR]
vec0:
.globl vec0
```

```
.org        vec0+0x300
vec300:
.org        vec0+0x400
vec400:
#+
# Instruction TB miss flow
# Entry:
#           Vec = 1000
#           srr0        -> address of instruction that missed
#           srr1        -> 0:3=cr0 4=lru way bit 16:31 = saved MSR
#           msr<tgpr> -> 1
#           iMiss       -> ea that missed
#           iCmp        -> the compare value for the va that missed
#           hash1       -> pointer to first hash pteg
#           hash2       -> pointer to second hash pteg
#
# Register usage:
#           r0 is saved counter
#           r1 is junk
#           r2 is pointer to pteg
#           r3 is current compare value

.org        vec0+0x1000

tlbInstrMiss:
            mfspr    r2, hash1    # get first pointer
             addi    r1, 0, 8     # load 8 for counter
            mfctr    r0           # save counter
            mfspr    r3, iCmp     # get first compare value
             addi    r2, r2, -8   # pre dec the pointer
im0:        mtctr    r1           # load counter
im1:        lwzu     r1, 8(r2)    # get next pte
             cmp     c0, r1, r3   # see if found pte
            bdneq    im1          # dec count br if cmp ne and if count not zero
            bne      instrSecHash# if not found set up second hash or exit
            l        r1, +4(r2)   # load tlb entry lower-word
            andi.    r3, r1, 8    # check G-bit
            bne      doISIp       # if guarded, take an ISI
            mtctr    r0           # restore counter
            mfspr    r0, iMiss    # get the miss address for the tlbli
            mfspr    r3, srr1     # get the saved cr0 bits
            mtcrf    0x80, r3     # restore CR0
            mtspr    rpa, r1      # set the pte
             ori     r1, r1, 0x100# set reference bit
            srw      r1, r1, 8    # get byte 7 of pte
            tlbli    r0           # load the itlb
            stb      r1, +6(r2)   # update page table
            rfi                   # return to executing program
#+
# Register usage:
```

```
#               r0 is saved counter
#               r1 is junk
#               r2 is pointer to pteg
#               r3 is current compare value
#-
instrSecHash:
                andi.       r1, r3, 0x0040# see if we have done second hash
                bne         doISI       # if so, go to ISI exception
                mfspr       r2, hash2   # get the second pointer
                ori         r3, r3, 0x0040# change the compare value
                addi        r1, 0, 8    # load 8 for counter
                addi        r2, r2, -8  # pre dec for update on load
                b           im0         # try second hash
#+
# entry Not Found: synthesize an ISI exception
# guarded memory protection violation: synthesize an ISI exception
# Entry:
#               r0 is saved counter
#               r1 is junk
#               r2 is pointer to pteg
#               r3 is current compare value
#
doISIp:
                mfspr       r3, srr1    # get srr1
                andi.       r2,r3,0xffff # clean upper srr1
                addis       r2, r2, 0x0800# or in srr<4> = 1 to flag prot violation
                b           isi1:

doISI:
                mfspr       r3, srr1    # get srr1
                andi.       r2, r3, 0xffff# clean srr1
                addis       r2, r2, 0x4000# or in srr1<1> = 1 to flag pte not found
                mtctr       r0          # restore counter
isi1            mtspr       srr1, r2    # set srr1
                mfmsr       r0          # get msr
                xori        r0, r0, 0x8000# flip the msr<tgpr> bit
                mtcrf       0x80, r3    # restore CR0
                mtmsr       r0          # flip back to the native gprs
                b           vec400      # go to instr. access exception
#

#+
# Data TLB miss flow
# Entry:
#               Vec = 1100
#               srr0          -> address of instruction that caused data tlb miss
#               srr1          -> 0:3=cr0 4=lru way bit 5=1 if store 16:31 = saved MSR
#               msr<tgpr> -> 1
#               dMiss         -> ea that missed
#               dCmp          -> the compare value for the va that missed
#               hash1         -> pointer to first hash pteg
#               hash2           -> pointer to second hash pteg
#
# Register usage:
#               r0 is saved counter
#               r1 is junk
#               r2 is pointer to pteg
#               r3 is current compare value
#-
.csect          tlbmiss[PR]
.org            vec0+0x1100

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

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

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

tlbCeq0:
                mfspr       r2, hash1    # get first pointer
                 addi       r1, 0, 8     # load 8 for counter
                mfctr       r0           # save counter
                mfspr       r3, dCmp     # get first compare value
                 addi       r2, r2, -8   # pre dec the pointer
ceq0:           mtctr       r1           # load counter
ceq1:           lwzu        r1, 8(r2)    # get next pte
                cmp         c0, r1, r3   # see if found pte
                bdneq       ceq1         # dec count br if cmp ne and if count not zero
                bne         cEq0SecHash# if not found set up second hash or exit
                l           r1, +4(r2)   # load tlb entry lower-word
                andi.       r3,r1,0x80   # check the C-bit
                beq         cEq0ChkProt# if (C==0) go check protection modes
ceq2:            mtctr      r0           # restore counter
                mfspr       r0, dMiss    # get the miss address for the tlbld
                mfspr       r3, srr1     # get the saved cr0 bits
                mtcrf       0x80, r3     # restore CR0
                mtspr       rpa, r1      # set the pte
                tlbld       r0           # load the dtlb
                rfi                      # return to executing program
#+
# Register usage:
#               r0 is saved counter
#               r1 is junk
#               r2 is pointer to pteg
#               r3 is current compare value
#-
cEq0SecHash:
                andi.       r1, r3, 0x0040# see if we have done second hash
```

```
        bne     doDSI       # if so, go to DSI exception
        mfspr   r2, hash2   # get the second pointer
        ori     r3, r3, 0x0040# change the compare value
        addi    r1, 0, 8    # load 8 for counter
        addi    r2, r2, -8  # pre dec for update on load
        b       ceq0        # try second hash
#+
# entry found and PTE(c-bit==0):
# (check protection before setting PTE(c-bit)
# Register usage:
#       r0 is saved counter
#       r1 is PTE entry
#       r2 is pointer to pteg
#       r3 is trashed
#-
cEq0ChkProt:
        rlwinm. r3,r1,30,0,1# test PP
        bge-    chk0        # if (PP==00 or PP==01) goto chk0:
        andi.   r3,r1,1     # test PP[0]
        beq+    chk2        # return if PP[0]==0
        b       doDSIp      # else DSIp

chk0:   mfspr   r3,srr1     # get old msr
        andis.  r3,r3,0x0008# test the KEY bit (SRR0-bit 12)
        beq     chk2        # if (KEY==0) goto chk2:
        b       doDSIp      # else DSIp
chk2:   ori     r1, r1, 0x180# set reference and change bit
        sth     r1, -2(r2)  # update page table
        b       ceq2        # and back we go
#
#+
# entry Not Found: synthesize a DSI exception
# Entry:
#       r0 is saved counter
#       r1 is junk
#       r2 is pointer to pteg
#       r3 is current compare value
#
doDSI:
        mfspr   r3, srr1    # get srr1
        rlwinm  r1, r3, 9,6,6# get srr1<flag> to bit 6 for load/store, zero rest
        addis   r1, r1, 0x4000# or in dsisr<1> = 1 to flag pte not found
        b       dsi1:
doDSIp:
        mfspr   r3, srr1    # get srr1
        rlwinm  r1, r3, 9,6,6# get srr1<flag> to bit 6 for load/store, zero rest
        addis   r1, r1, 0x0800# or in dsisr<4> = 1 to flag prot violation
dsi1:   mtctr   r0          # restore counter
        andi.   r2, r3, 0xffff# clear upper bits of srr1
        mtspr   srr1, r2    # set srr1
        mtspr   dsisr, r1   # load the dsisr
        mfspr     r1, dMiss    # get miss address
        rlwinm.   r2,r2,0,31,31# test LE bit
        bne       dsi2:        # if little endian then:
        xor       r1,r1,0x07   #   de-mung the data address
dsi2:   mtspr     dar, r1      # put in dar
        mfmsr     r0           # get msr
        xoris     r0, r0, 0x2  # flip the msr<tgpr> bit
        mtcrf     0x80, r3     # restore CR0
        mtmsr     r0           # flip back to the native gprs
        b         vec300       # branch to DSI exception
```

5.5.3 Page Table Updates

When TLBs are implemented (as in the 603e) they are defined as noncoherent caches of the page tables. TLB entries must be flushed explicitly with the TLB invalidate entry instruction (**tlbie**) whenever the corresponding PTE is modified. Since the 603e is intended primarily for uniprocessor environments, it does not provide coherency of TLBs between multiple processors. If the 603e is used in a multiprocessor environment where TLB coherency is required, all synchronization must be implemented in software.

Processors may write referenced and changed bits with unsynchronized, atomic byte store operations. Note that the V, R, and C bits each resides in a distinct byte of a PTE. Therefore, extreme care must be taken to use byte writes when updating only one of these bits.

Explicitly altering certain MSR bits (using the **mtmsr** instruction), or explicitly altering PTEs, or certain system registers, may have the side effect of changing the effective or physical addresses from which the current instruction stream is being fetched. This kind of side effect is defined as an implicit branch. Implicit branches are not supported and an attempt to perform one causes boundedly-undefined results. Therefore, PTEs must not be changed in a manner that causes an implicit branch. Chapter 2, “PowerPC Register Set,” in *The Programming Environments Manual*, lists the possible implicit branch conditions that can occur when system registers and MSR bits are changed.

5.5.4 Segment Register Updates

There are certain synchronization requirements for using the move to segment register instructions. These are described in “Synchronization Requirements for Special Registers and for Lookaside Buffers” in Chapter 2, “PowerPC Register Set,” in *The Programming Environments Manual*.

Chapter 6
Instruction Timing

This chapter describes instruction prefetch and execution through all of the execution units of the PowerPC 603e microprocessor. It also provides examples of instruction sequences showing concurrent execution and various register dependencies to illustrate timing interactions. Bus signals described in this chapter are only accurate to within half clock cycle increments. See Chapter 8, “System Interface Operation,” for more specific information regarding bus operation timing. Instruction mnemonics used in this chapter can be identified by referring to Chapter 8, “Instruction Set,” in *The Programming Environments Manual.*

6.1 Terminology and Conventions

This section describes terminology and conventions used in this chapter.

- Branch prediction—The process of guessing whether a branch will be taken. Such predictions can be correct or incorrect; the term predicted as it is used here does not imply that the prediction is correct (successful). The PowerPC architecture defines a means for static branch prediction, which is part of the instruction encoding.
- Branch resolution—The determination of whether a branch is taken or not taken. A branch is said to be resolved when it can exactly be determined which path it will take. If the branch is resolved as predicted, the instructions following the predicted branch can be completed. If the branch is not resolved as predicted, instructions on the mispredicted path are purged from the instruction pipeline and are replaced with the instructions from the nonpredicted path.
- Completion—Completion occurs when an instruction is removed from the completion buffer. When an instruction completes we can be sure that this instruction and all previous instructions will cause no exceptions. In some situations, an instruction can finish and complete in the same cycle.
- Finish—The term indicates the final cycle of execution. In this cycle, the completion buffer is updated to indicate that the instruction has finished executing.
- Latency—The number of clock cycles necessary to execute an instruction and make ready the results of that execution for a subsequent instruction.

- Pipeline—In the context of instruction timing, the term pipeline refers to the interconnection of the stages. The events necessary to process an instruction are broken into several cycle-length tasks to allow work to be performed on several instructions simultaneously—analogous to an assembly line. As an instruction is processed, it passes from one stage to the next. When it does, the stage becomes available for the next instruction.

  Although an individual instruction may take many cycles to complete (the number of cycles is called instruction latency), pipelining makes it possible to overlap the processing so that the throughput (number of instructions completed per cycle) is greater than if pipelining were not implemented.
- Program order—The original order in which program instructions are provided to the instruction queue from the cache.
- Rename buffer—Temporary buffers used by instructions that have not completed and as write-back buffers for those that have.
- Reservation station—A buffer between the dispatch and execute stages that allows instructions to be dispatched even though the operands required for execution may not yet be available.
- Stage—An element in the pipeline at which certain actions are performed, such as decoding the instruction, performing an arithmetic operation, and writing back the results. A stage typically takes a cycle to perform its operation; however, some stages are repeated (a double-precision floating-point multiply, for example). When this occurs, an instruction immediately following it in the pipeline is forced to stall in its cycle.

  In some cases, an instruction may also occupy more than one stage simultaneously—for example, instructions may complete and write back their results in the same cycle.

  After an instruction is fetched, it can always be defined as being in one or more stages.
- Stall—An occurrence when an instruction cannot proceed to the next stage.
- Superscalar—A superscalar processor is one that can issue multiple instructions concurrently from a conventional linear instruction stream. In a superscalar implementation, multiple instructions can be in the same stage at the same time.
- Throughput—A measure of the number of instructions that are processed per cycle. For example, a series of double-precision floating-point multiply instructions has a throughput of one instruction per clock cycle.
- Write-back—Write-back (in the context of instruction handling) occurs when a result is written from the rename registers into the architectural registers (typically the GPRs and FPRs). Results are written back at completion time or are moved into the write-back buffer. Results in the write-back buffer cannot be flushed. If an exception occurs, these buffers must write back before the exception is taken.

6.2 Instruction Timing Overview

The 603e has been designed to minimize average instruction execution latency. Latency is defined as the number of clock cycles necessary to execute an instruction and make ready the results of that execution for a subsequent instruction. For many of the instructions in the 603e, this can be simplified to include only the execute phase for a particular instruction. However, data access instructions require additional clock cycles between the execute phase and the write-back phase due to memory latencies.

In accordance with this definition, logical, bit-field, and most integer instructions have a latency of one clock cycle (for example, results for these instructions are ready for use on the next clock cycle after issue). Other instructions, such as the integer multiply, require more than one clock cycle to complete execution.

Effective throughput of more than one instruction per clock cycle can be realized by the many performance features in the 603e including pipelining, superscalar instruction issue, branch acceleration, and multiple execution units that operate independently and in parallel.

The load/store and floating-point units on the 603e are pipelined, which means that the execution units are broken into stages. Each stage performs a specific step, which contributes to the overall execution of an instruction. The pipelined design is analogous to an assembly line where workers perform a specific task and pass the partially complete product to the next worker. (Note: The EC603e microprocessor does not support the floating-point unit.)

When an instruction is issued to a pipelined execution unit, the first stage in the pipeline begins its designated work on that instruction. As an instruction is passed from one stage in the pipeline to the next, evacuated stages may accept new instructions. This design allows a single execution unit to be working on several different instructions simultaneously. While it may take several cycles for a given instruction to propagate through the execution pipeline, once the pipeline has been filled with instructions the execution unit is capable of completing an instruction every clock.

Figure 6-1 shows a graphical representation of a generic pipelined execution unit.

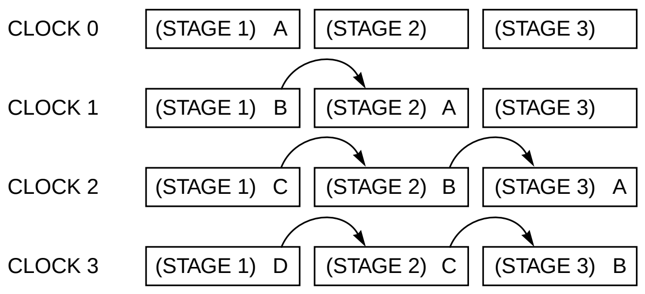


Figure 6-1. Pipelined Execution Unit

If the number of stages in each pipeline is equal to the total latency in clock cycles of its respective execution unit, the processor can continuously issue instructions to the same execution unit without stalling. Thus, when enough instructions have been issued to an execution unit to fill its pipeline, the first instruction will have completed execution and exited the pipeline, allowing subsequent instructions to be issued into the tail of the pipeline without interruption.

The 603e’s completion buffer is capable of retiring two instructions on every clock cycle. In general, instruction processing is accomplished in four stages described as follows:

- The fetch pipeline stage primarily involves retrieving instructions from the memory system and determining the location of the next instruction fetch. Additionally, the BPU decodes branches during the fetch stage and folds out branch instructions before the dispatch stage if possible. The instruction fetch stage includes the clock cycles necessary to request instructions from the on-chip cache as well as the time it takes the on-chip cache to respond to that request.
- The decode/dispatch pipeline stage is responsible for decoding the instructions supplied by the instruction fetch stage, and determining which of the instructions are eligible to be dispatched in the current cycle. In addition, the source operands of the instructions are read from the appropriate register file and dispatched with the instruction to the execute pipeline stage. At the end of the dispatch pipeline stage, the dispatched instructions and their operands are latched by the appropriate execution unit.
- During the execute pipeline stage each execution unit that has an executable instruction executes the selected instruction (perhaps over multiple cycles), writes the instruction's result into the appropriate rename register, and notifies the completion stage that the instruction has finished execution. In the case of an internal exception, the execution unit reports the exception to the completion/writeback pipeline stage and discontinues instruction execution until the exception is handled. The exception is not signaled until that instruction is the next to be completed. Execution of most floating-point instructions is pipelined within the FPU allowing up to three instructions to be executing in the FPU concurrently. The pipeline stages

for the floating-point unit are multiply, add, and round-convert. Execution of most load/store instructions is also pipelined. The load/store unit has two pipeline stages. The first stage is for effective address calculation and MMU translation and the second stage is for accessing the data in the cache.

- The complete/writeback pipeline stage maintains the correct architectural machine state and transfers the contents of the rename registers to the GPRs and FPRs as instructions are retired. If the completion logic detects an instruction causing an exception, all following instructions are canceled, their execution results in rename registers are discarded, and instructions are fetched from the correct instruction stream.

More information regarding these operations are provided in the following paragraphs.

6.3 Timing Considerations

A superscalar processor is one that issues multiple independent instructions into multiple pipelines allowing instructions to execute in parallel. The 603e has five independent execution units (four execution units on the EC603e microprocessor), one each for integer instructions, floating-point instructions (not supported on the EC603e microprocessor), branch instructions, load/store instructions, and system register instructions. The IU and the FPU each have dedicated register files for maintaining operands (GPRs and FPRs, respectively), allowing integer calculations and floating-point calculations to occur simultaneously without interference. Integer division performance of the PID7v-603e has been improved, with the **divwu***x* and **divw***x* instructions executing in 20 clock cycles, instead of the 37 cycles required in the PID6-603e.

Note: The FPU is not supported on the EC603e microprocessor; therefore, floating-point instructions are trapped by the floating-point unavailable exception and can be emulated in software.

The 603e is a true superscalar implementation of the PowerPC architecture since a maximum of three instructions can be issued to the execution units (one branch instruction to the branch processing unit, and two instructions issued from the dispatch queue to the other execution units) during each clock cycle. Although a superscalar implementation complicates instruction timing, these complications are transparent to the software. While the 603e appears to the programmer to execute instructions in sequential order, the 603e provides increased performance by executing multiple instructions at a time, and using hardware to manage dependencies.

The 603e provides support for single-cycle store and it provides an adder/comparator in the system register unit that allows the dispatch and execution of multiple integer add and compare instructions on each cycle.

When an instruction is issued, the register file places the appropriate source data on the appropriate source bus. The corresponding execution unit then reads the data from the bus.

The register files and source buses have sufficient bandwidth to allow the dispatching of two instructions per clock.

The 603e contains the following execution units that operate independently and in parallel:

- Branch processing unit (BPU)
- 32-bit integer unit (IU)
- 64-bit floating-point unit (FPU) (not supported on the EC603e microprocessor)
- Load/store unit (LSU)
- System register unit (SRU)

The 603e's branch processing unit decodes and executes branches immediately after they are fetched. The resources of the branch unit include—a count register (CTR) rename register for **mtspr**(CTR), a link register (LR) rename register for **mtspr**(LR), a link register (LR) rename register for branches specifying an update of the link register, and a branch reservation station for conditional branches that cannot be resolved due to a CR-data dependency.

When a conditional branch cannot be resolved due to a CR-data dependency, the branch direction is predicted and execution commences down the predicted path. If the branch resolves as incorrectly guessed, then:

1. the instruction buffer is purged and fetching of the correct path commences,
2. any instructions executed prior to the predicted branch in the completion buffer are allowed to "complete",
3. all instructions executed subsequent to the mispredicted branch are purged from the machine, and 4) dispatching down the correct path commences.

When the IU, SRU, or FPU (not supported on the EC603e microprocessor) finishes executing an instruction, it places the resulting data, if any, into one of the general-purpose register (GPR) or floating-point register (FPR) rename registers. The results are then stored into the correct GPR during the write-back stage. If a subsequent instruction is waiting for this data, it is forwarded past the register file, directly into the appropriate execution unit for the immediate execution of the waiting instruction. This allows a data-dependent instruction to be decoded without waiting for the data to be written into the register file and then read back out again. This feature, known as feed forwarding, significantly shortens the time the machine may stall on data dependencies.

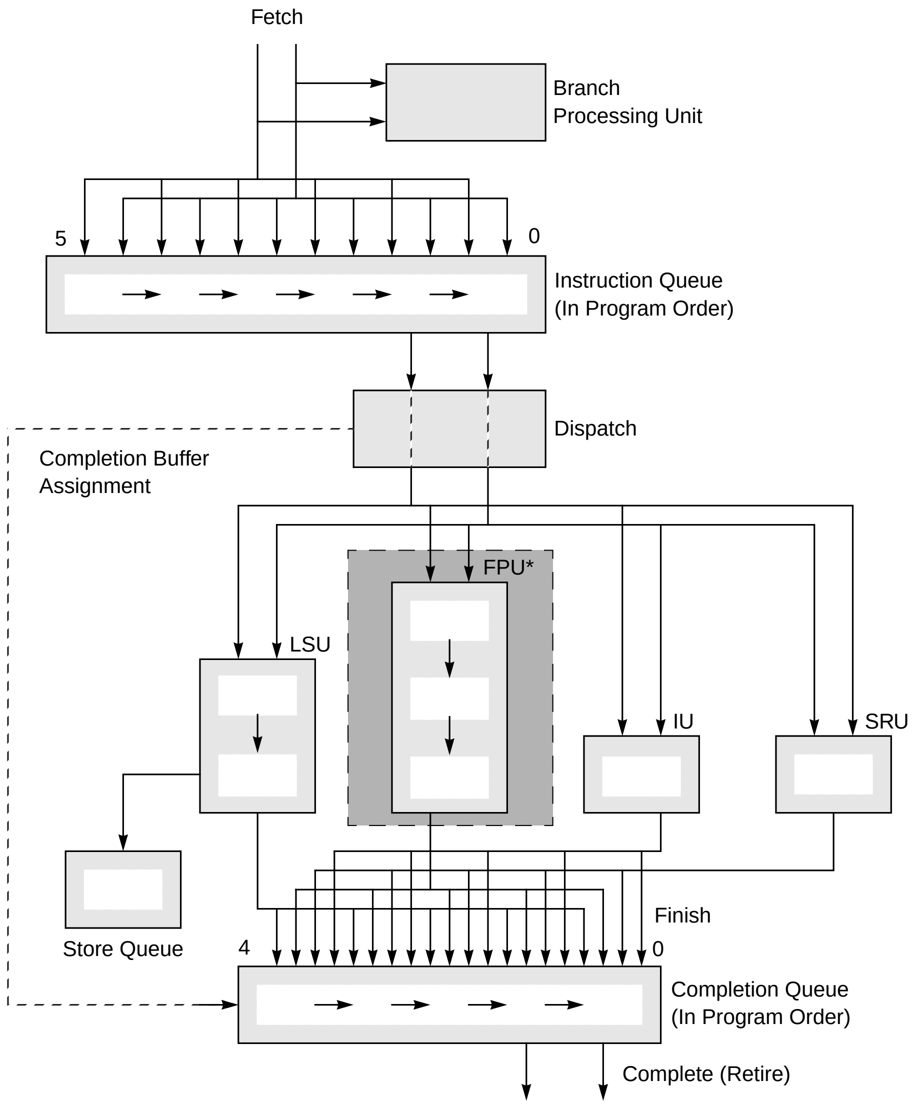
6.3.1 General Instruction Flow

Instructions are fetched from the instruction cache at a peak rate of two per cycle, and placed in either the instruction queue (IQ) or the BPU. Instructions enter the IQ and are issued to the various execution units from the dispatch queue. The IQ is a six-entry queue, which is the backbone of the master pipeline for the microprocessor. The 603e tries to keep the IQ full at all times. Although two instructions can be brought in from the on-chip cache in a single clock cycle, if there is a one-instruction vacancy in the IQ, one instruction will be fetched from the cache to fill it. If while topping off the IQ, the request for new instructions misses in the on-chip cache, then arbitration for a memory access will begin.

Instructions enter the IQ through entry 5 and filter down to be issued from queue entry 1 or 0. The fetch bus between the IQ and the on-chip cache is wide enough for two instructions to be brought into the IQ simultaneously, which matches the dispatcher’s ability to issue two instructions per cycle.

Branch instructions are identified by the fetcher, and forwarded to the BPU directly, bypassing the dispatch queue. The branch is either executed and resolved (if the branch is unconditional or if required conditions are available), or is predicted. Once a branch instruction has been executed, it may need to update a special-purpose register. In that case, the branch instruction will do its write back sometime after the decode/execute phase. If no write back is needed, the branch instruction is retired. All other instructions are issued from the dispatch queue, with dispatch rate contingent on execution unit busy status, rename and completion buffer availability, and the serializing behavior of some instructions. Instruction dispatch is done in program order, and if the instruction in queue entry 0 is unable to be dispatched, it will inhibit the instruction in queue entry 1 from being issued.

Figure 6-2 reflects the organization of the 603e, and the paths taken by instructions issued from the instruction queue and how those instructions progress through the various execution units.



Fetch
Branch Processing Unit
5
0
Instruction Queue (In Program Order)
Dispatch
Completion Buffer Assignment
FPU*
LSU
IU
SRU
Store Queue
Finish
4
0
Completion Queue (In Program Order)
Complete (Retire)

Note: The EC603e microprocessor does not support the floating-point unit.

Figure 6-2. Instruction Flow Diagram

6.3.2 Instruction Fetch Timing

The timing of the instruction fetch mechanism on the 603e depends heavily on the state of the on-chip cache. The speed with which the required instruction is returned to the fetcher depends on whether the instruction being asked for is in the on-chip cache (cache hit) or whether a memory transaction is required to bring the data into the cache (cache miss).

These issues are discussed further in the following sections.

6.3.2.1 Cache Arbitration

When the instruction fetcher attempts to fetch instructions from the on-chip cache, the cache may or may not be able to immediately respond to the request. There are two scenarios that may be encountered by the instruction fetcher when it requests instructions from the on-chip cache.

The first scenario is when the on-chip cache is idle and a request comes in from the instruction fetcher for additional instructions. In this case, the on-chip cache responds with the requested instructions on the next clock cycle.

The second scenario occurs if at the time the instruction fetcher requests instructions, the on-chip cache is busy due to a cache-line-reload operation. When this case arises, the on-chip cache will be inaccessible until the reload operation is complete.

6.3.2.2 Cache Hit

Assuming that the instruction fetcher is not blocked from the cache by a cache-reload operation and the instructions it needs are in the on-chip cache (a cache hit has occurred), there will be only one clock cycle between the time that the instruction fetcher requests the instructions and the time that the instructions enter the IQ. As previously stated, two instructions can be simultaneously fetched from the on-chip cache and loaded into the IQ.

Figure 6-3 shows a brief example of instruction fetching that hits in the on-chip cache. In this example, two instructions are fetched into the IQ during clock cycle 0. During clock cycle 1, instructions 0 and 1 are dispatched to the integer and floating-point execution units. During clock cycle 2, a branch instruction is fetched into the branch processing unit. The BPU is immediately able to determine that the branch will indeed change program flow and sends a request to the on-chip cache for the new instruction stream.

During clock cycle 4, the new instructions arrive in the IQ. In clock cycle 5, one integer instruction is dispatched to the integer unit, and the following instruction (also an integer instruction) is blocked from dispatch until clock cycle 6. Instructions fetched in clock cycle 5 are held in the IQ until the dispatch queue is cleared on the next cycle. As the IQ is emptied into the individual execution units, additional instructions will be requested from the on-chip cache.

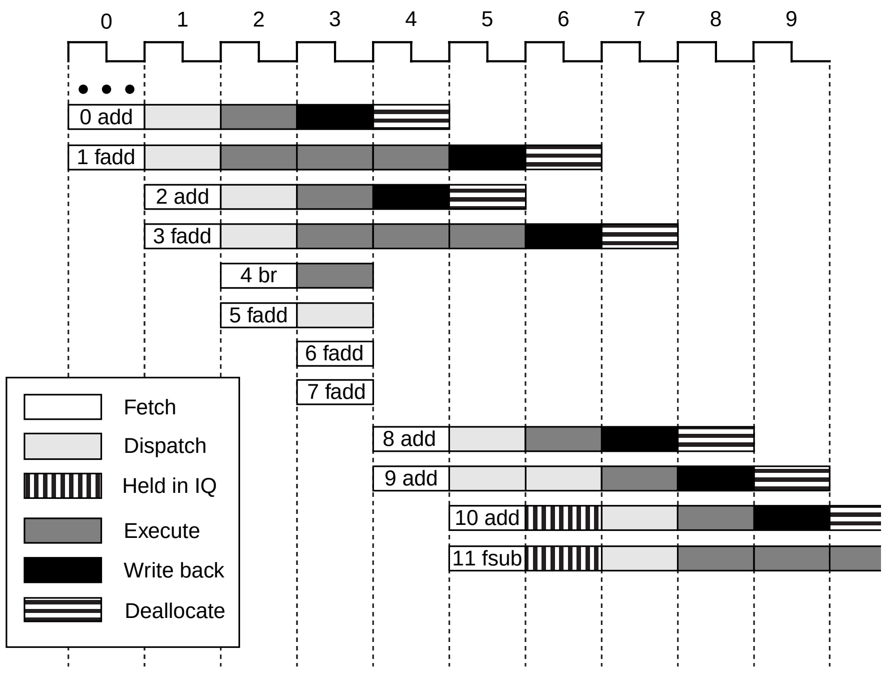


Figure 6-3. Instruction Timing—Cache Hit

6.3.2.3 Cache Miss

Figure 6-4 shows a brief example of an instruction fetch that misses in the on-chip cache and how that fetch affects the instruction issue. Note that the processor/bus clock ratio is 1:1 in this example.

In this example, two instructions are fetched into the IQ during clock cycle 0. During clock cycle 1, instructions 0 and 1 are dispatched to the integer and floating-point execution units. During clock cycle 2, a branch instruction is fetched into the branch processing unit. The BPU is immediately able to determine that the branch will indeed change program flow and sends a request to the on-chip cache for the new instruction stream.

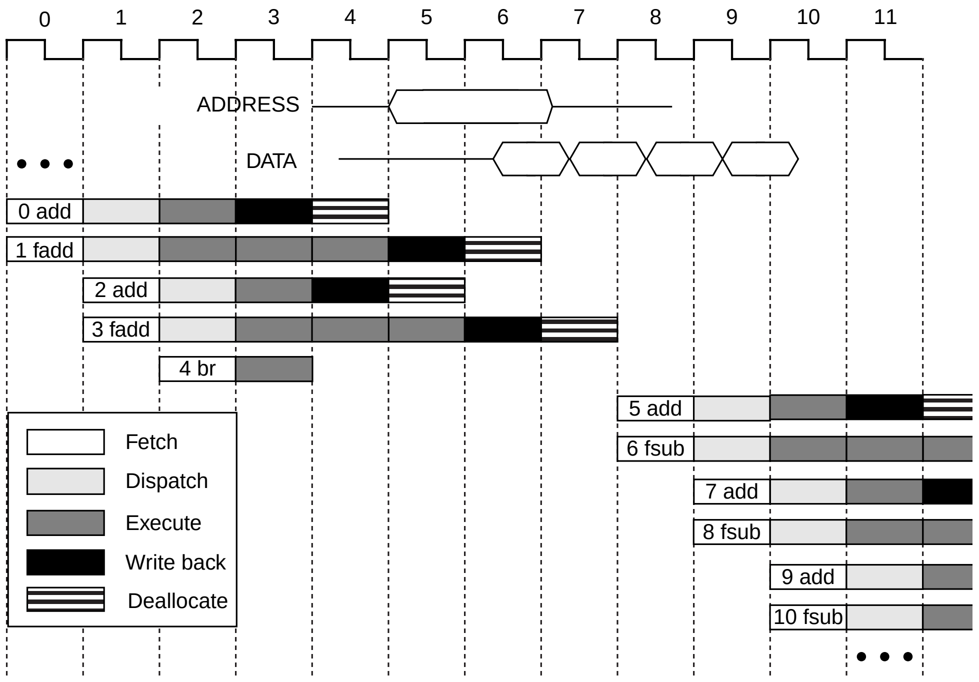


Figure 6-4. Instruction Timing—Cache Miss

During clock cycle 3, the on-chip cache misses the access and determines that a memory access will have to occur. During clock cycle 5, the address of the block of instructions is applied to the system bus. During clock cycle 7, two instructions (64 bits) are returned from memory, and are forwarded to the cache and the instruction fetcher. In subsequent clock cycles, one integer and one floating-point instruction is dispatched to their respective execution units. Instructions are forwarded to the instruction fetcher and the cache until the cache line reload is completed in cycle 10.

6.3.3 Instruction Dispatch and Completion Considerations

Several factors affect the 603e's ability to dispatch instructions at a peak rate of two per cycle. These factors include execution unit availability, destination rename register availability, completion buffer availability, and the handling of dispatch-serialized instructions.

To avoid dispatch unit stalls due to instruction data dependencies, the 603e provides a reservation station for each execution unit. If a data dependency exists that may preclude an instruction from beginning execution, that instruction will be dispatched to the reservation station associated with its execution unit, thereby clearing the dispatch unit. When the data that the operation depends upon is returned via a cache access or as a result of a previous operation, execution will begin during the same clock cycle that the register file is being updated. If the second instruction in the dispatch unit requires the same execution unit, dispatch of that instruction will stall until the first instruction completes execution.

The completion unit provides a mechanism to track instructions from dispatch through execution, and then retire or "complete" them in program order. Completing an instruction implies the commitment of the results of instruction execution to the architected registers. In-order completion ensures the correct architectural state when the 603e must recover from a mispredicted branch, or any other exception or interrupt. (Note that the term exception is referred to as interrupt in the architecture specification.)

Instruction state and all information required for completion is kept in a first-in, first-out queue of five completion buffers. A single completion buffer is allocated for each instruction once it is dispatched by the dispatch unit. A completion buffer is a required resource for dispatch; if there are no completion buffers available, the dispatch unit will stall. While a maximum of two instructions per cycle may be completed and retired in program order from the completion unit, instruction completion can be stalled by the instruction reaching the last position in the completion queue while the instruction is still being executed. Store instructions, and instructions executed by the FPU (not supported by the EC603e microprocessor) and SRU (with the exception of integer add and compare instructions) can only be retired from the last position in the completion queue.

The rate of instruction completion is also affected by the 603e's ability to write the instruction results from the rename registers to the architected registers when the instruction is retired. The 603e can perform two write-back operations from the rename registers to the GPRs each clock cycle, but can perform only one write back per cycle to the CR, FPR (not supported on the EC603e microprocessor), LR, and CTR.

Due to the 603e's out-of-order execution capability, the in-order completion of instructions by the completion unit provides a precise exception mechanism. All program-related exceptions are signaled when the instruction causing the exception has reached the last position in the completion buffer. All prior instructions are allowed to complete before the exception is taken.

6.3.3.1 Rename Register Operation

To avoid contention for a given register file location in the course of out-of-order execution, the 603e provides rename registers for the storage of instruction results prior to their commitment to the architected register by the completion unit. Five rename registers are provided for the GPRs, four for the FPRs (not supported on the EC603e microprocessor), and one each for the condition register, the link register and the count register.

When the dispatch unit dispatches an instruction to its execution unit, it allocates a rename register for the results of that instruction. If an instruction is dispatched to a reservation station associated with an execution unit due to a data dependency, the dispatcher will also provide a tag to the execution unit identifying which rename register will forward the required data upon instruction completion. When the data is available in the rename register, the pending execution may begin.

Instruction results are transferred from the rename registers to the architected registers by the completion unit when an instruction is retired from the completion queue without exceptions and after any predicted branch conditions preceding it in the completion queue have been resolved correctly. If a predicted branch is found to have been incorrectly predicted, the instructions following the branch will be flushed from the completion queue, and the results of those instructions will be flushed from the rename registers.

6.3.3.2 Instruction Serialization

While the 603e is capable of dispatching and completing two instructions per cycle, there is a class of instructions referred to as serializing instructions that limit dispatch and completion to one instruction per cycle. The type of serialization caused by these instructions fall into three categories—completion, dispatch, and refetch serialization.

Completion serialized instructions are held in the execution unit until all prior instructions in the completion unit have been retired. Completion serialization is used for instructions that access or modify nonrenamed resources. Results from these instructions will not be available or forwarded for subsequent instructions until the serializing instruction is retired from the completion unit. Instructions that are completion serialized are as follows:

- Instructions (with the exception of integer add and compare instructions) executed by the system register unit
- Floating-point instructions that access or modify the FPSCR (not supported on the EC603e microprocessor) or CR (**mtfsb1**, **mcrfs**, **mtfsfi**, **mffs**, and **mtfsf**)
- Instructions that manage caches and TLBs
- Instructions that directly access the GPRs (load and store multiple word and load and store string instructions)
- Instructions defined by the architecture to have synchronizing behavior

A subset of the completion serialized instructions are dispatch serialized. Dispatch serialized instructions inhibit the dispatching of subsequent instructions until the serializing instruction is retired from the completion unit. Dispatch serialization is used for instructions that access renamed resources used by the dispatcher, and for instructions requiring refetch serialization, including:

- The load multiple instructions, **lmw**, **lswi**, and **lswx**
- The **mtspr**(XER) and **mcrxr** instructions
- The synchronizing instructions, **sync**, **isync**, **mtmsr**, **rfi**, and **sc**

A subset of the dispatch serialized instructions are also refetch serialized. Refetch serialized instructions inhibit dispatching of subsequent instructions and force the refetching of subsequent instructions after the serializing instructions are retired from the completion unit. The context synchronizing instruction, **isync**, is a refetch serializing instruction.

6.3.3.3 Execution Unit Considerations

As previously noted, the 603e is capable of dispatching and retiring two instructions per clock cycle. One of the factors affecting the peak dispatch rate is the availability of execution units on each clock cycle.

For an instruction to be issued, the required execution unit must be available. The dispatcher monitors the availability of all execution units and suspends instruction dispatch if the required execution unit is not available. An execution unit may not be available if it can accept and execute only one instruction per cycle, or if an execution unit's pipeline becomes full. This situation may occur if instruction execution takes more clock cycles than the number of pipeline stages in the unit, and additional instructions are issued to that unit to fill the remaining pipeline stages.

6.4 Execution Unit Timings

The following sections describe instruction timing considerations within each of the respective execution units in the 603e. Refer to Table 6-1 for branch instruction execution timing.

6.4.1 Branch Processing Unit Execution Timing

Flow control operations (conditional branches, unconditional branches, and traps) are typically expensive to execute in most machines because they disrupt normal flow in the instruction stream. When a change in program flow occurs, the IQ must be reloaded with the target instruction stream. During this time the execution units will be idle. However, previously issued instructions will continue to execute while the new instruction stream makes its way into the IQ.

Performance features such as branch folding and static branch prediction help minimize the penalties associated with flow control operations on the 603e. The timing for branch instruction execution is determined by many factors including the following:

- Whether the branch is taken
- Whether the target instruction stream is in the on-chip cache
- Whether the branch is predicted
- Whether the prediction is correct

6.4.1.1 Branch Folding

When a branch instruction is encountered by the fetcher, the BPU immediately tries to pull that instruction out of the instruction stream and resolve it. When the BPU pulls the branch instruction out of the instruction stream, the instruction above the branch is shifted down to take the place of the removed branch. The technique of removing the branch instruction from the instruction sequence seen by the other execution units, is known as branch folding.

Often, branch folding reduces the penalties of flow control instructions to zero since instruction execution proceeds as though the branch was never there.

If the folded branch instruction changes program flow (the branch is said to be “taken” in this case), the BPU immediately requests the instructions at the new target from the on-chip cache. In most cases, the new instructions arrive in the IQ before any bubbles are introduced into the execution units. If the folded branch does not change program flow (the branch is said to be “not taken” in this case), the branch is already removed from the instruction stream and execution continues as if there were never a branch in the original sequence.

When a conditional branch cannot be resolved due to a CR data dependency, the branch is executed by means of static branch prediction, and instruction fetching proceeds down the predicted path. If the branch prediction was incorrect when the branch is resolved, the instruction queue and all subsequently executed instructions are purged, instructions executed prior to the predicted branch are allowed to complete, and instruction fetching resumes down the correct path.

There are several situations where instruction sequences create dependencies that prevent a branch instruction from being resolved immediately, thereby causing execution of the subsequent instruction stream based on the predicted outcome of the branch instruction. The instruction sequences, and the resulting action of the branch instruction is described as follows:

- An **mtspr**(LK) followed by a **bclr**—Fetching is stopped, and the branch waits for the **mtspr** to execute.
- An **mtspr**(CTR) followed by a **bcctr**—Fetching is stopped, and the branch waits for the **mtspr** to execute.
- An **mtspr**(CTR) followed by a **bc**(CTR)—Fetching is stopped, and the branch waits for the **mtspr** to execute.
- A **bc**(CTR) followed by another **bc**(CTR)—Fetching is stopped, and the second branch waits for the first branch to be completed.
- A **bc**(CTR) followed by a **bcctr**—Fetching is stopped, and the **bcctr** waits for the first branch to be completed.
- A branch(LK = 1) followed by a branch(LK = 1)—Fetching is stopped, and the second branch waits for the first branch to be completed. (Note: a **bl** instruction does not have to wait for a branch(LK = 1) to complete.)
- A **bc**(based-on-CR) waiting for resolution due to a CR-dependency followed by a **bc**(based-on-CR)—Fetching is stopped and the second branch waits for the first CR-dependency to be resolved. (Note: branch conditions can be a function of the CTR and the CR; if the CTR condition is sufficient to resolve the branch, then a CR-dependency is ignored.)

6.4.1.2 Static Branch Prediction

Static branch prediction is a mechanism by which software (for example, compilers) can give a hint to the machine hardware about the direction the branch is likely to take. When a branch instruction encounters a data dependency, the BPU waits for the required condition code to become available. Rather than stalling instruction issue until the source operand is ready, the 603e predicts which path the branch instruction is likely to take, and instructions are fetched and executed along that path. When the branch operand becomes available, the branch is evaluated. If the predicted path was correct, program flow continues along that path uninterrupted; otherwise, the processor backs up, and program flow resumes along the correct path.

There is a scenario where a flow control instruction will not be predicted on the 603e. If the target address of the branch (link or count register) will be modified by an instruction that appears before the branch instruction, the BPU must wait until the target address is available.

The 603e executes through one level of prediction. The microprocessor may not predict a branch if a prior branch instruction is still unresolved.

The number of instructions that can be executed after the issue of a predicted branch instruction is limited by the fact that no instruction executed after a predicted branch may actually update the register files or memory until the branch is completed. That is, instructions may be issued and executed, but may not reach the write-back stage in the completion unit. When an instruction following a predicted branch has completed execution, it will not be moved into the write-back stage, instead, it will simply stall in the last stage of the completion unit. This means that the completion queue may become full, which will limit the number of additional instructions that may be issued subsequent to an unresolved predicted branch.

In the case of a misprediction, the 603e is able to redirect its machine state rather painlessly because the programing model has not been updated. When a branch is found to be mispredicted, all instructions that were issued subsequent to the predicted branch instruction are simply flushed from the completion queue, and their results flushed from the rename registers. No architected register state needs to be restored because no architected register state was modified by the instructions following the unresolved predicted branch.

6.4.1.2.1 Predicted Branch Timing Examples

Figure 6-5 depicts the cases where branch instructions are predicted, and shows both “taken” and “not taken” branch outcomes. During clock cycle 0, two instructions are dispatched to their respective execution units. Notice that the BPU has a combined decode/execute stage, thus the branch (instruction 1) is predicted not to be taken during clock cycle 1 because its source register (condition register) is not available.

During clock cycle 2, instructions 0 and 2 progress through their pipelines. In addition, the branch (instruction 1) remains predicted. Notice that the next branch instruction (instruction 5) is not able to begin its decode/execute phase while instruction 1 is predicted.

During clock cycle 3, instruction 0 begins its write-back stage. The write back of instruction 0 resolves the data dependency for the first branch (instruction 1); thus the first branch becomes resolved and it is determined that the prediction was correct. Recall that only one branch may be predicted at a time; thus, when instruction 1 is resolved the BPU is free to predict instruction 5.

During clock 4, the second branch instruction remains predicted while additional instructions move through the various pipelines.

During clock cycle 5, the BPU realizes that the prediction made for instruction 5 was incorrect. Note that since instruction 6 was issued and executed conditionally, it never performed its write back. As a result of the misprediction, all instructions that followed the branch in the instruction stream must be flushed from the respective execution unit pipelines. Notice that instructions 6 and 7 do not continue execution since it has been determined that these instructions should have never been issued in the first place. Since the branch has been resolved, a request is sent to the on-chip cache for the new instruction stream (based on the execution of instruction 5). During clock 6, the new set of instructions are in the IQ and the appropriate dispatching begins on clock cycle 7.

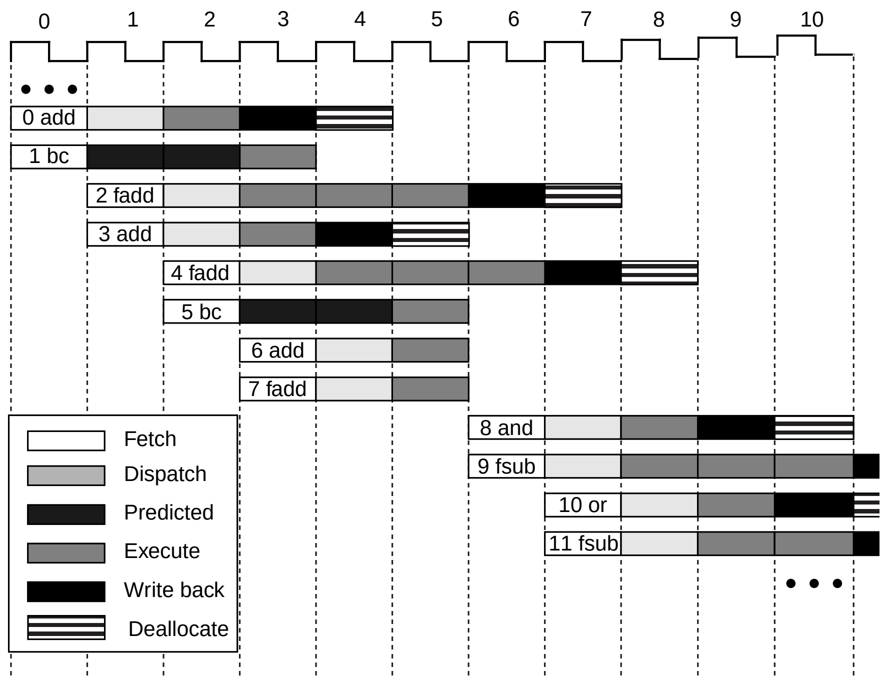


Figure 6-5. Branch Instruction Timing

6.4.2 Integer Unit Execution Timing

The integer unit executes all integer and bit-field instructions. Many of these instructions execute in a single clock cycle. The integer unit has one execute phase in its pipeline, thus when a multicycle integer instruction is being executed, no other integer instructions may begin an execute phase. Refer to Table 6-4 for integer instruction execution timing.

6.4.3 Floating-Point Unit Execution Timing

The floating-point unit on the 603e (not supported on the EC603e microprocessor) executes all floating-point instructions. Execution of most floating-point instructions is pipelined within the FPU, allowing up to three instructions to be executing in the FPU concurrently. While most floating-point instructions execute with three- or four-cycle latency, and one- or two-cycle throughput, three instructions (**fdivs**, **fdiv**, and **fres**) execute with latencies of 18 to 33 cycles. The **fdivs**, **fdiv**, **fres**, **mtfsb0**, **mtfsb1**, **mtfsfi**, **mffs**, and **mtfsf** instructions block the floating-point unit pipeline until they complete execution, and thereby inhibit the dispatch of additional floating-point instructions. With the exception of the **mcrfs** instruction, all floating-point instructions will immediately forward their CR results to the BPU for fast branch resolution without waiting for the instruction to be retired by the completion unit, and the CR updated. Refer to Table 6-5 for floating-point instruction execution timing.

6.4.4 Load/Store Unit Execution Timing

The execution of most load and store instructions is pipelined. The LSU has two pipeline stages; the first stage is for effective address calculation and MMU translation, and the second stage is for accessing the data in the cache. Load and store instructions have a two-cycle latency and one-cycle throughput. Load instructions that miss in the cache block subsequent accesses to the cache while the cache line refill is in process. Refer to Table 6-6 for load and store instruction execution timing.

6.4.5 System Register Unit Execution Timing

The majority of the instructions executed by the SRU access or modify nonrenamed registers, or directly access renamed registers, and generally execute in a serial manner. Results from these instructions will not be available or forwarded for use by subsequent instructions until the instruction completes and is retired. The SRU can also execute the integer instructions **addi**, **addis**, **add**, **addo**, **cmpi**, **cmp**, **cmpli**, and **cmpl** without serialization, and in parallel with another integer instruction. Refer to Section 6.3.3.2, “Instruction Serialization,” for additional information on serializing instructions executed by the SRU, and Table 6-2, Table 6-3, and Table 6-4 for SRU instruction execution timing.

6.5 Memory Performance Considerations

Due to the 603e’s instruction throughput of three instructions per clock cycle, lack of data bandwidth can become a performance bottleneck. In order for the 603e to approach its potential performance levels, it must be able to read and write data quickly and efficiently.

If there are many processors in a system environment, one processor may experience long memory latencies while another bus master (for example, a direct memory access controller) is using the external bus.

In order to alleviate this possible contention, the 603e provides three memory update modes—copy-back, write-through, and cache-inhibit. Each page of memory is specified to be in one of these modes. If a page is in copy-back mode, data being stored to that page is written only to the on-chip cache. If a page is in write-through mode, writes to that page update the on-chip cache on hits and always update main memory. If a page is cache-inhibited, data in that page will never be stored in the on-chip cache. All three of these modes of operation have advantages and disadvantages. A decision as to which mode to use depends on the system environment as well as the application.

This section describes how performance is impacted by each memory update mode. For details about the operation of the on-chip cache and the memory update modes, see Chapter 3, “Instruction and Data Cache Operation.”

6.5.1 Copy-Back Mode

When storing data while in copy-back mode, store operations for cacheable data do not necessarily cause an external bus cycle to update memory. Instead, memory updates only occur on modified line replacements, cache flushes, or when another processor attempts to access a specific address for which there is a corresponding modified cache entry. For this reason, copy-back mode may be preferred when external bus bandwidth is a potential bottleneck—for example, in a multiprocessor environment. Copy-back mode is also well suited for data that is closely coupled to a processor, such as local variables.

If more than one device uses data stored in a page that is in copy-back mode, snooping must be enabled to allow copy-back operations and cache invalidations of modified data. The 603e implements snooping hardware to prevent other devices from accessing invalid data. When bus snooping is enabled, the processor monitors the transactions of the other devices. For example, if another device accesses a memory location and its memory-coherent (M) bit is set, and the 603e’s on-chip cache has a modified value for that address, the processor preempts the bus transaction, and updates memory with the cache data. If the cache contents associated with the snooped address are unmodified, the 603e will invalidate the cache block. The other device is then free to attempt an access to the updated memory address. See Chapter 3, “Instruction and Data Cache Operation,” for complete information about bus snooping.

Copy-back mode provides complete cache/memory coherency as well as maximizing available external bus bandwidth.

6.5.2 Write-Through Mode

Store operations to memory in write-through mode always update memory as well as the on-chip cache (on cache hits). Write-through mode is used when the data in the cache must always agree with external memory (for example, video memory), or when there is shared (global) data that may be used frequently, or when allocation of a cache line on a cache miss is undesirable. Automatic copy back of cached data is not performed if that data is from a memory page marked as write-through mode since valid cache data always agrees with memory.

Stores to memory that are in write-through mode may cause a decrease in performance. Each time a store is performed to memory in write-through mode, the bus will be busy for the extra clock cycles required to perform the memory update; therefore, load operations that miss the on-chip cache must wait while the external store operation completes.

6.5.3 Cache-Inhibited Accesses

If a memory page is specified to be cache-inhibited, data from this page will not be stored in the on-chip cache.

Areas of the memory map can be cache-inhibited by the operating system software. If a cache-inhibited access hits in the on-chip cache, the corresponding cache line is invalidated. If the line is marked as modified, it is copied back to memory before being invalidated.

In summary, the copy-back mode allows both load and store operations to use the on-chip cache. The write-through mode allows load operations to use the on-chip cache, but store operations cause a memory access and a cache update if the data is already in the cache. Lastly, the cache-inhibited mode causes memory access for both loads and stores.

6.6 Instruction Scheduling Guidelines

The performance of the 603e can be improved by avoiding resource conflicts and promoting parallel utilization of execution units through efficient instruction scheduling. Instruction scheduling on the 603e can be improved by observing the following guidelines:

- Implement good static branch prediction (setting of y bit in BO field).
- When branch prediction is uncertain, or an even probability, predict fall through.
- To reduce mispredictions, separate the instruction that sets CR bits from the branch instruction that evaluates them; separation by more than nine instructions ensures that the CR bits will be immediately available for evaluation.
- When branching conditionally to a location specified by count registers (CTRs) or link registers (LRs), or when branching conditionally based on the value in the count register, separate the **mtspr** instruction that initializes the CTR or LR from the branch instruction performing the evaluation. Separation of the branch instruction and the **mtspr** instruction by more than nine instructions ensures the register values will be immediately available for use by the branch instruction.
- Schedule instructions such that they can dual issue.
- Schedule instructions to minimize execution-unit-busy stalls.

- Avoid using serializing instructions.
- Schedule instructions to avoid dispatch stalls due to renamed resource limitations.
  - Only five instructions can be in execute-complete stage at any one time
  - Only five GPR destinations can be in execute-complete-deallocate stage at any one time. Note that load with update address instructions use two destination registers.
  - Only four FPR destinations can be in execute-complete-deallocate stage at any one time. (Not supported on the EC603e microprocessor)

6.6.1 Branch, Dispatch, and Completion Unit Resource Requirements

This section describes the specific resources required to avoid stalls during branch resolution, instruction dispatching, and instruction completion.

6.6.1.1 Branch Resolution Resource Requirements

The following is a list of branch instructions and the resources required to avoid stalling the fetch unit in the course of branch resolution:

- The **bclr** instruction requires LR availability.
- The **bcctr** instruction requires CTR availability.
- “Branch and link” instructions require shadow LR availability.
- The “branch conditional on counter decrement and CR condition” requires CTR availability or the CR condition must be false, and 603e cannot be executing instructions following an unresolved predicted branch when the branch is encountered by the BPU.
- The “branch conditional on CR condition” cannot be executed following an unresolved predicted branch instruction.

6.6.1.2 Dispatch Unit Resource Requirements

The following is a list of resources required to avoid stalls in the dispatch unit; note that the two dispatch buffers are described as DQ[0] and DQ[1], where DQ[0] is the dispatch buffer located at the very bottom of the dispatch queue:

- Requirements for dispatching from DQ[0] are as follows:
  - Needed execution unit available
  - Needed GPR rename register(s) available
  - Needed FPR rename registers available (not supported on the EC603e microprocessor)
  - Completion buffer is not full
  - Instruction is dispatch serialized and completion buffer is empty
  - A dispatch serialized instruction is not currently being executed

- Requirements for dispatching from DQ[1] are as follows:
  - Instruction in DQ[0] must dispatch
  - Instruction dispatched by DQ[0] is not dispatch serialized
  - Needed execution unit is available (after dispatch from DQ[0])
  - Needed GPR rename registers(s) are available (after dispatch from DQ[0])
  - Needed FPR rename register is available (after dispatch from DQ[0]) (Not supported on the EC603e microprocessor)
  - Completion buffer is not full (after dispatch from DQ[0])
  - Instruction dispatched from DQ[1] is not dispatch serialized

6.6.1.3 Completion Unit Resource Requirements

The following is a list of resources required to avoid stalls in the completion unit; note that the two completion buffers are described as CQ[0] and CQ[1], where CQ[0] is the completion buffer located at the very end of the completion queue:

- Requirements for completing an instruction from CQ[0] are as follows:
  - Instruction in CQ[0] must be finished
  - Instruction in CQ[0] must not follow an unresolved predicted branch
  - Instruction in CQ[0] must not cause an exception
- Requirements for completing an instruction from CQ[1] are as follows:
  - Instruction in CQ[0] must complete in same cycle
  - Instruction in CQ[1] must be finished
  - Instruction in CQ[1] must not follow an unresolved predicted branch
  - Instruction in CQ[1] must not cause an exception
  - Instruction in CQ[1] must be an integer or load instruction
  - Number of CR updates from both CQ[0] and CQ[1] must not exceed one
  - Number of GPR updates from both CQ[0] and CQ[1] must not exceed two
  - Number of FPR updates from both CQ[0] and CQ[1] must not exceed one (not supported on the EC603e microprocessor)

6.7 Instruction Latency Summary

Table 6-1 through Table 6-6 list the latencies associated with each instruction executed by the 603e. Note that the instruction latency tables contain no 64-bit architected instructions. These instructions will trap to an illegal instruction exception handler when encountered. Recall that the term latency is defined as the total time it takes to execute an instruction and make ready the results of that instruction.

Table 6-1 provides the latencies for the branch instructions.

Table 6-1. Branch Instructions

Primary	Extended	Mnemonic	Unit	Cycles
16	---	**bc**[l][a]	BPU	1*
18	---	**b**[l][a]	BPU	1*
19	016	**bclr**[l]	BPU	1*
19	528	**bcctr**[l]	BPU	1*

*These operations may be folded for an effective cycle time of 0.

Table 6-2 provides the latencies for the system register instructions.

Table 6-2. System Register Instructions

Primary	Extended	Mnemonic	Unit	Cycles
17	- -1	**sc**	SRU	3
19	050	**rfi**	SRU	3
19	150	**isync**	SRU	1&
31	083	**mfmsr**	SRU	1
31	146	**mtmsr**	SRU	2
31	210	**mtsr**	SRU	2
31	242	**mtsrin**	SRU	2
31	339	**mfspr** (not I/DBATs)	SRU	1
31	339	**mfspr** (DBATs)	SRU	3&
31	339	**mfspr** (IBATs)	SRU	3&
31	467	**mtspr** (not IBATs)	SRU	2 (XER-&)
31	467	**mtspr** (IBATs)	SRU	2&
31	595	**mfsr**	SRU	3&
31	598	**sync**	SRU	1&
31	659	**mfsrin**	SRU	3&
31	854	**eieio**	SRU	1
31	371	**mftb**	SRU	1
31	467	**mttb**	SRU	1

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

Table 6-3 provides the latencies for the condition register logical instructions.

Table 6-3. Condition Register Logical Instructions

Primary	Extended	Mnemonic	Unit	Cycles
19	000	**mcrf**	SRU	1
19	033	**crnor**	SRU	1
19	129	**crandc**	SRU	1
19	193	**crxor**	SRU	1
19	225	**crnand**	SRU	1
19	257	**crand**	SRU	1
19	289	**creqv**	SRU	1
19	417	**crorc**	SRU	1
19	449	**cror**	SRU	1
31	019	**mfcr**	SRU	1
31	144	**mtcrf**	SRU	1
31	512	**mcrxr**	SRU	1&

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

Table 6-4 provides the latencies for the integer instructions.

Table 6-4. Integer Instructions

Primary	Extended	Mnemonic	Unit	Cycles
03	—	**twi**	Integer	2
07	—	**mulli**	Integer	2,3
08	—	**subfic**	Integer	1
10	—	**cmpli**	Integer & SRU	1^
11	—	**cmpi**	Integer & SRU	1^
12	—	**addic**	Integer	1
13	—	**addic.**	Integer	1
14	—	**addi**	Integer & SRU	1
15	—	**addis**	Integer & SRU	1
20	—	**rlwimi**[.]	Integer	1
21	—	**rlwinm**[.]	Integer	1

Table 6-4. Integer Instructions (Continued)

Primary	Extended	Mnemonic	Unit	Cycles
23	—	**rlwnm**[.]	Integer	1
24	—	**ori**	Integer	1
25	—	**oris**	Integer	1
26	—	**xori**	Integer	1
27	—	**xoris**	Integer	1
28	—	**andi.**	Integer	1
29	—	**andis.**	Integer	1
31	000	**cmp**	Integer & SRU	1^
31	004	**tw**	Integer	2
31	008	**subfc**[o][.]	Integer	1
31	010	**addc**[o][.]	Integer	1
31	011	**mulhwu**[.]	Integer	2,3,4,5,6
31	024	**slw**[.]	Integer	1
31	026	**cntlzw**[.]	Integer	1
31	028	**and**[.]	Integer	1
31	032	**cmpl**	Integer & SRU	1^
31	040	**subf**[.]	Integer	1
31	060	**andc**[.]	Integer	1
31	075	**mulhw**[.]	Integer	2,3,4,5
31	104	**neg**[o][.]	Integer	1
31	124	**nor**[.]	Integer	1
31	136	**subfe**[o][.]	Integer	1
31	138	**adde**[o][.]	Integer	1
31	200	**subfze**[o][.]	Integer	1
31	202	**addze**[o][.]	Integer	1
31	232	**subfme**[o][.]	Integer	1
31	234	**addme**[o][.]	Integer	1
31	235	**mull**[o][.]	Integer	2,3,4,5
31	266	**add**[o][.]	Integer & SRU[1]	1
31	284	**eqv**[.]	Integer	1
31	316	**xor**[.]	Integer	1

Table 6-4. Integer Instructions (Continued)

Primary	Extended	Mnemonic	Unit	Cycles
31	412	**orc**[.]	Integer	1
31	444	**or**[.]	Integer	1
31	459	**divwu**[o][.]	Integer	37
31	476	**nand**[.]	Integer	1
31	491	**divw**[o][.]	Integer	37
31	536	**srw**[.]	Integer	1
31	792	**sraw**[.]	Integer	1
31	824	**srawi**[.]	Integer	1
31	922	**extsh**[.]	Integer	1
31	954	**extsb**[.]	Integer	1

Notes:

"^" indicates that the cycle time immediately forwards their CR results to the BPU for fast branch resolution.

1. The SRU can only execute the **add** and **add[o]** instructions.

Table 6-5 provides the latencies for the floating-point instructions. Note that floating-point instructions are not supported on the EC603e microprocessor and execution of a floating-point instruction will result in a trap to the floating-point unavailable exception vector.

Table 6-5. Floating-Point Instructions

Primary	Extended	Mnemonic	Unit	Cycles
59	018	**fdivs**[.]	FPU	18^
59	020	**fsubs**[.]	FPU	1-1-1^
59	021	**fadds**[.]	FPU	1-1-1^
59	024	**fres**[.]	FPU	18^
59	025	**fmuls**[.]	FPU	1-1-1^
59	028	**fmsubs**[.]	FPU	1-1-1^
59	029	**fmadds**[.]	FPU	1-1-1^
59	030	**fnmsubs**[.]	FPU	1-1-1^
59	031	**fnmadds**[.]	FPU	1-1-1^
63	000	**fcmpu**	FPU	1-1-1^
63	012	**frsp**[.]	FPU	1-1-1^
63	014	**fctiw**[.]	FPU	1-1-1^
63	015	**fctiwz**[.]	FPU	1-1-1^
63	018	**fdiv**[.]	FPU	33^

Table 6-5. Floating-Point Instructions (Continued)

Primary	Extended	Mnemonic	Unit	Cycles
63	020	**fsub**[.]	FPU	1-1-1^
63	021	**fadd**[.]	FPU	1-1-1^
63	023	**fsel**[.]	FPU	1-1-1^
63	025	**fmul**[.]	FPU	2-1-1^
63	026	**frsqrte**[.]	FPU	1-1-1^
63	028	**fmsub**[.]	FPU	2-1-1^
63	029	**fmadd**[.]	FPU	2-1-1^
63	030	**fnmsub**[.]	FPU	2-1-1^
63	031	**fnmadd**[.]	FPU	2-1-1^
63	032	**fcmpo**	FPU	1-1-1^
63	038	**mtfsb1**[.]	FPU	1-1-1&^
63	040	**fneg**[.]	FPU	1-1-1^
63	064	**mcrfs**	FPU	1-1-1&
63	070	**mtfsb0**[.]	FPU	1-1-1&^
63	072	**fmr**[.]	FPU	1-1-1^
63	134	**mtfsfi**[.]	FPU	1 1 1&^
63	136	**fnabs**[.]	FPU	1-1-1^
63	264	**fabs**[.]	FPU	1-1-1^
63	583	**mffs**[.]	FPU	1-1-1&^
63	711	**mtfsf**[.]	FPU	1-1-1&^

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

Cycle times marked with “^” immediately forward their CR results to the BPU for fast branch resolution.

Cycle times marked with a “-” specify the number of clock cycles in each pipeline stage. Instructions with a single entry in the cycles column are not pipelined.

Table 6-6 provides latencies for the load and store instructions.

Table 6-6. Load and Store Instructions

Primary	Extended	Mnemonic	Unit	Cycles
31	020	**lwarx**	LSU	2:1
31	023	**lwzx**	LSU	2:1
31	054	**dcbst**	LSU	2/5&
31	055	**lwzux**	LSU	2:1
31	086	**dcbf**	LSU	2/5&
31	087	**lbzx**	LSU	2:1
31	119	**lbzux**	LSU	2:1
31	150	**stwcx.**	LSU	8
31	151	**stwx**	LSU	2:1
31	183	**stwux**	LSU	2:1
31	215	**stbx**	LSU	2:1
31	246	**dcbtst**	LSU	2
31	247	**stbux**	LSU	2:1
31	278	**dcbt**	LSU	2
31	279	**lhzx**	LSU	2:1
31	306	**tlbie**	LSU	3&
31	310	**eciwx**	LSU	2:1
31	311	**lhzux**	LSU	2:1
31	343	**lhax**	LSU	2:1
31	375	**lhaux**	LSU	2:1
31	407	**sthx**	LSU	2:1
31	438	**ecowx**	LSU	2:1
31	439	**sthux**	LSU	2:1
31	470	**dcbi**	LSU	2&
31	533	**lswx**	LSU	2 + n&
31	534	**lwbrx**	LSU	2:1
31	535	**lfsx**	LSU	2:1
31	566	**tlbsync**	LSU	2&
31	567	**lfsux**	LSU	2:1
31	597	**lswi**	LSU	2 + n&
31	599	**lfdx**	LSU	2:1

Table 6-6. Load and Store Instructions (Continued)

Primary	Extended	Mnemonic	Unit	Cycles
31	631	**lfdux**	LSU	2:1
31	661	**stswx**	LSU	1 + n&
31	662	**stwbrx**	LSU	2:1
31	663	**stfsx**	LSU	2:1
31	695	**stfsux**	LSU	2:1
31	725	**stswi**	LSU	1 + n&
31	727	**stfdx**	LSU	2:1
31	759	**stfdux**	LSU	2:1
31	790	**lhbrx**	LSU	2:1
31	918	**sthbrx**	LSU	2:1
31	978	**tlbld**	LSU	2&
31	982	**icbi**	LSU	3&
31	983	**stfiwx**	LSU	2:1
31	1010	**tlbli**	LSU	3&
31	1014	**dcbz**	LSU	10&
32	---	**lwz**	LSU	2:1
33	---	**lwzu**	LSU	2:1
34	---	**lbz**	LSU	2:1
35	---	**lbzu**	LSU	2:1
36	---	**stw**	LSU	2:1
37	---	**stwu**	LSU	2:1
38	---	**stb**	LSU	2:1
39	---	**stbu**	LSU	2:1
40	---	**lhz**	LSU	2:1
41	---	**lhzu**	LSU	2:1
42	---	**lha**	LSU	2:1
43	---	**lhau**	LSU	2:1
44	---	**sth**	LSU	2:1
45	---	**sthu**	LSU	2:1
46	---	**lmw**	LSU	2 + n&
47	---	**stmw**	LSU	1 + n&
48	---	**lfs**	LSU	2:1

Table 6-6. Load and Store Instructions (Continued)

Primary	Extended	Mnemonic	Unit	Cycles
49	---	**lfsu**	LSU	2:1
50	---	**lfd**	LSU	2:1
51	---	**lfdu**	LSU	2:1
52	---	**stfs**	LSU	2:1
53	---	**stfsu**	LSU	2:1
54	---	**stfd**	LSU	2:1
55	---	**stfdu**	LSU	2:1

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

Cycle times marked with a “/”specify hit and miss times for cache management instructions that require conditional bus activity.

Cycle times marked with a “:” specify cycles of total latency and throughput for pipelined load and store instructions.

Load and store multiple and string instruction cycles are shown as a fixed number of cycles plus a variable number of cycles where “n” is the number of words accessed by the instruction.

Chapter 7
Signal Descriptions

This chapter describes the PowerPC 603e microprocessor's external signals. It contains a concise description of individual signals, showing behavior when the signal is asserted and negated and when the signal is an input and an output.

NOTE

A bar over a signal name indicates that the signal is active low—for example, $\overline{\text{ARTRY}}$ (address retry) and $\overline{\text{TS}}$ (transfer start). Active-low signals are referred to as asserted (active) when they are low and negated when they are high. Signals that are not active-low, such as 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 603e signals are grouped as follows:

- Address arbitration signals—The 603e uses these signals to arbitrate for address bus mastership.
- Address transfer start signals—These signals indicate that a bus master has begun a transaction on the address bus.
- Address transfer signals—These signals, which consist of the address bus, address parity, and address parity error signals, are used to transfer the address and to ensure the integrity of the transfer.
- Transfer attribute signals—These signals provide information about the type of transfer, such as the transfer size and whether the transaction is bursted, write-through, or cache-inhibited.
- Address transfer termination signals—These signals are used to acknowledge the end of the address phase of the transaction. They also indicate whether a condition exists that requires the address phase to be repeated.
- Data arbitration signals—The 603e uses these signals to arbitrate for data bus mastership.
- Data transfer signals—These signals, which consist of the data bus, data parity, and data parity error signals, are used to transfer the data and to ensure the integrity of the transfer.

- Data transfer termination signals—Data termination signals are required after each data beat in a data transfer. In a single-beat transaction, the data termination signals also indicate the end of the tenure, while in burst accesses, the data termination signals apply to individual beats and indicate the end of the tenure only after the final data beat. They also indicate whether a condition exists that requires the data phase to be repeated.
- System status signals—These signals include the external interrupt signal, checkstop signals, and both soft- and hard-reset signals. These signals are used to interrupt and, under various conditions, to reset the processor.
- JTAG/COP interface signals—The JTAG (IEEE 1149.1) interface and common on-chip processor (COP) unit provides a serial interface to the system for performing monitoring and boundary tests.
- Processor status—These signals include the memory $\overline{\text{reservation}}$ signal, machine quiesce control signals, time base enable signal, and $\overline{\text{TLBISYNC}}$ signal.
- Clock signals—These signals provide for system clock input and frequency control.

7.1 Signal Configuration

Figure 7-1 illustrates the 603e microprocessor's signal configuration, showing how the signals are grouped.

NOTE

A pinout showing actual pin numbers is included in the 603e hardware specifications.

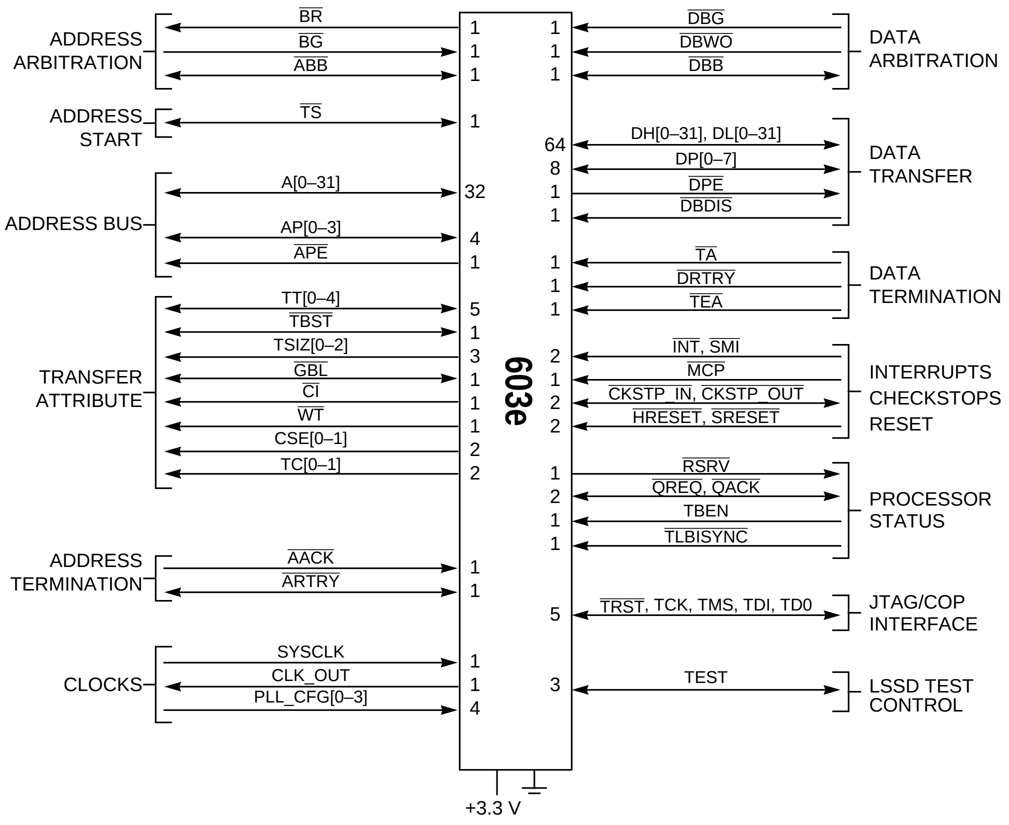


Figure 7-1. Signal Groups

7.2 Signal Descriptions

This section describes individual 603e signals, grouped according to Figure 7-1. Note that the following sections are intended to provide a quick summary of signal functions. Chapter 8, “System Interface Operation,” describes many of these signals in greater detail, both with respect to how individual signals function and how groups of signals interact.

7.2.1 Address Bus Arbitration Signals

The address arbitration signals are a collection of input and output signals the 603e uses to request the address bus, recognize when the request is granted, and indicate to other devices when mastership is granted. For a detailed description of how these signals interact, see Section 8.3.1, “Address Bus Arbitration.”

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

The bus request ($\overline{\text{BR}}$) signal is an output signal on the 603e. Following are the state meaning and timing comments for the $\overline{\text{BR}}$ signal.

State Meaning	Asserted—Indicates that the 603e is requesting mastership of the address bus. Note that $\overline{\text{BR}}$ may be asserted for one or more cycles, and then de-asserted due to an internal cancellation of the bus request (for example, due to a load hit in the touch load buffer). See Section 8.3.1, “Address Bus Arbitration.” Negated—Indicates that the 603e is not requesting the address bus. The 603e 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—Occurs when the 603e is not parked and a bus transaction is needed. This may occur even if the two possible pipeline accesses have occurred. $\overline{\text{BR}}$ will also be asserted for one cycle during the execution of a **dcbz** instruction, and during the execution of a load instruction which 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 bus clock cycle when the assertion of $\overline{\text{ARTRY}}$ is detected on the bus.

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

The bus grant ($\overline{\text{BG}}$) signal is an input signal on the 603e. Following are the state meaning and timing comments for the $\overline{\text{BG}}$ signal.

State Meaning Asserted—Indicates that the 603e may, with the proper qualification, assume mastership of the address bus. A qualified bus grant occurs when $\overline{\text{BG}}$ is asserted and $\overline{\text{ABB}}$ and $\overline{\text{ARTRY}}$ (after $\overline{\text{AACK}}$) are not asserted. The $\overline{\text{ABB}}$ and $\overline{\text{ARTRY}}$ signals are driven by the 603e or other bus masters. If the 603e is parked, $\overline{\text{BR}}$ need not be asserted for the qualified bus grant. See Section 8.3.1, "Address Bus Arbitration."

Negated— Indicates that the 603e is not the next potential address bus master.

Timing Comments Assertion—May occur at any time to indicate the 603e is free to use the address bus. After the 603e assumes bus mastership, it does not check for a qualified bus grant again until the cycle during which the address bus tenure is completed (assuming it has another transaction to run). The 603e does not accept a $\overline{\text{BG}}$ in the cycles between the assertion of any $\overline{\text{TS}}$ and $\overline{\text{AACK}}$.

Negation—May occur at any time to indicate the 603e cannot use the bus. The 603e may still assume bus mastership on the bus clock cycle of the negation of $\overline{\text{BG}}$ because during the previous cycle $\overline{\text{BG}}$ indicated to the 603e that it was free to take mastership (if qualified).

7.2.1.3 Address Bus Busy ($\overline{\text{ABB}}$)

The address bus busy ($\overline{\text{ABB}}$) signal is both an input and an output signal.

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

Following are the state meaning and timing comments for the $\overline{\text{ABB}}$ output signal.

State Meaning Asserted—Indicates that the 603e is the address bus master. See Section 8.3.1, "Address Bus Arbitration."

Negated—Indicates that the 603e is not using the address bus. If $\overline{\text{ABB}}$ is negated during the bus clock cycle following a qualified bus grant, the 603e did not accept mastership, even if $\overline{\text{BR}}$ was asserted. This can occur if a potential transaction is aborted internally before the transaction is started.

Timing Comments Assertion—Occurs on the bus clock cycle following a qualified $\overline{\text{BG}}$ that is accepted by the processor (see Negated).

Negation—Occurs for a minimum of one-half bus clock cycle following the assertion of $\overline{\text{AACK}}$. If $\overline{\text{ABB}}$ is negated during the bus clock cycle following a qualified bus grant, the 603e did not accept mastership, even if $\overline{\text{BR}}$ was asserted.

High Impedance—Occurs after $\overline{\text{ABB}}$ is negated.

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

Following are the state meaning and timing comments for the $\overline{\text{ABB}}$ input signal.

State Meaning Asserted—Indicates that the address bus is in use. This condition effectively blocks the 603e from assuming address bus ownership, regardless of the $\overline{\text{BG}}$ input; see Section 8.3.1, “Address Bus Arbitration.”

Negated—Indicates that the address bus is not owned by another bus master and that it is available to the 603e when accompanied by a qualified bus grant.

Timing Comments Assertion—May occur when the 603e must be prevented from using the address bus (and the processor is not currently asserting $\overline{\text{ABB}}$).

Negation—May occur whenever the 603e can use the address bus.

7.2.2 Address Transfer Start Signals

Address transfer start signals are input and output signals that indicate that an address bus transfer has begun. The transfer start ($\overline{\text{TS}}$) signal identifies the operation as a memory transaction.

For detailed information about how $\overline{\text{TS}}$ interacts with other signals, refer to Section 8.3.2, “Address Transfer.”

7.2.2.1 Transfer Start ($\overline{\text{TS}}$)

The $\overline{\text{TS}}$ signal is both an input and an output signal on the 603e.

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

Following are the state meaning and timing comments for the $\overline{\text{TS}}$ output signal.

State Meaning Asserted—Indicates that the 603e has begun a memory bus transaction and that the address bus and transfer attribute signals are valid. When asserted with the appropriate TT[0–4] signals it is also an implied data bus request for a memory transaction (unless it is an address-only operation).

Negated—Indicates that no bus transaction is occurring during normal operation.

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—Coincides with the negation of $\overline{\text{ABB}}$.

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

Following are the state meaning and timing comments for the $\overline{\text{TS}}$ input signal.

State Meaning	Asserted—Indicates that another master has begun a bus transaction and that the address bus and transfer attribute signals are valid for snooping (see $\overline{\text{GBL}}$).
	Negated—Indicates that no bus transaction is occurring.
Timing Comments	Assertion—May occur during the assertion of $\overline{\text{ABB}}$. Negation—Must occur one bus clock cycle after $\overline{\text{TS}}$ is asserted.

7.2.3 Address Transfer Signals

The address transfer signals are used to transmit the address and to generate and monitor parity for the address transfer. For a detailed description of how these signals interact, refer to Section 8.3.2, “Address Transfer.”

7.2.3.1 Address Bus (A[0–31])

The address bus (A[0–31]) consists of 32 signals that are both input and output signals.

7.2.3.1.1 Address Bus (A[0–31])—Output

Following are the state meaning and timing comments for the A[0–31] output signals.

State Meaning	Asserted/Negated—Represents the physical address (real address in the architecture specification) of the data to be transferred. On burst transfers, the address bus presents the double-word–aligned address containing the critical code/data that missed the cache on a read operation, or the first double word of the cache line on a write operation. Note that the address output during burst operations is not incremented. See Section 8.3.2, “Address Transfer.”
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.

7.2.3.1.2 Address Bus (A[0–31])—Input

Following are the state meaning and timing comments for the A[0–31] input signals.

State Meaning	Asserted/Negated—Represents the physical 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 603e only on this cycle.

7.2.3.2 Address Bus Parity (AP[0–3])

The address bus parity (AP[0–3]) signals are both input and output signals reflecting one bit of odd-byte parity for each of the 4 bytes of address when a valid address is on the bus.

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

Following are the state meaning and timing comments for the AP[0–3] output signal on the 603e.

State Meaning Asserted/Negated—Represents odd parity for each of 4 bytes of the physical address for a transaction. Odd parity means that an odd number of bits, including the parity bit, are driven high. The signal assignments correspond to the following:

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

For more information, see Section 8.3.2.1, “Address Bus Parity.”

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

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

Following are the state meaning and timing comments for the AP[0–3] input signal on the 603e.

State Meaning Asserted/Negated—Represents odd parity for each of 4 bytes of the physical address for snooping operations. Detected even parity causes the processor to take a machine check exception or enter the checkstop state if address parity checking is enabled in the HID0 register; see Section 2.1.2.1, “Hardware Implementation Registers (HID0 and HID1).” (See also the $\overline{\text{APE}}$ signal description.)

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

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

The address parity error ($\overline{\text{APE}}$) signal is an output signal on the 603e. Note that the ($\overline{\text{APE}}$) signal is an open-drain type output, and requires an external pull-up resistor (for example, 10 kΩ to Vdd) to assure proper de-assertion of the $\overline{\text{APE}}$ signal. Following are the state meaning and timing comments for the $\overline{\text{APE}}$ signal on the 603e. The $\overline{\text{APE}}$ signal will not be asserted if address parity checking is disabled (HID0[EBA] cleared to 0). For more information, see Section 8.3.2.1, “Address Bus Parity.”

State Meaning Asserted—Indicates incorrect address bus parity has been detected by the 603e on a snoop ($\overline{\text{GBL}}$ asserted).

Negated—Indicates that the 603e has not detected a parity error (even parity) on the address bus.

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

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

7.2.4 Address Transfer Attribute Signals

The transfer attribute signals are a set of signals that further characterize the transfer—such as the size of the transfer, whether it is a read or write operation, and whether it is a burst or single-beat transfer. For a detailed description of how these signals interact, see Section 8.3.2, “Address Transfer.”

Note that some signal functions vary depending on whether the transaction is a memory access or an I/O access.

7.2.4.1 Transfer Type (TT[0–4])

The transfer type (TT[0–4]) signals consist of five input/output signals on the 603e. For a complete description of TT[0–4] signals and for transfer type encodings, see Table 7-1.

7.2.4.1.1 Transfer Type (TT[0–4])—Output

Following are the state meaning and timing comments for the TT[0–4] output signals on the 603e.

State Meaning Asserted/Negated—Indicates the type of transfer in progress.

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

7.2.4.1.2 Transfer Type (TT[0–4])—Input

Following are the state meaning and timing comments for the TT[0–3] input signals on the 603e.

State Meaning Asserted/Negated—Indicates the type of transfer in progress (see Table 7-2).

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

Table 7-1 describes the transfer encodings for a 603e bus master.

Table 7-1. Transfer Encoding for the Bus Master

603e Bus Master Transaction	Transaction Source	TT0	TT1	TT2	TT3	TT4	60x Bus Specification Command	Transaction
N/A	N/A	0	0	0	0	0	Clean block	Address only
N/A	N/A	0	0	1	0	0	Flush block	Address only
N/A	N/A	0	1	0	0	0	**sync**	Address only
Address only	**dcbz**	0	1	1	0	0	Kill block	Address only
N/A	N/A	1	0	0	0	0	**eieio**	Address only

Table 7-1. Transfer Encoding for the Bus Master (Continued)

603e Bus Master Transaction	Transaction Source	TT0	TT1	TT2	TT3	TT4	60x Bus Specification Command	Transaction
Single-beat write (non$\overline{\text{GBL}}$)	**ecowx**	1	0	1	0	0	External control word write	Single-beat write
N/A	N/A	1	1	0	0	0	TLB invalidate	Address only
Single-beat read (non$\overline{\text{GBL}}$)	**eciwx**	1	1	1	0	0	External control word read	Single-beat read
N/A	N/A	0	0	0	0	1	**lwarx** Reservation set	Address only
N/A	N/A	0	0	1	0	1	Reserved	—
N/A	N/A	0	1	0	0	1	**tlbsync**	Address only
N/A	N/A	0	1	1	0	1	**icbi**	Address only
N/A	N/A	1	X	X	0	1	Reserved	—
Single-beat write	Caching-inhibited or write-through store	0	0	0	1	0	Write-with-flush	Single-beat write or burst
Burst (non$\overline{\text{GBL}}$)	Cast-out, or snoop copyback	0	0	1	1	0	Write-with-kill	Single-beat write or burst
Single-beat read	Caching-inhibited load or instruction fetch	0	1	0	1	0	Read	Single-beat read or burst
Burst	Load miss, store miss, or instruction fetch	0	1	1	1	0	Read-with-intent-to-modify	Burst
Single-beat write	**stwcx.**	1	0	0	1	0	Write-with-flush-atomic	Single-beat write
N/A	N/A	1	0	1	1	0	Reserved	N/A
Single-beat read	**lwarx** (caching-inhibited load)	1	1	0	1	0	Read-atomic	Single-beat read or burst
Burst	**lwarx** (load miss)	1	1	1	1	0	Read-with-intent-to-modify-atomic	Burst
N/A	N/A	0	0	0	1	1	Reserved	—
N/A	N/A	0	0	1	1	1	Reserved	—
N/A	N/A	0	1	0	1	1	Read-with-no-intent-to-cache	Single-beat read or burst
N/A	N/A	0	1	1	1	1	Reserved	—
N/A	N/A	1	X	X	1	1	Reserved	—

Table 7-2 describes the 60x bus specification transfer encodings and the 603e bus snoop response on an address hit.

Table 7-2. Snoop Hit Response

60x Bus Specification Command	Transaction	TT0	TT1	TT2	TT3	TT4	603e Bus Snooper; Action on Hit
Clean block	Address only	0	0	0	0	0	N/A
Flush block	Address only	0	0	1	0	0	N/A
sync	Address only	0	1	0	0	0	N/A
Kill block	Address only	0	1	1	0	0	Kill, cancel reservation
eieio	Address only	1	0	0	0	0	N/A
External control word write	Single-beat write	1	0	1	0	0	N/A
TLB Invalidate	Address only	1	1	0	0	0	N/A
External control word read	Single-beat read	1	1	1	0	0	N/A
lwarx Reservation set	Address only	0	0	0	0	1	N/A
Reserved	—	0	0	1	0	1	N/A
tlbsync	Address only	0	1	0	0	1	N/A
icbi	Address only	0	1	1	0	1	N/A
Reserved	—	1	X	X	0	1	N/A
Write-with-flush	Single-beat write or burst	0	0	0	1	0	Flush, cancel reservation
Write-with-kill	Single-beat write or burst	0	0	1	1	0	Kill, cancel reservation
Read	Single-beat read or burst	0	1	0	1	0	Clean or flush
Read-with-intent-to-modify	Burst	0	1	1	1	0	Flush
Write-with-flush-atomic	Single-beat write	1	0	0	1	0	Flush, cancel reservation
Reserved	N/A	1	0	1	1	0	N/A
Read-atomic	Single-beat read or burst	1	1	0	1	0	Clean or flush
Read-with-intent-to modify-atomic	Burst	1	1	1	1	0	Flush
Reserved	—	0	0	0	1	1	N/A
Reserved	—	0	0	1	1	1	N/A
Read-with-no-intent-to-cache	Single-beat read or burst	0	1	0	1	1	Clean
Reserved	—	0	1	1	1	1	N/A
Reserved	—	1	X	X	1	1	N/A

The 603e provides transfer type signals (TT[0–4]) that characterize bus transfers. When HID0[ABE] is set, the PID7v-603e performs address-only bus transactions with the encodings shown in Table 7-3.

Table 7-3. Implementation-Specific Transfer Encoding

TT0	TT1	TT2	TT3	TT4	PID7v-603e Transaction	Transaction	Transaction Source
0	0	0	0	0	Clean block	Address only	**dcbst**
0	0	1	0	0	Flush block	Address only	**dcbf**
0	1	1	0	0	Kill block	Address only	**dcbz**, **dcbi**

The 603e provides a CLK_OUT signal for test purposes that allows the monitoring of the processor and bus clock frequencies. The frequency of the CLK_OUT signal is determined by the configuration of the HID0[SBCLK] and HID0[ECLK] bits, as shown in Table 7-4. Note that the PID7v-603e's CLK_OUT signal will be driven at the processor frequency during the assertion of $\overline{\text{HRESET}}$; when the $\overline{\text{HRESET}}$ signal is deasserted, the CLK_OUT signal enters the default high-impedance state.

Table 7-4. CLK_OUT Signal Configuration

HID0[SBCLK]	HID0[ECLK]	CLK_OUT Output State
0	0	High-impedance
0	1	Processor clock frequency
1	0	Half-bus clock frequency
1	1	Bus clock frequency

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

The transfer size (TSIZ[0–2]) signals consist of three output signals on the 603e. Following are the state meaning and timing comments for the TSIZ[0–2] output signals on the 603e.

State Meaning Asserted/Negated—For memory accesses, these signals along with $\overline{\text{TBST}}$, indicate the data transfer size for the current bus operation, as shown in Table 7-5. Table 8-4 shows how the transfer size signals are used with the address signals for aligned transfers. Table 8-5 shows how the transfer size signals are used with the address signals for misaligned transfers.

For external control instructions (**eciwx** and **ecowx**), TSIZ[0–2] are used to output bits 29–31 of the external access register (EAR), which are used to form the resource ID ($\overline{\text{TBST}}$||TSIZ[0–2]).

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

Table 7-5. Data Transfer Size

$\overline{\text{TBST}}$	TSIZ[0–2]	Transfer Size
Asserted	010	Burst (32 bytes)
Negated	000	8 bytes
Negated	001	1 byte
Negated	010	2 bytes
Negated	011	3 bytes
Negated	100	4 bytes
Negated	101	5 bytes
Negated	110	6 bytes
Negated	111	7 bytes

7.2.4.3 Transfer Burst ($\overline{\text{TBST}}$)

The transfer burst ($\overline{\text{TBST}}$) signal is an input/output signal on the 603e.

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

Following are the state meaning and timing comments for the $\overline{\text{TBST}}$ output signal.

State Meaning Asserted—Indicates that a burst transfer is in progress.

Negated—Indicates that a burst transfer is not in progress.

For external control instructions (**eciwx** and **ecowx**), $\overline{\text{TBST}}$ is used to output bit 28 of the EAR, which is used to form the resource ID ($\overline{\text{TBST}}$||TSIZ[0–2]).

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

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

Following are the state meaning and timing comments for the $\overline{\text{TBST}}$ input signal.

State Meaning Asserted/Negated—Used when snooping for single-beat reads (read with no intent to cache).

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

7.2.4.4 Transfer Code (TC[0–1])—Output

The transfer code (TC[0–1]) consists of two output signals on the 603e. Following are the state meaning and timing comments for the TC[0–1] signals.

State Meaning Asserted/Negated—Represents a special encoding for the transfer in progress (see Table 7-6).

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

Table 7-6. Encodings for TC[0–1] Signals

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

7.2.4.5 Cache Inhibit ($\overline{\text{CI}}$)—Output

The cache inhibit ($\overline{\text{CI}}$) signal is an output signal on the 603e. Following are the state meaning and timing comments for the $\overline{\text{CI}}$ signal.

State Meaning Asserted—Indicates that a single-beat transfer will not be cached, reflecting the setting of the I bit for the block or page that contains the address of the current transaction.

Negated—Indicates that a burst transfer will allocate a line in the 603e data cache.

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

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

The write-through ($\overline{\text{WT}}$) signal is an output signal on the 603e. Following are the state meaning and timing comments for the $\overline{\text{WT}}$ signal.

State Meaning Asserted—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—Indicates that a transaction is not write-through.

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

7.2.4.7 Global ($\overline{\text{GBL}}$)

The global ($\overline{\text{GBL}}$) signal is an input/output signal on the 603e.

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

Following are the state meaning and timing comments for the $\overline{\text{GBL}}$ output signal.

State Meaning	Asserted—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 copy-back operations and instruction fetches, which are nonglobal.)
	Negated—Indicates that a transaction is not global.
Timing Comments	Assertion/Negation—The same as A[0–31]. High Impedance—The same as A[0–31].

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

Following are the state meaning and timing comments for the $\overline{\text{GBL}}$ input signal.

State Meaning	Asserted—Indicates that a transaction must be snooped by the 603e.
	Negated—Indicates that a transaction is not snooped by the 603e.
Timing Comments	Assertion/Negation—The same as A[0–31].

7.2.4.8 Cache Set Entry (CSE[0–1])—Output

Following are the state meaning and timing comments for the CSE[0–1] signals.

State Meaning	Asserted/Negated—Represents the cache replacement set element for the current transaction reloading into or writing out of the cache. Can be used with the address bus and the transfer attribute signals to externally track the state of each cache line in the 603e's cache. Note that the CSE[0–1] signals are not meaningful during data cache touch load operations.
Timing Comments	Assertion/Negation—The same as A[0–31]. High Impedance—The same as A[0–31].

7.2.5 Address Transfer Termination Signals

The address transfer termination signals are used to indicate either that the address phase of the transaction has completed successfully or must be repeated, and when it should be terminated. For detailed information about how these signals interact, see Section 8.3.3, "Address Transfer Termination."

7.2.5.1 Address Acknowledge ($\overline{\text{AACK}}$)—Input

The address acknowledge ($\overline{\text{AACK}}$) signal is an input signal (input-only) on the 603e. Following are the state meaning and timing comments for the $\overline{\text{AACK}}$ signal.

State Meaning	Asserted—Indicates that the address phase of a transaction is complete. The address bus will go to a high impedance state on the next bus clock cycle. The 603e samples $\overline{\text{ARTRY}}$ on the bus clock cycle following the assertion of $\overline{\text{AACK}}$.
	Negated—(During $\overline{\text{ABB}}$) indicates that the address bus and the transfer attributes must remain driven.
Timing Comments	Assertion—May occur as early as the bus clock cycle after $\overline{\text{TS}}$ is asserted (unless 603e is configured for 1:1 or 1.5:1 clock modes, when $\overline{\text{AACK}}$ can be asserted no sooner than the second cycle following the assertion of $\overline{\text{TS}}$—one address wait state); assertion can be delayed to allow adequate address access time for slow devices. For example, if an implementation supports slow snooping devices, an external arbiter can postpone the assertion of $\overline{\text{AACK}}$.
	Negation—Must occur one bus clock cycle after the assertion of $\overline{\text{AACK}}$.

7.2.5.2 Address Retry ($\overline{\text{ARTRY}}$)

The address retry ($\overline{\text{ARTRY}}$) signal is both an input and output signal on the 603e.

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

Following are the state meaning and timing comments for the $\overline{\text{ARTRY}}$ output signal.

State Meaning	Asserted—Indicates that the 603e detects a condition in which a snooped address tenure must be retried. If the 603e needs to update memory as a result of the snoop that caused the retry, the 603e asserts $\overline{\text{BR}}$ the second cycle after $\overline{\text{AACK}}$ if $\overline{\text{ARTRY}}$ is asserted.
	High Impedance—Indicates that the 603e does not need the snooped address tenure to be retried.
Timing Comments	Assertion—Asserted the third bus cycle following the assertion of $\overline{\text{TS}}$ if a retry is required.
	Negation—Occurs the second bus cycle after the assertion of $\overline{\text{AACK}}$. Since this signal 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 special method of negation may be disabled by setting precharge disable in HID0.

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

Following are the state meaning and timing comments for the $\overline{\text{ARTRY}}$ input signal.

State Meaning Asserted—If the 603e is the address bus master, $\overline{\text{ARTRY}}$ indicates that the 603e must retry the preceding address tenure and immediately negate $\overline{\text{BR}}$ (if asserted). If the associated data tenure has already started, the 603e will also abort the data tenure immediately, even if the burst data has been received. If the 603e is not the address bus master, this input indicates that the 603e should immediately negate $\overline{\text{BR}}$ for one bus clock cycle following the assertion of $\overline{\text{ARTRY}}$ by the snooping bus master to allow an opportunity for a copy-back operation to main memory. Note that the subsequent address presented on the address bus may not be the same one associated with the assertion of the $\overline{\text{ARTRY}}$ signal.

Negated/High Impedance—Indicates that the 603e does not need to retry the last address tenure.

Timing Comments Assertion—May occur as early as the second cycle following the assertion of $\overline{\text{TS}}$, and must occur by the bus clock cycle immediately following the assertion of $\overline{\text{AACK}}$ if an address retry is required.

Negation—Must occur during the second cycle after the assertion of $\overline{\text{AACK}}$.

7.2.6 Data Bus Arbitration Signals

Like the address bus arbitration signals, data bus arbitration signals maintain an orderly process for determining data bus mastership. Note that there is no data bus arbitration signal equivalent to the address bus arbitration signal $\overline{\text{BR}}$ (bus request), because, except for address-only transactions, $\overline{\text{TS}}$ implies data bus requests. For a detailed description on how these signals interact, see Section 8.4.1, “Data Bus Arbitration.”

One special signal, $\overline{\text{DBWO}}$, allows the 603e to be configured dynamically to write data out of order with respect to read data. For detailed information about using $\overline{\text{DBWO}}$, see Section 8.10, “Using Data Bus Write Only.”

7.2.6.1 Data Bus Grant ($\overline{\text{DBG}}$)—Input

The data bus grant ($\overline{\text{DBG}}$) signal is an input signal (input-only) on the 603e. Following are the state meaning and timing comments for the $\overline{\text{DBG}}$ signal.

State Meaning Asserted—Indicates that the 603e may, with the proper qualification, assume mastership of the data bus. The 603e derives a qualified data bus grant when $\overline{\text{DBG}}$ is asserted and $\overline{\text{DBB}}$, $\overline{\text{DRTRY}}$, and $\overline{\text{ARTRY}}$ are negated; that is, the data bus is not busy ($\overline{\text{DBB}}$ is negated), there is no outstanding attempt to retry the current data tenure ($\overline{\text{DRTRY}}$ is negated), and there is no outstanding attempt to perform an $\overline{\text{ARTRY}}$ of the associated address tenure.

Negated—Indicates that the 603e must hold off its data tenures.

Timing Comments Assertion—May occur any time to indicate the 603e is free to take data bus mastership. It is not sampled until $\overline{\text{TS}}$ is asserted.

Negation—May occur at any time to indicate the 603e cannot assume data bus mastership.

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

The data bus write only ($\overline{\text{DBWO}}$) signal is an input signal (input-only) on the 603e. Following are the state meaning and timing comments for the $\overline{\text{DBWO}}$ signal.

State Meaning Asserted—Indicates that the 603e may run the data bus tenure for an outstanding write address even if a read address is pipelined before the write address. Refer to Section 8.10, "Using Data Bus Write Only," for detailed instructions for using $\overline{\text{DBWO}}$.

Negated—Indicates that the 603e must run the data bus tenures in the same order as the address tenures.

Timing Comments Assertion—Must occur no later than a qualified $\overline{\text{DBG}}$ for an outstanding write tenure. $\overline{\text{DBWO}}$ is only recognized by the 603e on the clock of a qualified $\overline{\text{DBG}}$. If no write requests are pending, the 603e will ignore $\overline{\text{DBWO}}$ and assume data bus ownership for the next pending read request.

Negation—May occur any time after a qualified $\overline{\text{DBG}}$ and before the next assertion of $\overline{\text{DBG}}$.

7.2.6.3 Data Bus Busy ($\overline{\text{DBB}}$)

The data bus busy ($\overline{\text{DBB}}$) signal is both an input and output signal on the 603e.

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

Following are the state meaning and timing comments for the $\overline{\text{DBB}}$ output signal.

State Meaning Asserted—Indicates that the 603e is the data bus master. The 603e always assumes data bus mastership if it needs the data bus and is given a qualified data bus grant (see $\overline{\text{DBG}}$).

Negated—Indicates that the 603e is not using the data bus.

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

Negation—Occurs for a minimum of one-half bus clock cycle (dependent on clock mode) following the assertion of the final $\overline{\text{TA}}$.

High Impedance—Occurs after $\overline{\text{DBB}}$ is negated.

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

Following are the state meaning and timing comments for the $\overline{\text{DBB}}$ input signal.

State Meaning Asserted—Indicates that another device is bus master.
Negated—Indicates that the data bus is free (with proper qualification, see $\overline{\text{DBG}}$) for use by the 603e.

Timing Comments Assertion—Must occur when the 603e must be prevented from using the data bus.

Negation—May occur whenever the data bus is available.

7.2.7 Data Transfer Signals

Like the address transfer signals, the data transfer signals are used to transmit data and to generate and monitor parity for the data transfer. For a detailed description of how the data transfer signals interact, see Section 8.4.3, “Data Transfer.”

7.2.7.1 Data Bus (DH[0–31], DL[0–31])

The data bus (DH[0–31] and DL[0–31]) consists of 64 signals that are both input and output on the 603e. Following are the state meaning and timing comments for the DH and DL signals.

State Meaning The data bus has two halves—data bus high (DH) and data bus low (DL). See Table 7-7 for the data bus lane assignments.

Timing Comments The data bus is driven once for noncached transactions and four times for cache transactions (bursts).

Table 7-7. 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

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

Following are the state meaning and timing comments for the DH and DL output signals.

State Meaning Asserted/Negated—Represents the state of data during a data write. Byte lanes not selected for data transfer will not supply valid data.

Timing Comments Assertion/Negation—Initial beat coincides with $\overline{\text{DBB}}$ and, for bursts, transitions on the bus clock cycle following each assertion of $\overline{\text{TA}}$.

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

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

Following are the state meaning and timing comments for the DH and DL input signals.

State Meaning	Asserted/Negated—Represents the state of data during a data read transaction.
Timing Comments	Assertion/Negation—Data must be valid on the same bus clock cycle that $\overline{\text{TA}}$ is asserted.

7.2.7.2 Data Bus Parity (DP[0–7])

The eight data bus parity (DP[0–7]) signals on the 603e are both output and input signals.

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

Following are the state meaning and timing comments for the DP output signals.

State Meaning	Asserted/Negated—Represents odd parity for each of 8 bytes of data write transactions. Odd parity means that an odd number of bits, including the parity bit, are driven high. The signal assignments are listed in Table 7-8.
Timing Comments	Assertion/Negation—The same as DL[0–31]. High Impedance—The same as DL[0–31].

Table 7-8. DP[0–7] Signal Assignments

Signal Name	Signal Assignments
DP0	DH[0–7]
DP1	DH[8–15]
DP2	DH[16–23]
DP3	DH[24–31]
DP4	DL[0–7]
DP5	DL[8–15]
DP6	DL[16–23]
DP7	DL[24–31]

7.2.7.2.2 Data Bus Parity (DP[0–7])—Input

Following are the state meaning and timing comments for the DP input signals.

State Meaning	Asserted/Negated—Represents odd parity for each byte of read data. Parity is checked on all data byte lanes, regardless of the size of the transfer. Detected even parity causes a checkstop if data parity errors are enabled in the HID0 register. (See $\overline{\text{DPE}}$.)
Timing Comments	Assertion/Negation—The same as DL[0–31].

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

The data parity error ($\overline{\text{DPE}}$) signal is an output signal (output-only) on the 603e. Note that the ($\overline{\text{DPE}}$) signal is an open-drain type output, and requires an external pull-up resistor (for example, 10 kΩ to Vdd) to assure proper de-assertion of the ($\overline{\text{DPE}}$) signal. Following are the state meaning and timing comments for the $\overline{\text{DPE}}$ signal.

State Meaning	Asserted—Indicates incorrect data bus parity. Negated—Indicates correct data bus parity.
Timing Comments	Assertion—Occurs on the second bus clock cycle after $\overline{\text{TA}}$ is asserted to the 603e, unless $\overline{\text{TA}}$ is cancelled by an assertion of $\overline{\text{DRTRY}}$. High Impedance—Occurs on the third bus clock cycle after $\overline{\text{TA}}$ is asserted to the 603e.

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

The Data Bus Disable ($\overline{\text{DBDIS}}$) signal is an input signal (input-only) on the 603e. Following are the state meanings and timing comments for the $\overline{\text{DBDIS}}$ signal.

State Meaning	Asserted—Indicates (for a write transaction) that the 603e must release data bus and the data bus parity to high impedance during the following cycle. The data tenure will remain active, $\overline{\text{DBB}}$ will remain driven, and the transfer termination signals will still be monitored by the 603e. Negated—Indicates the data bus should remain normally driven. $\overline{\text{DBDIS}}$ is ignored during read transactions.
Timing Comments	Assertion/Negation—May be asserted on any clock cycle when the 603e is driving, or will be driving the data bus; may remain asserted multiple cycles.

7.2.8 Data Transfer Termination Signals

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

For a detailed description of how these signals interact, see Section 8.4.4, "Data Transfer Termination."

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

The transfer acknowledge ($\overline{\text{TA}}$) signal is an input signal (input-only) on the 603e. Following are the state meaning and timing comments for the $\overline{\text{TA}}$ signal.

State Meaning Asserted—Indicates that a single-beat data transfer completed successfully or that a data beat in a burst transfer completed successfully (unless $\overline{\text{DRTRY}}$ is asserted on the next bus clock cycle). Note that $\overline{\text{TA}}$ must be asserted for each data beat in a burst transaction, and must be asserted during assertion of $\overline{\text{DRTRY}}$. For more information, see Section 8.4.4, "Data Transfer Termination."

Negated—(During $\overline{\text{DBB}}$) indicates that, until $\overline{\text{TA}}$ is asserted, the 603e must continue to drive the data for the current write or must wait to sample the data for reads.

Timing Comments Assertion—Must not occur before $\overline{\text{AACK}}$ for the current transaction (if the address retry mechanism is to be used to prevent invalid data from being used by the processor); otherwise, assertion may occur at any time during the assertion of $\overline{\text{DBB}}$. The system can withhold assertion of $\overline{\text{TA}}$ to indicate that the 603e should insert wait states to extend the duration of the data beat.

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. (Note: when the 603e is configured for 1:1 clock mode and is performing a burst read into the data cache, the 603e requires one wait state between the assertion of $\overline{\text{TS}}$ and the first assertion of $\overline{\text{TA}}$ for that transaction. If no-DRTRY mode is also selected, the 603e requires two wait states for 1:1 clock mode, or 1 wait state for 1.5:1 clock mode.)

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

The data retry ($\overline{\text{DRTRY}}$) signal is input only on the 603e. Following are the state meaning and timing comments for the $\overline{\text{DRTRY}}$ signal.

State Meaning Asserted—Indicates that the 603e must invalidate the data from the previous read operation.

Negated—Indicates that data presented with $\overline{\text{TA}}$ on the previous read operation is valid. Note that $\overline{\text{DRTRY}}$ is ignored for write transactions.

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 may be held asserted for multiple bus clock cycles. When $\overline{\text{DRTRY}}$ 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—The $\overline{\text{DRTRY}}$ signal is sampled at the negation of $\overline{\text{HRESET}}$; if $\overline{\text{DRTRY}}$ is asserted, No-DRTRY mode is selected. If $\overline{\text{DRTRY}}$ is negated at start-up, $\overline{\text{DRTRY}}$ is enabled.

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

The transfer error acknowledge ($\overline{\text{TEA}}$) signal is input only on the 603e. Following are the state meaning and timing comments for the $\overline{\text{TEA}}$ signal.

State Meaning Asserted—Indicates that a bus error occurred. Causes a machine check exception (and possibly causes the processor to enter checkstop state if machine check enable bit is cleared (MSR[ME] = 0)). For more information, see Section 4.5.2.2, “Checkstop State (MSR[ME] = 0).” Assertion terminates the current transaction; that is, assertion of $\overline{\text{TA}}$ and $\overline{\text{DRTRY}}$ are ignored. The assertion of $\overline{\text{TEA}}$ 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. (Note that the term, ‘exception,’ is also referred to as ‘interrupt’ in the architecture specification.)

Negated—Indicates that no bus error was detected.

Timing Comments Assertion—May be asserted while $\overline{\text{DBB}}$ is asserted, and the cycle after $\overline{\text{TA}}$ during a read operation. $\overline{\text{TEA}}$ should be asserted for one cycle only.

Negation—$\overline{\text{TEA}}$ must be negated no later than the negation of $\overline{\text{DBB}}$.

7.2.9 System Status Signals

Most system status signals are input signals that indicate when exceptions are received, when checkstop conditions have occurred, and when the 603e must be reset. The 603e generates the output signal, $\overline{\text{CKSTP\_OUT}}$, when it detects a checkstop condition. For a detailed description of these signals, see Section 8.7, “Interrupt, Checkstop, and Reset Signals.”

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

The interrupt ($\overline{\text{INT}}$) signal is input only. Following are the state meaning and timing comments for the $\overline{\text{INT}}$ signal.

State Meaning Asserted—The 603e initiates an interrupt if MSR[EE] is set; otherwise, the 603e ignores the interrupt. To guarantee that the 603e will take the external interrupt, the $\overline{\text{INT}}$ signal must be held active until the 603e takes the interrupt; otherwise, whether the 603e takes an external interrupt, depends on whether the MSR[EE] bit was set while the $\overline{\text{INT}}$ signal was held active.

Negated—Indicates that normal operation should proceed. See Section 8.7.1, “External Interrupts.”

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 interrupt is taken.

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

The system management interrupt ($\overline{\text{SMI}}$) signal is input only. Following are the state meaning and timing comments for the $\overline{\text{SMI}}$ signal.

State Meaning Asserted—The 603e initiates a system management interrupt operation if the MSR[EE] is set; otherwise, the 603e ignores the exception condition. The 603e must hold the $\overline{\text{SMI}}$ signal active until the exception is taken.

Negated—Indicates that normal operation should proceed. See Section 8.7.1, “External Interrupts.”

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 interrupt is taken.

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

The machine check interrupt ($\overline{\text{MCP}}$) signal is input only on the 603e. Following are the state meaning and timing comments for the $\overline{\text{MCP}}$ signal.

State Meaning Asserted—The 603e 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 603e 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 603e ignores the interrupt condition. The $\overline{\text{MCP}}$ signal must be held asserted for 2 bus clock cycles.

Negated—Indicates that normal operation should proceed. See Section 8.7.1, “External Interrupts.”

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

Negation—May be negated 2 bus cycles after assertion.

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

The checkstop input ($\overline{\text{CKSTP\_IN}}$) signal is input only on the 603e. Following are the state meaning and timing comments for the $\overline{\text{CKSTP\_IN}}$ signal.

State Meaning Asserted—Indicates that the 603e must terminate operation by internally gating off all clocks, and release all outputs (except

	$\overline{\text{CKSTP\_OUT}}$) to the high impedance state. Once $\overline{\text{CKSTP\_IN}}$ has been asserted it must remain asserted until the system has been reset. Negated—Indicates that normal operation should proceed. See Section 8.7.2, “Checkstops.”
Timing Comments	Assertion—May occur at any time and may be asserted asynchronously to the input clocks. Negation—May occur any time after the $\overline{\text{CKSTP\_OUT}}$ output signal has been asserted.

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

The checkstop output ($\overline{\text{CKSTP\_OUT}}$) signal is output only on the 603e. Note that the $\overline{\text{CKSTP\_OUT}}$ signal is an open-drain type output, and requires an external pull-up resistor (for example, 10 kΩ to Vdd) to assure proper de-assertion of the $\overline{\text{CKSTP\_OUT}}$ signal. Following are the state meaning and timing comments for the $\overline{\text{CKSTP\_OUT}}$ signal.

State Meaning	Asserted—Indicates that the 603e has detected a checkstop condition and has ceased operation. Negated—Indicates that the 603e is operating normally. See Section 8.7.2, “Checkstops.”
Timing Comments	Assertion—May occur at any time and may be asserted asynchronously to the 603e input clocks. Negation—Is negated upon assertion of $\overline{\text{HRESET}}$.

7.2.9.6 Reset Signals

There are two reset signals on the 603e—hard reset ($\overline{\text{HRESET}}$) and soft reset ($\overline{\text{SRESET}}$). Descriptions of the reset signals are as follows:

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

The hard reset ($\overline{\text{HRESET}}$) signal is input only and must be used at power-on to properly reset the processor. Following are the state meaning and timing comments for the $\overline{\text{HRESET}}$ signal.

State Meaning	Asserted—Initiates a complete hard reset operation when this input transitions from asserted to negated. Causes a reset exception as described in Section 4.5.1.1, “Hard Reset and Power-On Reset.” Output drivers are released to high impedance within five clocks after the assertion of $\overline{\text{HRESET}}$. Negated—Indicates that normal operation should proceed. See Section 8.7.3, “Reset Inputs.”

Timing Comments Assertion—May occur at any time and may be asserted asynchronously to the 603e input clock; must be held asserted for a minimum of 255 clock cycles after the PLL lock time has been met. Refer to the appropriate hardware specifications for further timing comments.

Negation—May occur any time after the minimum reset pulse width has been met.

This input has additional functionality in certain test modes.

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

The soft reset ($\overline{\text{SRESET}}$) signal is input only. Following are the state meaning and timing comments for the $\overline{\text{SRESET}}$ signal.

State Meaning Asserted— Initiates processing for a reset exception as described in Section 4.5.1.2, “Soft Reset.”

Negated—Indicates that normal operation should proceed. See Section 8.7.3, “Reset Inputs.”

Timing Comments Assertion—May occur at any time and may be asserted asynchronously to the 603e input clock. The $\overline{\text{SRESET}}$ input is negative edge-sensitive.
Negation—May be negated 2 bus cycles after assertion.

This input has additional functionality in certain test modes.

7.2.9.7 Processor Status Signals

Processor status signals indicate the state of the processor. This includes the memory reservation signal, machine quiesce control signals, time base enable signal, and $\overline{\text{TLBISYNC}}$ signal.

7.2.9.7.1 Quiescent Request ($\overline{\text{QREQ}}$)

The quiescent request ($\overline{\text{QREQ}}$) signal is output only. Following are the state meaning and timing comments for the $\overline{\text{QREQ}}$ signal.

State Meaning Asserted—Indicates that the 603e is requesting all bus activity normally required to be snooped to terminate or to pause so the 603e may enter a quiescent (low power) state. Once the 603e has entered a quiescent state, it no longer snoops bus activity.

Negated—Indicates that the 603e is not making a request to enter the quiescent state.

Timing Comments Assertion/Negation—May occur on any cycle. $\overline{\text{QREQ}}$ will remain asserted for the duration of the quiescent state.

7.2.9.7.2 Quiescent Acknowledge ($\overline{\text{QACK}}$)

The quiescent acknowledge ($\overline{\text{QACK}}$) signal is input only. Following are the state meaning and timing comments for the $\overline{\text{QACK}}$ signal.

State Meaning Asserted—Indicates that all bus activity that requires snooping has terminated or paused, and that the 603e may enter the quiescent (or low power) state.

Negated—Indicates that the 603e may not enter a quiescent state, and must continue snooping the bus.

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

Start-Up—$\overline{\text{QACK}}$ is sampled at the negation of $\overline{\text{HRESET}}$ to select reduced-pinout mode; if $\overline{\text{QACK}}$ is asserted at start-up, reduced-pinout mode is disabled.

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

The reservation ($\overline{\text{RSRV}}$) signal is output only on the 603e. Following are the state meaning and timing comments for the $\overline{\text{RSRV}}$ signal.

State Meaning Asserted/Negated—Represents the state of the reservation coherency bit in the reservation address register that is used by the **lwarx** and **stwcx.** instructions. See Section 8.8.1, “Support for the lwarx/stwcx. Instruction Pair.”

Timing Comments Assertion/Negation—Occurs synchronously with respect to bus clock cycles. The execution of an **lwarx** instruction sets the internal reservation condition.

7.2.9.7.4 Time Base Enable (TBEN)—Input

The time base enable (TBEN) signal is input only on the 603e. Following are the state meanings and timing comments for the TBEN signal.

State Meaning Asserted—Indicates that the time base should continue clocking. This input is essentially a “count enable” control for the time base counter.

Negated—Indicates the time base should stop clocking.

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

7.2.9.7.5 TLBI Sync ($\overline{\text{TLBISYNC}}$)

The TLBI Sync ($\overline{\text{TLBISYNC}}$) signal is input only on the 603e. Following are the state meanings and timing comments for the $\overline{\text{TLBISYNC}}$ signal.

State Meaning Asserted—Indicates that instruction execution should stop after execution of a **tlbsync** instruction.

Negated—Indicates that the instruction execution may continue or resume after the completion of a **tlbsync** instruction.

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

Start-Up—$\overline{\text{TLBISYNC}}$ is sampled at the negation of $\overline{\text{HRESET}}$ to select 32-bit data bus mode; if $\overline{\text{TLBISYNC}}$ is negated at start-up, 32-bit mode is disabled and the default 64-bit mode is selected.

7.2.10 COP/Scan Interface

The 603e has extensive on-chip test capability including the following:

- Built-in instruction and data cache self test (BIST)
- Debug control/observation (COP)
- Boundary scan (IEEE 1149.1 compliant interface)
- LSSD test control

The BIST hardware is not exercised as part of the power-on reset (POR) sequence. The COP and boundary scan logic are not used under typical operating conditions.

Detailed discussion of the 603e test functions is beyond the scope of this document; however, sufficient information has been provided to allow the system designer to disable the test functions that would impede normal operation.

The COP/scan interface is shown in Figure 7-2. For more information, see Section 8.9, "IEEE 1149.1-Compliant Interface."

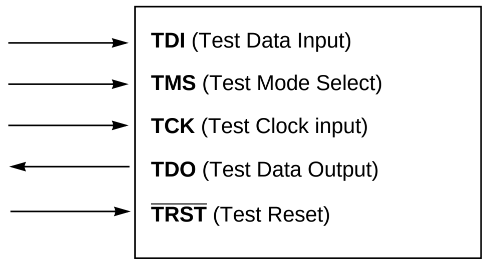

TDI (Test Data Input)
TMS (Test Mode Select)
TCK (Test Clock input)
TDO (Test Data Output)
$\overline{\text{TRST}}$ (Test Reset)

Figure 7-2. IEEE 1149.1-Compliant Boundary Scan Interface

7.2.11 Pipeline Tracking Support

The 603e provides for nonintrusive instruction pipeline tracking. Setting the HID0[EICE] bit causes the address parity and data parity signals to be redefined as outputs providing pipeline tracking information. These signals toggle at the CPU clock rate and will have special loading and timing requirements when in this mode.

Table 7-9 shows the outputs when HID0[EICE] is set.

Table 7-9. Pipeline Tracking Outputs

Bit(s)	Function	Encoding
DP[0–1]	Fetch	00 None 01 Two 10 One 11 Branch
DP[2–3]	Retire	00 None 01 Two 10 One 11 Exception
DP[4–5]	Fold	00 None 01 First 10 Second 11 Both
DP[6–7]	Prediction	00 Nonspec 01 Spec_2nd 10 Spec_both 11 Flush_spec
AP[0–3]	FEA	FEA[20–23]

Given the object code, these signals provide sufficient information to track instruction execution (except for register indirect branches). Register indirect branches may be tracked either by examining and matching potential target streams (nonintrusive but not always resolvable), or by forcing register indirect branch targets to be fetched externally by setting HID0[FBIOB].

Setting HID0[EICE] also enables the processor clock to the CLK_OUT signal which provides a synchronizing clock to the pipeline tracking outputs.

7.2.12 Clock Signals

The clock signal inputs of the 603e determine the system clock frequency and provide a flexible clocking scheme that allows the processor to operate at an integer multiple of the system clock frequency.

Refer to the appropriate hardware specifications for exact timing relationships of the clock signals.

7.2.12.1 System Clock (SYSCLK)—Input

The 603e requires a single system clock (SYSCLK) input. This input sets the frequency of operation for the bus interface. Internally, the 603e uses a phase-locked loop (PLL) circuit to generate a master clock for all of the CPU circuitry (including the bus interface circuitry) which is phase-locked to the SYSCLK input. The master clock may be set to an integer or half-integer multiple (1:1, 1.5:1, 2:1, 2.5:1, 3:1, 3.5:1 or 4:1) of the SYSCLK frequency allowing the CPU core to operate at an equal or greater frequency than the bus interface.

State Meaning	Asserted/Negated—The SYSCLK input is the primary clock input for the 603e, and represents the bus clock frequency for 603e bus operation. Internally, the 603e may be operating at an integer or half-integer multiple of the bus clock frequency.
Timing Comments	Duty cycle—Refer to the appropriate hardware specifications for timing comments. **Note**: SYSCLK is used as the frequency reference for the internal PLL clock generator, and must not be suspended or varied during normal operation to ensure proper PLL operation.

7.2.12.2 Test Clock (CLK_OUT)—Output

The test clock (CLK_OUT) signal is an output-only signal on the 603e. Following are the state meaning and timing comments for the CLK_OUT signal.

State Meaning	Asserted/Negated—Provides PLL clock output for PLL testing and monitoring. The CLK_OUT signal clocks at either the processor clock frequency, the bus clock frequency, or the half-bus clock frequency if enabled by the appropriate bits in the HID0 register; the default state of the CLK_OUT signal is high-impedance. The CLK_OUT signal is provided for testing purposes only.
Timing Comments	Assertion/Negation—Refer to the appropriate hardware specifications for timing comments.

7.2.12.3 PLL Configuration (PLL_CFG[0–3])—Input

The PLL (phase-lock loop) is configured by the PLL_CFG[0–3] signals. For a given SYSCLK (bus) frequency, the PLL configuration signals set the internal CPU frequency of operation.

Following are the state meaning and timing comments for the PLL_CFG[0–3] signals.

State Meaning	Asserted/Negated— Configures the operation of the PLL and the internal processor clock frequency. Settings are based on the desired bus and internal frequency of operation.
Timing Comments	Assertion/Negation—Must remain stable during operation; should only be changed during the assertion of $\overline{\text{HRESET}}$ or during sleep mode. These bits may be read through bits PC0–PC3 in the HID1 register.

Table 7-10. PLL Configuration

	Bus, CPU and PLL Frequencies							
PLL_CFG[0–3]	**CPU/ SYSCLK Ratio**	**Bus 16.6 MHz**	**Bus 20 MHz**	**Bus 25 MHz**	**Bus 33.3 MHz**	**Bus 40 MHz**	**Bus 50 MHz**	**Bus 66.6 MHz**
0000	1:1	—	—	—	—	—	—	66.6 (133)
0001	1:1	—	—	—	33.3 (133)	40 (160)	50 (200)	—
0010	1:1	16.6 (133)	20 (160)	25 (200)	—	—	—	—
1100	1.5:1	—	—	—	—	—	75 (150)	100 (200)
0100	2:1	—	—	—	66.6 (133)	80 (160)	100 (200)	—
0101	2:1	33.3 (133)	40 (160)	50 (200)	—	—	—	—
0110	2.5:1	—	—	—	83.3 (166)	100 (200)	—	—
1000	3:1	—	—	75 (150)	100 (200)	—	—	—
1110	3.5:1	—	70 (140)	87.5 (175)	—	—	—	—
1010	4:1	66.6 (133)	80 (160)	100 (200)	—	—	—	—
0011	PLL Bypass							
1111	Clock Off							

Notes:

1. Some PLL configurations may select bus, CPU, or PLL frequencies which are not useful, not supported, or not tested for by the 603e. For complete and up-to-date information, refer to the appropriate hardware specifications. PLL frequencies, shown in parentheses, should not fall below 133 MHz, and should not exceed 200 MHz.
2. In PLL-bypass mode, the SYSCLK input signal clocks the internal processor directly, and the bus is set for 1:1 mode operation. In clock-off mode, no clocking occurs inside the 603e regardless of the SYSCLK input.

7.2.13 Power and Ground Signals

The 603e provides the following connections for power and ground:

- VDD and OVDD—The VDD and OVDD signals provide the connection for the supply voltage. On the 603e, there is no electrical distinction between the VDD and the OVDD signals.
- AVDD—The AVDD power signal provides power to the clock generation phase-lock ed loop. See the appropriate hardware specifications for information on how to use this signal.
- GND and OGND—The GND and OGND signals provide the connection for grounding the 603e. On the 603e, there is no electrical distinction between the GND and OGND signals.

Chapter 8
System Interface Operation

This chapter describes the PowerPC 603e microprocessor's bus interface and its operation. It shows how the 603e signals, defined in Chapter 7, "Signal Descriptions," interact to perform address and data transfers.

8.1 Overview

The system interface prioritizes requests for bus operations from the instruction and data caches, and performs bus operations per the 603e bus protocol. It includes address register queues, prioritization logic, and bus control unit. The system interface latches snoop addresses for snooping in the data cache and in the address register queues, snoops for direct-store reply operations and for reservations controlled by the Load Word and Reserve Indexed (**lwarx**) and Store Word Conditional Indexed (**stwcx.**) instructions, and maintains the touch load address for the cache. The interface allows one level of pipelining; that is, with certain restrictions discussed later, there can be two outstanding transactions at any given time. Accesses are prioritized with load operations preceding store operations.

Instructions are automatically fetched from the memory system into the instruction unit where they are dispatched to the execution units at a peak rate of three instructions per clock. Conversely, load and store instructions explicitly specify the movement of operands to and from the integer and floating-point register files and the memory system. (The EC603e microprocessor does not support the floating-point register files.)

When the 603e encounters an instruction or data access, it calculates the logical address (effective address in the architecture specification) and uses the low-order address bits to check for a hit in the on-chip, 16-Kbyte instruction and data caches. During cache lookup, the instruction and data memory management units (MMUs) use the higher-order address bits to calculate the virtual address, from which they calculate the physical address (real address in the architecture specification). The physical address bits are then compared with the corresponding cache tag bits to determine if a cache hit occurred. If the access misses in the corresponding cache, the physical address is used to access system memory.

In addition to the loads, stores, and instruction fetches, the 603e performs software table search operations following TLB misses, cache cast-out operations when least-recently used cache lines are written to memory after a cache miss, and cache-line snoop push-out operations when a modified cache line experiences a snoop hit from another bus master.

Figure 8-1 shows the address path from the execution units and instruction fetcher, through the translation logic to the caches and system interface logic.

The 603e uses separate address and data buses and a variety of control and status signals for performing reads and writes. The address bus is 32 bits wide and the data bus can be configured to be 32 or 64 bits wide. The interface is synchronous—all 603e inputs are sampled at and all outputs are driven from the rising edge of the bus clock. The bus can run at the full processor-clock frequency or at an integer division of the processor-clock speed. While the 603e operates at 3.3 volts, all the I/O signals are 5.0 volt TTL-compatible.

8.1.1 Operation of the Instruction and Data Caches

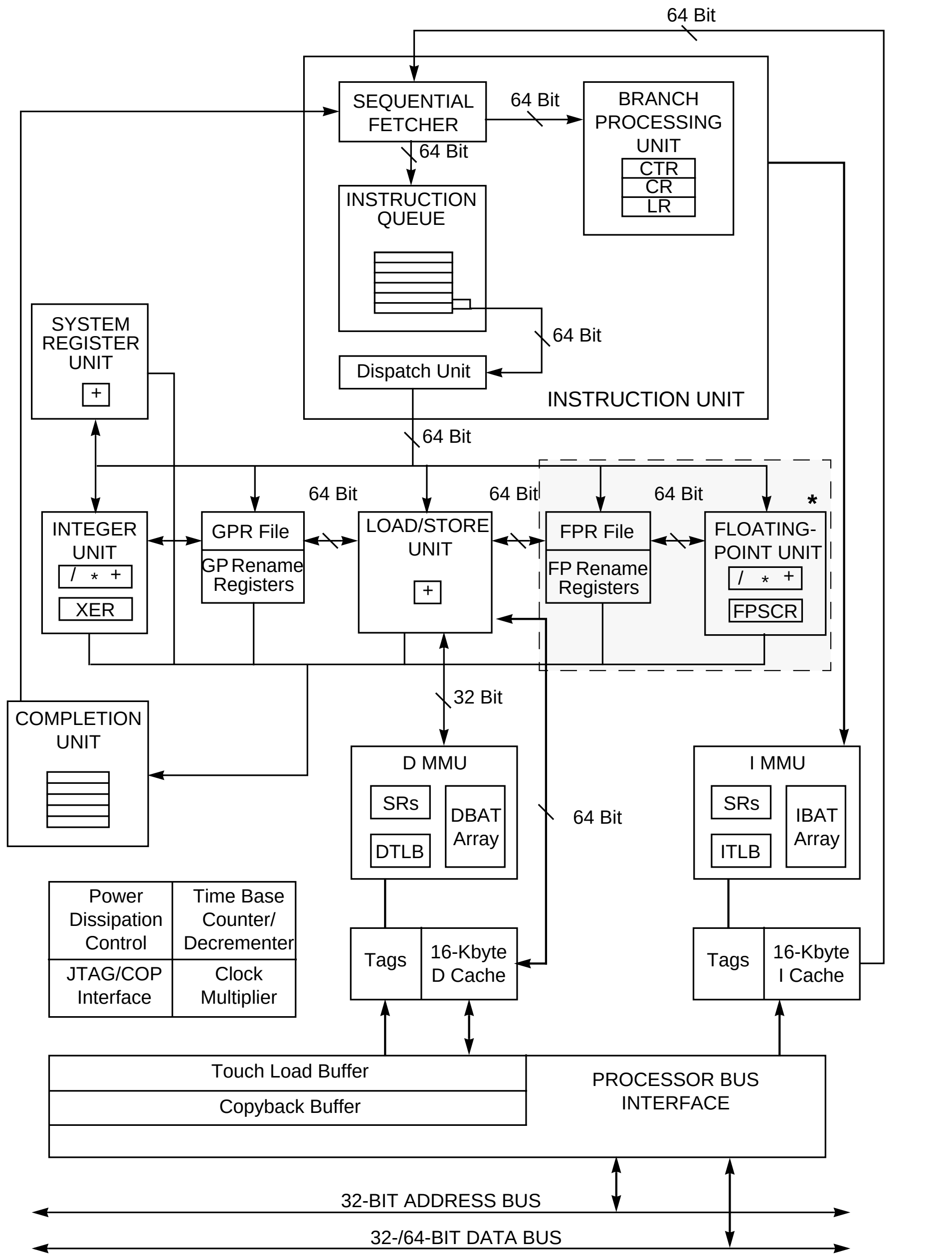
The 603e provides independent instruction and data caches. Each cache is a physically-addressed, 16-Kbyte cache with four-way set associativity. Both caches consist of 128 sets of four cache lines, with eight words in each cache line.

Because the data cache on the 603e is an on-chip, write-back primary cache, the predominant type of transaction for most applications is burst-read memory operations, followed by burst-write memory operations, direct-store operations, and single-beat (noncacheable or write-through) memory read and write operations. Additionally, there can be address-only operations, variants of the burst and single-beat operations (global memory operations that are snooped, and atomic memory operations, for example), and address retry activity (for example, when a snooped read access hits a modified line in the cache).

Since the 603e data cache tags are single ported, simultaneous load or store and snoop accesses cause resource contention. Snoop accesses have the highest priority and are given first access to the tags, unless the snoop access coincides with a tag write, in which case the snoop is retried and must re-arbitrate for access to the cache. Loads or stores that are deferred due to snoop accesses are performed on the clock cycle following the snoop.

The 603e supports a three-state coherency protocol that supports the modified, exclusive, and invalid (MEI) cache states. The protocol is a subset of the MESI (modified/exclusive/shared/invalid) four-state protocol and operates coherently in systems that contain four-state caches. With the exception of the **dcbz** instruction, the 603e does not broadcast cache control instructions. The cache control instructions are intended for the management of the local cache but not for other caches in the system.

Cache lines in the 603e are loaded in four beats of 64 bits each (or eight beats of 32 bits each when operating in 32-bit bus mode). The burst load is performed as “critical double word first.” The cache that is being loaded is blocked to internal accesses until the load completes (that is, no hits under misses). The critical double word is simultaneously written to the cache and forwarded to the requesting unit, thus minimizing stalls due to load delays.



* Note that the EC603e microprocessor does not support the floating-point unit or the floating-point register file.

Figure 8-1. Block Diagram

Cache lines are selected for replacement based on an LRU (least recently used) algorithm. Each time a cache line is accessed, it is tagged as the most recently used line of the set. When a miss occurs, if both lines in the set are marked as valid, the least recently used line is replaced with the new data. When data to be replaced is in the modified state, the modified data is written into a write-back buffer while the missed data is being read from memory. When the load completes, the 603e then pushes the replaced line from the write-back buffer to main memory in a burst write operation.

8.1.2 Operation of the System Interface

Memory accesses can occur in single-beat (1–8 bytes) and four-beat (32 bytes) burst data transfers when the 603e is configured with a 64-bit data bus. When the 603e is in the optional 32-bit data bus mode, memory accesses can occur in single-beat (1 to 4 bytes), two-beat (8 bytes), and eight-beat (32 bytes) bursts. The address and data buses are independent for memory accesses to support pipelining and split transactions. The 603e can pipeline as many as two transactions and has limited support for out-of-order split-bus transactions.

Access to the system interface is granted through an external arbitration mechanism that allows devices to compete for bus mastership. This arbitration mechanism is flexible, allowing the 603e to be integrated into systems that implement various fairness and bus-parking procedures to avoid arbitration overhead.

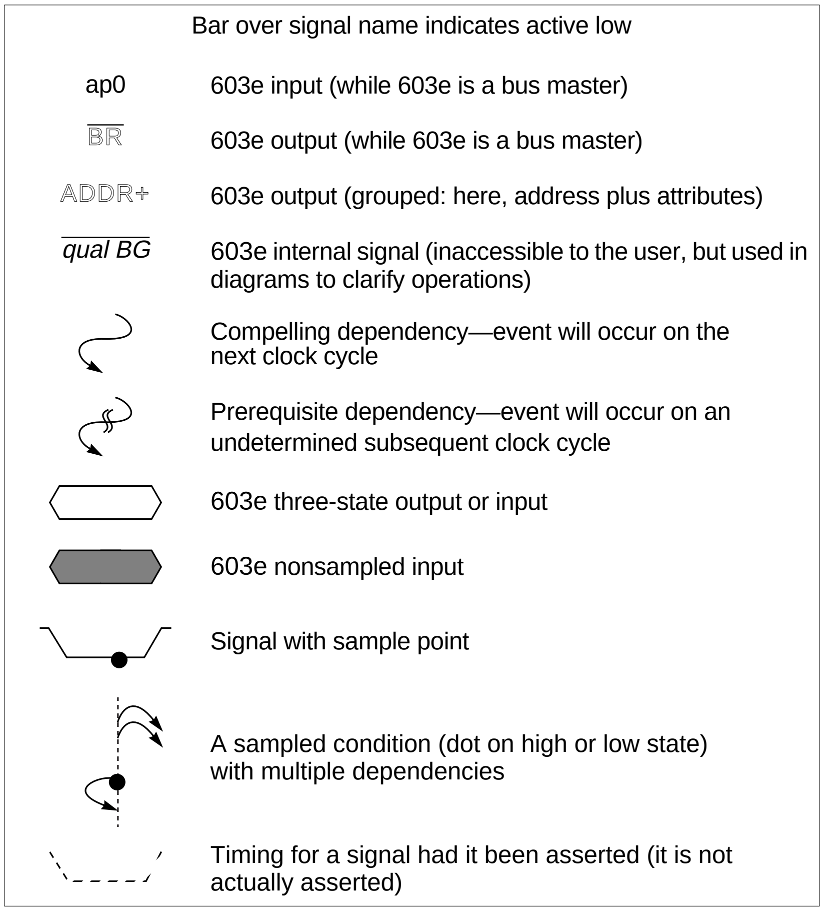
Typically, memory accesses are weakly ordered—sequences of operations, including load/store string and multiple instructions, do not necessarily complete in the order they begin—maximizing the efficiency of the bus without sacrificing coherency of the data. The 603e allows load operations to precede store operations (except when a dependency exists). In addition, the 603e can be configured to reorder high-priority store operations ahead of lower-priority store operations. Because the processor can dynamically optimize run-time ordering of load/store traffic, overall performance is improved.

Note that the Synchronize (**sync**) instruction can be used to enforce strong ordering.

The following sections describe how the 603e interface operates, providing detailed timing diagrams that illustrate how the signals interact. A collection of more general timing diagrams are included as examples of typical bus operations.

Figure 8-2 is a legend of the conventions used in the timing diagrams.

This is a synchronous interface—all 603e input signals are sampled and output signals are driven on the rising edge of the bus clock cycle (see the *PowerPC 603e RISC Microprocessor Hardware Specifications* for exact timing information).


Bar over signal name indicates active low

ap0	603e input (while 603e is a bus master)
$\overline{\text{BR}}$	603e output (while 603e is a bus master)
ADDR+	603e output (grouped: here, address plus attributes)
$\overline{\textit{qual BG}}$	603e internal signal (inaccessible to the user, but used in diagrams to clarify operations)
	Compelling dependency—event will occur on the next clock cycle
	Prerequisite dependency—event will occur on an undetermined subsequent clock cycle
	603e three-state output or input
	603e nonsampled input
	Signal with sample point
	A sampled condition (dot on high or low state) with multiple dependencies
	Timing for a signal had it been asserted (it is not actually asserted)

Figure 8-2. Timing Diagram Legend

8.1.2.1 Optional 32-Bit Data Bus Mode

The 603e supports an optional 32-bit data bus mode. The 32-bit data bus mode operates the same as the 64-bit data bus mode with the exception of the byte lanes involved in the transfer and the number of data beats that are performed. The number of data beats required for a data tenure in the 32-bit data bus mode is one, two, or eight beats depending on the size of the program transaction and the cache mode for the address. For additional information about 32-bit data bus mode, see Section 8.6.1, “32-Bit Data Bus Mode.”

8.1.3 Direct-Store Accesses

The 603e does not support the extended transfer protocol for accesses to the direct-store storage space. The transfer protocol used for any given access is selected by the T bit in the MMU segment registers; if the T bit is set, the memory access is a direct-store access. An attempt to access to a direct-store segment will result in the 603e taking a DSI exception.

8.2 Memory Access Protocol

Memory accesses are divided into address and data tenures. Each tenure has three phases—bus arbitration, transfer, and termination. The 603e also supports address-only transactions. Note that address and data tenures can overlap, as shown in Figure 8-3.

Figure 8-3 shows that the address and data tenures are distinct from one another and that both consist of three phases—arbitration, transfer, and termination. Address and data tenures are independent (indicated in Figure 8-3 by the fact that the data tenure begins before the address tenure ends), which allows split-bus transactions to be implemented at the system level in multiprocessor systems. Figure 8-3 shows a data transfer that consists of a single-beat transfer of as many as 64 bits. Four-beat burst transfers of 32-byte cache lines require data transfer termination signals for each beat of data.

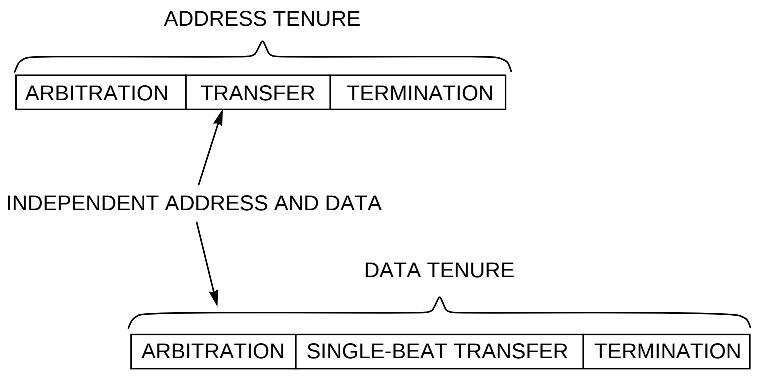


Figure 8-3. Overlapping Tenures on the Bus for a Single-Beat Transfer

The basic functions of the address and data tenures are as follows:

- Address tenure
  - Arbitration: During arbitration, address bus arbitration signals are used to gain mastership of the address bus.
  - Transfer: After the 603e is the address bus master, it transfers the address on the address bus. The address signals and the transfer attribute signals control the address transfer. The address parity and address parity error signals ensure the integrity of the address transfer.
  - Termination: After the address transfer, the system signals that the address tenure is complete or that it must be repeated.
- Data tenure
  - Arbitration: To begin the data tenure, the 603e arbitrates for mastership of the data bus.
  - Transfer: After the 603e is the data bus master, it samples the data bus for read operations or drives the data bus for write operations. The data parity and data parity error signals ensure the integrity of the data transfer.
  - Termination: Data termination signals are required after each data beat in a data transfer. Note that in a single-beat transaction, the data termination signals also indicate the end of the tenure, 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.

The 603e generates an address-only bus transfer during the execution of the **dcbz** instruction, which uses only the address bus with no data transfer involved. Additionally, the 603e's retry capability provides an efficient snooping protocol for systems with multiple memory systems (including caches) that must remain coherent.

8.2.1 Arbitration Signals

Arbitration for both address and data bus mastership is performed by a central, external arbiter and, minimally, by the arbitration signals shown in Section 7.2.1, "Address Bus Arbitration Signals." 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. These signals are inputs unless the 603e has mastership of one or both of the respective buses; they must be connected high through pull-up resistors so that they remain negated when no devices have control of the buses.

The following list describes the address arbitration signals:

- **$\overline{\text{BR}}$ (bus request)**—Assertion indicates that the 603e is requesting mastership of the address bus.
- **$\overline{\text{BG}}$ (bus grant)**—Assertion indicates that the 603e may, with the proper qualification, assume mastership of the address bus. A qualified bus grant occurs when $\overline{\text{BG}}$ is asserted and $\overline{\text{ABB}}$ and $\overline{\text{ARTRY}}$ are negated.

  If the 603e is parked, $\overline{\text{BR}}$ need not be asserted for the qualified bus grant.
- **$\overline{\text{ABB}}$ (address bus busy)**—Assertion by the 603e indicates that the 603e is the address bus master.

The following list describes the data arbitration signals:

- **$\overline{\text{DBG}}$ (data bus grant**)**—**Indicates that the 603e may, with the proper qualification, assume mastership of the data bus. A qualified data bus grant occurs when $\overline{\text{DBG}}$ is asserted while $\overline{\text{DBB}}$, $\overline{\text{DRTRY}}$, and $\overline{\text{ARTRY}}$ are negated.

  The $\overline{\text{DBB}}$ signal is driven by the current bus master, $\overline{\text{DRTRY}}$ is only driven from the bus, and $\overline{\text{ARTRY}}$ is from the bus, but only for the address bus tenure associated with the current data bus tenure (that is, not from another address tenure).
- **$\overline{\text{DBWO}}$ (data bus write only**)—Assertion indicates that the 603e may perform the data bus tenure for an outstanding write address even if a read address is pipelined before the write address. If $\overline{\text{DBWO}}$ is asserted, the 603e will assume data bus mastership for a pending data bus write operation; the 603e will take the data bus for a pending read operation if this input is asserted along with $\overline{\text{DBG}}$ and no write is pending. Care must be taken with $\overline{\text{DBWO}}$ to ensure the desired write is queued (for example, a cache-line snoop push-out operation).
- **$\overline{\text{DBB}}$ (data bus busy)**—Assertion by the 603e indicates that the 603e is the data bus master. The 603e always assumes data bus mastership if it needs the data bus and is given a qualified data bus grant (see $\overline{\text{DBG}}$).

  For more detailed information on the arbitration signals, refer to Section 7.2.1, “Address Bus Arbitration Signals,” and Section 7.2.6, “Data Bus Arbitration Signals.”

8.2.2 Address Pipelining and Split-Bus Transactions

The 603e protocol provides independent address and data bus capability to support pipelined and split-bus transaction system organizations. Address pipelining allows the address tenure of a new bus transaction to begin before the data tenure of the current transaction has finished. Split-bus transaction capability allows other bus activity to occur (either from the same master or from different masters) between the address and data tenures of a transaction.

While this capability does not inherently reduce memory latency, support for address pipelining and split-bus transactions can greatly improve effective bus/memory throughput. For this reason, these techniques are most effective in shared-memory multiprocessor implementations where bus bandwidth is an important measurement of system performance.

External arbitration is required in systems in which multiple devices must compete for the system bus. 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 mastership of the address bus to the next requesting master before the current data bus tenure has completed. Two address tenures can occur before the current data bus tenure completes.

The 603e can pipeline its own transactions to a depth of one level (intraprocessor pipelining); however, the 603e bus protocol does not constrain the maximum number of levels of pipelining that can occur on the bus between multiple masters (interprocessor pipelining). The external arbiter must control the pipeline depth and synchronization between masters and slaves.

In a pipelined implementation, data bus tenures are kept in strict order with respect to address tenures. However, external hardware can further decouple the address and data buses, allowing the data tenures to occur out of order with respect to the address tenures. This requires some form of system tag to associate the out-of-order data transaction with the proper originating address transaction (not defined for the 603e interface). Individual bus requests and data bus grants from each processor can be used by the system to implement tags to support interprocessor, out-of-order transactions.

The 603e supports a limited intraprocessor out-of-order, split-transaction capability via the data bus write only ($\overline{\text{DBWO}}$) signal. For more information about using $\overline{\text{DBWO}}$, see Section 8.10, "Using Data Bus Write Only."

8.3 Address Bus Tenure

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

8.3.1 Address Bus Arbitration

When the 603e needs access to the external bus and it is not parked ($\overline{\text{BG}}$ is negated), it asserts bus request ($\overline{\text{BR}}$) until it is granted mastership of the bus and the bus is available (see Figure 8-4). The external arbiter must grant master-elect status to the potential master by asserting the bus grant ($\overline{\text{BG}}$) signal. The 603e requesting the bus determines that the bus is available when the $\overline{\text{ABB}}$ input is negated. When the address bus is not busy ($\overline{\text{ABB}}$ input is negated), $\overline{\text{BG}}$ is asserted and the address retry ($\overline{\text{ARTRY}}$) input is negated. This is referred to as a qualified bus grant. The potential master assumes address bus mastership by asserting $\overline{\text{ABB}}$ when it receives a qualified bus grant.

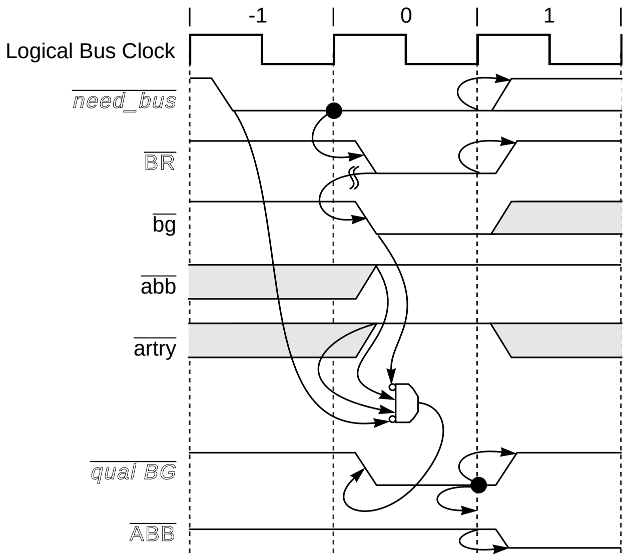


Figure 8-4. Address Bus Arbitration

External arbiters must allow only one device at a time to be the address bus master. In implementations in which no other device can be a master, $\overline{\text{BG}}$ can be grounded (always asserted) to continually grant mastership of the address bus to the 603e.

If the 603e asserts $\overline{\text{BR}}$ before the external arbiter asserts $\overline{\text{BG}}$, the 603e is considered to be unparked, as shown in Figure 8-4. Figure 8-5 shows the parked case, where a qualified bus grant exists on the clock edge following a need_bus condition. Notice that the bus clock cycle required for arbitration is eliminated if the 603e is parked, reducing overall memory latency for a transaction. The 603e always negates $\overline{\text{ABB}}$ for at least one bus clock cycle after $\overline{\text{AACK}}$ is asserted, even if it is parked and has another transaction pending.

Typically, bus parking is provided to the device that was the most recent bus master; however, system designers may choose other schemes such as providing unrequested bus grants in situations where it is easy to correctly predict the next device requesting bus mastership.

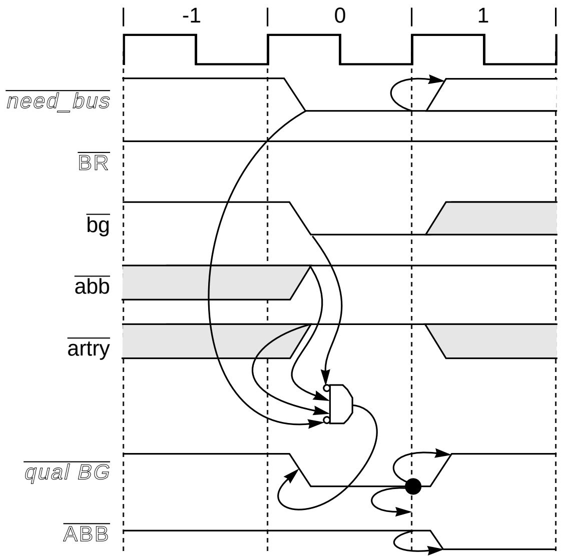


Figure 8-5. Address Bus Arbitration Showing Bus Parking

When the 603e 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 603e drives the address for the requested access onto the address bus and asserts $\overline{\text{TS}}$ to indicate the start of a new transaction.

When designing external bus arbitration logic, note that the 603e may assert $\overline{\text{BR}}$ without using the bus after it receives the qualified bus grant. For example, in a system using bus snooping, if the 603e asserts $\overline{\text{BR}}$ to perform a replacement copy-back operation, another device can invalidate that line before the 603e is granted mastership of the bus. Once the 603e is granted the bus, it no longer needs to perform the copy-back operation; therefore, the 603e does not assert $\overline{\text{ABB}}$ and does not use the bus for the copy-back operation. Note that the 603e asserts $\overline{\text{BR}}$ for at least one clock cycle in these instances.

8.3.2 Address Transfer

During the address transfer, the physical address and all attributes of the transaction are transferred from the bus master to the slave device(s). Snooping logic may monitor the transfer to enforce cache coherency; see discussion about snooping in Section 8.3.3, “Address Transfer Termination.”

The signals used in the address transfer include the following signal groups:

- Address transfer start signal: Transfer start ($\overline{\text{TS}}$)
- Address transfer signals: Address bus (A[0–31]), address parity (AP[0–3]), and address parity error ($\overline{\text{APE}}$)
- Address transfer attribute signals: Transfer type (TT[0–4]), transfer code (TC[0–1]), transfer size (TSIZ[0–2]), transfer burst ($\overline{\text{TBST}}$), cache inhibit ($\overline{\text{CI}}$), write-through ($\overline{\text{WT}}$), global ($\overline{\text{GBL}}$), and cache set element (CSE[0–1])

Figure 8-6 shows that the timing for all of these signals, except $\overline{\text{TS}}$ and $\overline{\text{APE}}$, is identical. All of the address transfer and address transfer attribute signals are combined into the ADDR+ grouping in Figure 8-6. The $\overline{\text{TS}}$ signal indicates that the 603e has begun an address transfer and that the address and transfer attributes are valid (within the context of a synchronous bus). The 603e always asserts $\overline{\text{TS}}$ coincident with $\overline{\text{ABB}}$. As an input, $\overline{\text{TS}}$ need not coincide with the assertion of $\overline{\text{ABB}}$ on the bus (that is, $\overline{\text{TS}}$ can be asserted with, or on, a subsequent clock cycle after $\overline{\text{ABB}}$ is asserted; the 603e tracks this transaction correctly).

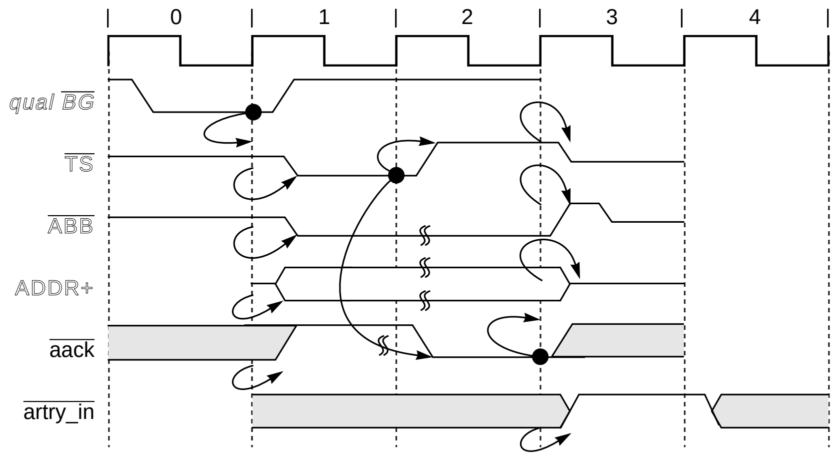


Figure 8-6. Address Bus Transfer

In Figure 8-6, the address transfer occurs during bus clock cycles 1 and 2 (arbitration occurs in bus clock cycle 0 and the address transfer is terminated in bus clock 3). In this diagram, the address bus termination input, $\overline{\text{AACK}}$, is asserted to the 603e on the bus clock following assertion of $\overline{\text{TS}}$ (as shown by the dependency line). This is the minimum duration of the address transfer for the 603e; the duration can be extended by delaying the assertion of $\overline{\text{AACK}}$ for one or more bus clocks.

8.3.2.1 Address Bus Parity

The 603e always generates 1 bit of correct odd-byte parity for each of the 4 bytes of address when a valid address is on the bus. The calculated values are placed on the AP[0–3] outputs when the 603e is the address bus master. If the 603e is not the master and $\overline{\text{TS}}$ and $\overline{\text{GBL}}$ are asserted together (qualified condition for snooping memory operations), the calculated values are compared with the AP[0–3] inputs. If there is an error, and address parity checking is enabled (HID0[EBA] set to 1), the $\overline{\text{APE}}$ output is asserted. An address bus parity error causes a checkstop condition if MSR[ME] is cleared to 0. For more information about checkstop conditions, see Chapter 4, “Exceptions.”

8.3.2.2 Address Transfer Attribute Signals

The transfer attribute signals include several encoded signals such as the transfer type (TT[0–4]) signals, transfer burst ($\overline{\text{TBST}}$) signal, transfer size (TSIZ[0–2]) signals, and transfer code (TC[0–1]) signals. Section 7.2.4, “Address Transfer Attribute Signals,” describes the encodings for the address transfer attribute signals.

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

Snooping logic should fully decode the transfer type signals if the $\overline{\text{GBL}}$ signal is asserted. Slave devices can sometimes use the individual transfer type signals without fully decoding the group. For a complete description of the encoding for transfer type signals TT[0–4], refer to Table 8-1 and Table 8-2.

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

The transfer size signals (TSIZ[0–2]) indicate the size of the requested data transfer as shown in Table 8-1. The TSIZ[0–2] signals may be used along with $\overline{\text{TBST}}$ and A[29–31] to determine which portion of the data bus contains valid data for a write transaction or which portion of the bus should contain valid data for a read transaction. Note that for a burst transaction (as indicated by the assertion of $\overline{\text{TBST}}$), TSIZ[0–2] are always set to 0b010. Therefore, if the $\overline{\text{TBST}}$ signal is asserted, the memory system should transfer a total of eight words (32 bytes), regardless of the TSIZ[0–2] encoding.

Table 8-1. Transfer Size Signal Encodings

$\overline{\text{TBST}}$	TSIZ0	TSIZ1	TSIZ2	Transfer Size
Asserted	0	1	0	Eight-word burst
Negated	0	0	0	Eight bytes
Negated	0	0	1	One byte
Negated	0	1	0	Two bytes
Negated	0	1	1	Three bytes
Negated	1	0	0	Four bytes
Negated	1	0	1	Five bytes (N/A)
Negated	1	1	0	Six bytes (N/A)
Negated	1	1	1	Seven bytes (N/A)

The basic coherency size of the bus is defined to be 32 bytes (corresponding to one cache line). Data transfers that cross an aligned, 32-byte boundary either must present a new address onto the bus at that boundary (for coherency consideration) or must operate as noncoherent data with respect to the 603e. The 603e never generates a bus transaction with a transfer size of 5 bytes, 6 bytes, or 7 bytes.

8.3.2.3 Burst Ordering During Data Transfers

During burst data transfer operations, 32 bytes of data (one cache line) are transferred to or from the cache in order. Burst write transfers are always performed zero double word first, but since burst reads are performed critical double word first, a burst read transfer may not start with the first double word of the cache line, and the cache line fill may wrap around the end of the cache line. This section describes the burst ordering for the 603e when operating in either the 64- or 32-bit bus mode.

Table 8-2 describes the burst ordering when the 603e is configured with a 64-bit data bus.

Table 8-2. Burst Ordering—64-Bit Bus

Data Transfer	For Starting Address:			
	A[27–28] = 00	A[27–28] = 01	A[27–28] = 10	A[27–28] = 11
First data beat	DW0	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 603e.

Table 8-3 describes the burst ordering when the 603e is configured with a 32-bit bus.

Table 8-3. Burst Ordering—32-Bit Bus

Data Transfer	For Starting Address:			
	A[27–28] = 00	A[27–28] = 01	A[27–28] = 10	A[27–28] = 11
First data beat	DW0-U	DW1-U	DW2-U	DW3-U
Second data beat	DW0-L	DW1-L	DW2-L	DW3-L
Third data beat	DW1-U	DW2-U	DW3-U	DW0-U
Fourth data beat	DW1-L	DW2-L	DW3-L	DW0-L
Fifth data beat	DW2-U	DW3-U	DW0-U	DW1-U
Sixth data beat	DW2-L	DW3-L	DW0-L	DW1-L
Seventh data beat	DW3-U	DW0-U	DW1-U	DW2-U
Eighth data beat	DW3-L	DW0-L	DW1-L	DW2-L

Notes: A[29–31] are always 0b000 for burst transfers by the 603e.

"U" and "L" represent the upper and lower word of the double word respectively.

8.3.2.4 Effect of Alignment in Data Transfers (64-Bit Bus)

Table 8-4 lists the aligned transfers that can occur on the 603e bus when configured with a 64-bit width. These are transfers in which the data is aligned to an address that is an integer multiple of the size of the data. For example, Table 8-4 shows that 1-byte data is always aligned; however, for a 4-byte word to be aligned, it must be oriented on an address that is a multiple of 4.

Table 8-4. Aligned Data Transfers (64-Bit Bus)

Transfer Size	TSIZ0	TSIZ1	TSIZ2	A[29–31]	Data Bus Byte Lane(s)							
					0	1	2	3	4	5	6	7
Byte	0	0	1	000	√	—	—	—	—	—	—	—
	0	0	1	001	—	√	—	—	—	—	—	—
	0	0	1	010	—	—	√	—	—	—	—	—
	0	0	1	011	—	—	—	√	—	—	—	—
	0	0	1	100	—	—	—	—	√	—	—	—
	0	0	1	101	—	—	—	—	—	√	—	—
	0	0	1	110	—	—	—	—	—	—	√	—
	0	0	1	111	—	—	—	—	—	—	—	√

Table 8-4. Aligned Data Transfers (64-Bit Bus) (Continued)

Transfer Size	TSIZ0	TSIZ1	TSIZ2	A[29–31]	Data Bus Byte Lane(s)							
					0	1	2	3	4	5	6	7
Half word	0	1	0	000	√	√	—	—	—	—	—	—
	0	1	0	010	—	—	√	√	—	—	—	—
	0	1	0	100	—	—	—	—	√	√	—	—
	0	1	0	110	—	—	—	—	—	—	√	√
Word	1	0	0	000	√	√	√	√	—	—	—	—
	1	0	0	100	—	—	—	—	√	√	√	√
Double word	0	0	0	000	√	√	√	√	√	√	√	√

Notes:

These entries indicate the byte portions of the requested operand that are read or written during that bus transaction.

These entries are not required and are ignored during read transactions and are driven with undefined data during all write transactions.

The 603e supports misaligned memory operations, although their use may substantially degrade performance. Misaligned memory transfers address memory that is not aligned to the size of the data being transferred (such as, a word read of an odd byte address). Although most of these operations hit in the primary cache (or generate burst memory operations if they miss), the 603e interface supports misaligned transfers within a word (32-bit aligned) boundary, as shown in Table 8-5. Note that the 4-byte transfer in Table 8-5 is only one example of misalignment. As long as the attempted transfer does not cross a word boundary, the 603e can transfer the data on the misaligned address (for example, a half-word read from an odd byte-aligned address). An attempt to address data that crosses a word boundary requires two bus transfers to access the data. Note that an attempt to load or store a floating-point operand that is not word-aligned will result in a floating-point alignment exception. For more information, refer to Section 4.5.6, “Alignment Exception (0x00600).”

Due to the performance degradations associated with misaligned memory operations, they are best avoided. In addition to the double-word straddle boundary condition, the address translation logic can generate substantial exception overhead when the load/store multiple and load/store string instructions access misaligned data. It is strongly recommended that software attempt to align code and data where possible.

Table 8-5. Misaligned Data Transfers (Four-Byte Examples)

Transfer Size (Four Bytes)	TSIZ[0–2]	A[29–31]	Data Bus Byte Lanes							
			0	1	2	3	4	5	6	7
Aligned	1 0 0	0 0 0	A	A	A	A	—	—	—	—
Misaligned—first access	0 1 1	0 0 1		A	A	A	—	—	—	—
second access	0 0 1	1 0 0	—	—	—	—	A	—	—	—
Misaligned—first access	0 1 0	0 1 0	—	—	A	A	—	—	—	—
second access	0 1 1	1 0 0	—	—	—	—	A	A	—	—
Misaligned—first access	0 0 1	0 1 1	—	—	—	A	—	—	—	—
second access	0 1 1	1 0 0	—	—	—	—	A	A	A	—
Aligned	1 0 0	1 0 0	—	—	—	—	A	A	A	A
Misaligned—first access	0 1 1	1 0 1	—	—	—	—	—	A	A	A
second access	0 0 1	0 0 0	A	—	—	—	—	—	—	—
Misaligned—first access	0 1 0	1 1 0	—	—	—	—	—	—	A	A
second access	0 1 0	0 0 0	A	A	—	—	—	—	—	—
Misaligned—first access	0 0 1	1 1 1	—	—	—	—	—	—	—	A
second access	0 1 1	0 0 0	A	A	A	—	—	—	—	—

Notes:

A: Byte lane used
—: Byte lane not used

8.3.2.5 Effect of Alignment in Data Transfers (32-Bit Bus)

The aligned data transfer cases for 32-bit data bus mode are shown in Table 8-6. All of the transfers require a single data beat (if caching-inhibited or write-through) except for double-word cases which require two data beats. The double-word case is only generated by the 603e for load or store double operations to/from the floating-point GPRs (not supported on the EC603e microprocessor). All caching-inhibited instruction fetches are performed as word operations.

Table 8-6. Aligned Data Transfers (32-Bit Bus Mode)

Transfer Size	TSIZ0	TSIZ1	TSIZ2	A[29–31]	Data Bus Byte Lane(s)							
					0	1	2	3	4	5	6	7
Byte	0	0	1	000	A	—	—	—	x	x	x	x
	0	0	1	001	—	A	x	—	x	x	x	x
	0	0	1	010	—	—	A	—	x	x	x	x
	0	0	1	011	—	—	—	A	x	x	x	x
	0	0	1	100	A	—	—	—	x	x	x	x
	0	0	1	101	—	A	—	—	x	x	x	x
	0	0	1	110	—	—	A	—	x	x	x	x
	0	0	1	111	—	—	—	A	x	x	x	x
Half word	0	1	0	000	A	A	—	—	x	x	x	x
	0	1	0	010	—	—	A	A	x	x	x	x
	0	1	0	100	A	A	—	—	x	x	x	x
	0	1	0	110	—	—	A	A	x	x	x	x
Word	1	0	0	000	A	A	A	A	x	x	x	x
	1	0	0	100	A	A	A	A	x	x	x	x
Double word	0	0	0	000	A	A	A	A	x	x	x	x
Second beat	0	0	0	000	A	A	A	A	x	x	x	x

Notes:

A: Byte lane used
—: Byte lane not used
x: Byte lane not used in 32-bit bus mode

Misaligned data transfers when the 603e is configured with a 32-bit data bus operate in the same way as when configured with a 64-bit data bus, with the exception that only the DH[0–31] data bus is used. See Table 8-7 for an example of a 4-byte misaligned transfer starting at each possible byte address within a double word.

Table 8-7. Misaligned 32-Bit Data Bus Transfer (Four-Byte Examples)

Transfer Size (Four Bytes)	TSIZ[0–2]	A[29–31]	Data Bus Byte Lanes							
			0	1	2	3	4	5	6	7
Aligned	1 0 0	0 0 0	A	A	A	A	x	x	x	x
Misaligned—first access	0 1 1	0 0 1		A	A	A	x	x	x	x
second access	0 0 1	1 0 0	A	—	—	—	x	x	x	x
Misaligned—first access	0 1 0	0 1 0	—	—	A	A	x	x	x	x
second access	0 1 0	1 0 0	A	A	—	x	x	x	x	x
Misaligned—first access	0 0 1	0 1 1	—	—	—	A	x	x	x	x
second access	0 1 1	1 0 0	A	A	A	—	x	x	x	x
Aligned	1 0 0	1 0 0	A	A	A	A	x	x	x	x
Misaligned—first access	0 1 1	1 0 1	—	A	A	A	x	x	x	x
second access	0 0 1	0 0 0	A	—	—	—	x	x	x	x
Misaligned—first access	0 1 0	1 1 0	—	—	A	A	x	x	x	x
second access	0 1 0	0 0 0	A	A	—	—	x	x	x	x
Misaligned—first access	0 0 1	1 1 1	—	—	—	A	x	x	x	x
second access	0 1 1	0 0 0	A	A	A	—	x	x	x	x

Notes:

A:	Byte lane used
—:	Byte lane not used
x:	Byte lane not used in 32-bit bus mode

8.3.2.5.1 Alignment of External Control Instructions

The size of the data transfer associated with the **eciwx** and **ecowx** instructions is always 4 bytes. However, if the **eciwx** or **ecowx** instruction is misaligned and crosses any word boundary, the 603e will generate two bus operations, each with a size of fewer than 4 bytes. For the first bus operation, bits A[29–31] equal bits 29–31 of the effective address of the instruction, which will be 0b101, 0b110, or 0b111. The size associated with the first bus operation will be 3, 2, or 1 bytes, respectively. For the second bus operation, bits A[29–31] equal 0b000, and the size associated with the operation will be 1, 2, or 3 bytes, respectively. For both operations, $\overline{\text{TBST}}$ and TSIZ[0–2] are redefined to specify the resource ID (RID). The resource ID is copied from bits 28–31 of the EAR. For **eciwx**/**ecowx** operations, the state of bit 28 of the EAR is presented by the $\overline{\text{TBST}}$ signal without inversion (if EAR[28] = 1, $\overline{\text{TBST}}$ = 1). The size of the second bus operation cannot be deduced from the operation itself; the system must determine how many bytes were transferred on the first bus operation to determine the size of the second operation.

Furthermore, the two bus operations associated with such a misaligned external control instruction are not atomic. That is, the 603e may initiate other types of memory operations between the two transfers. Also, the two bus operations associated with a misaligned **ecowx** may be interrupted by an **eciwx** bus operation, and vice versa. The 603e does guarantee that the two operations associated with a misaligned **ecowx** will not 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. (The term exception is referred to interrupt in the architecture specification.)

8.3.2.6 Transfer Code (TC[0–1]) Signals

The TC0 and TC1 signals provide supplemental information about the corresponding address. Note that the TC*x* signals can be used with the TT[0–4] and $\overline{\text{TBST}}$ signals to further define the current transaction.

Table 8-8 shows the encodings of the TC0 and TC1 signals.

Table 8-8. Transfer Code Encoding

TC[0–1]	Read	Write
0 0	Data transaction	Any write
0 1	Touch load	N/A
1 0	Instruction fetch	N/A
1 1	(Reserved)	N/A

8.3.3 Address Transfer Termination

The address tenure of a bus operation is terminated when completed with the assertion of $\overline{\text{AACK}}$, or retried with the assertion of $\overline{\text{ARTRY}}$. The 603e does not terminate the address transfer until the $\overline{\text{AACK}}$ (address acknowledge) input is asserted; therefore, the system can extend the address transfer phase by delaying the assertion of $\overline{\text{AACK}}$ to the 603e. Although $\overline{\text{AACK}}$ can be asserted as early as the bus clock cycle following $\overline{\text{TS}}$ (see Figure 8-7), which allows a minimum address tenure of two bus cycles when the 603e clock is configured for 1:1 (processor clock to bus clock) mode, the $\overline{\text{ARTRY}}$ snoop response cannot be determined in the minimum allowed address tenure period. When in 1:1 or 1.5:1 clock mode, $\overline{\text{AACK}}$ must not be asserted until the third clock of the address tenure (one address wait state) to allow the 603e an opportunity to assert $\overline{\text{ARTRY}}$ on the bus. For other clock configurations (2:1, 2.5:1, 3:1, 3.5:1, and 4:1), the $\overline{\text{ARTRY}}$ snoop response can be determined in the minimum address tenure period, and $\overline{\text{AACK}}$ may be asserted as early as the second bus clock of the address tenure. As shown in Figure 8-7, these signals are 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. 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 anytime during the address tenure and through the cycle following $\overline{\text{AACK}}$. The assertion causes the entire transaction (address and data tenure) to be rerun. As a snooping device, the 603e 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 a bus master, the 603e responds to an assertion of $\overline{\text{ARTRY}}$ by aborting the bus transaction and re-requesting the bus. Note that after recognizing an assertion of $\overline{\text{ARTRY}}$ and aborting the transaction in progress, the 603e is not guaranteed to run the same transaction the next time it is granted the bus due to internal reordering of load and store operations.

If an address retry is required, the $\overline{\text{ARTRY}}$ response will be asserted by a bus snooping device as early as the second cycle after the assertion of $\overline{\text{TS}}$ (or until the third cycle following $\overline{\text{TS}}$ if 1:1 or 1.5:1 processor to bus clock ratio is selected). 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 referred to as a qualified $\overline{\text{ARTRY}}$. An earlier assertion of $\overline{\text{ARTRY}}$ during the address tenure is referred to as an early $\overline{\text{ARTRY}}$.

As a bus master, the 603e recognizes either an early or qualified $\overline{\text{ARTRY}}$ and prevents the data tenure associated with the retried address tenure. If the data tenure has already begun, the 603e aborts and terminates the data tenure immediately even if the burst 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 603e ignores the first data beat, and if it is a load operation, does not forward data internally to the cache and execution units. If $\overline{\text{ARTRY}}$ is asserted after the first (or only) assertion of $\overline{\text{TA}}$, improper operation of the bus interface may result.

During the clock of a qualified $\overline{\text{ARTRY}}$, the 603e also determines if it should negate $\overline{\text{BR}}$ and ignore $\overline{\text{BG}}$ on the following cycle. On the following cycle, only the snooping master that asserted $\overline{\text{ARTRY}}$ and needs to perform a snoop copy-back operation is allowed to assert $\overline{\text{BR}}$. This guarantees the snooping master an opportunity to request and be granted the bus before the just-retried master can restart its transaction. Note that a nonclocked bus arbiter may detect the assertion of address bus request by the bus master that asserted $\overline{\text{ARTRY}}$, and return a qualified bus grant one cycle earlier than shown in Figure 8-7.

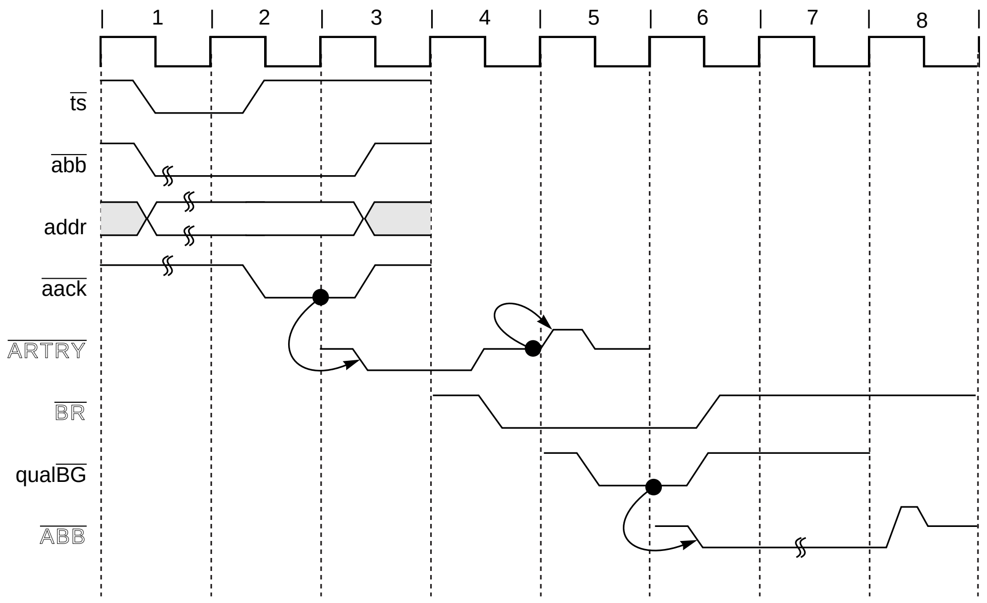


Figure 8-7. Snooped Address Cycle with $\overline{\text{ARTRY}}$

8.4 Data Bus Tenure

This section describes the data bus arbitration, transfer, and termination phases defined by the 603e memory access protocol. The phases of the data tenure are identical to those of the address tenure, underscoring the symmetry in the control of the two buses.

8.4.1 Data Bus Arbitration

Data bus arbitration uses the data arbitration signal group—$\overline{\text{DBG}}$, $\overline{\text{DBWO}}$, and $\overline{\text{DBB}}$. Additionally, the combination of $\overline{\text{TS}}$ and TT[0–4] provides information about the data bus request to external logic.

The $\overline{\text{TS}}$ signal is an implied data bus request from the 603e; the arbiter must qualify $\overline{\text{TS}}$ with the transfer type (TT) encodings to determine if the current address transfer is an address-only operation, which does not require a data bus transfer (see Figure 8-7). If the data bus is needed, the arbiter grants data bus mastership by asserting the $\overline{\text{DBG}}$ input to the 603e. As with the address bus arbitration phase, the 603e must qualify the $\overline{\text{DBG}}$ input with a number of input signals before assuming bus mastership, as shown in Figure 8-8.

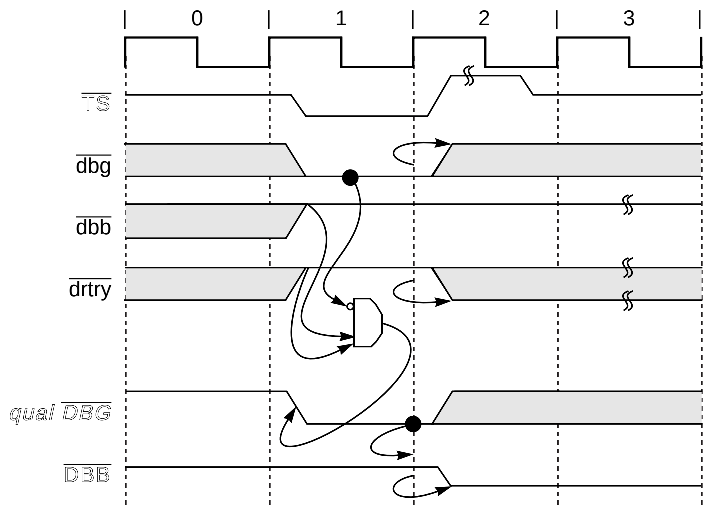


Figure 8-8. Data Bus Arbitration

A qualified data bus grant can be expressed as the following:

> QDBG = $\overline{\text{DBG}}$ asserted while $\overline{\text{DBB}}$, $\overline{\text{DRTRY}}$, and $\overline{\text{ARTRY}}$ (associated with the data bus operation) are negated.

When a data tenure overlaps with its associated address tenure, a qualified $\overline{\text{ARTRY}}$ assertion coincident with a data bus grant signal does not result in data bus mastership ($\overline{\text{DBB}}$ is not asserted). Otherwise, the 603e always asserts $\overline{\text{DBB}}$ on the bus clock cycle after recognition of a qualified data bus grant. Since the 603e can pipeline transactions, there may be an outstanding data bus transaction when a new address transaction is retried. In this case, the 603e becomes the data bus master to complete the previous transaction.

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

The $\overline{\text{DBB}}$ signal should be connected between masters if data tenure scheduling is left to the masters. Optionally, the memory system can control data tenure scheduling directly with $\overline{\text{DBG}}$. However, it is possible to ignore the $\overline{\text{DBB}}$ signal in the system if the $\overline{\text{DBB}}$ input is not used as the final data bus allocation control between data bus masters, and if the memory system can track the start and end of the data tenure. If $\overline{\text{DBB}}$ is not used to signal the end of a data tenure, $\overline{\text{DBG}}$ is only asserted to the next bus master the cycle before the cycle that the next bus master may actually begin its data tenure, rather than asserting it earlier (usually during another master's data tenure) and allowing the negation of $\overline{\text{DBB}}$ to be the final gating signal for a qualified data bus grant. Even if $\overline{\text{DBB}}$ is ignored in the system, the 603e always recognizes its own assertion of $\overline{\text{DBB}}$, and requires one cycle after data tenure completion to negate its own $\overline{\text{DBB}}$ before recognizing a qualified data bus grant for another data tenure. If $\overline{\text{DBB}}$ is ignored in the system, it must still be connected to a pull-up resistor on the 603e to ensure proper operation.

8.4.2 Data Bus Write Only

As a result of address pipelining, the 603e may have up to two data tenures queued to perform when it receives a qualified $\overline{\text{DBG}}$. Generally, the data tenures should be performed in strict order (the same order) as their address tenures were performed. The 603e, however, also supports a limited out-of-order capability with the data bus write only ($\overline{\text{DBWO}}$) input. When recognized on the clock of a qualified $\overline{\text{DBG}}$, $\overline{\text{DBWO}}$ may direct the 603e to perform the next pending data write tenure even if a pending read tenure would have normally been performed first. For more information on the operation of $\overline{\text{DBWO}}$, refer to Section 8.10, “Using Data Bus Write Only.”

If the 603e has any data tenures to perform, it always accepts data bus mastership to perform a 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 to run, the 603e still takes mastership of the data bus to perform the next pending read data tenure.

Generally, $\overline{\text{DBWO}}$ should only be used 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.

8.4.3 Data Transfer

The data transfer signals include DH[0–31], DL[0–31], DP[0–7] and $\overline{\text{DPE}}$. For memory accesses, the DH and DL signals form a 64-bit data path for read and write operations.

The 603e transfers data in either single- or four-beat burst transfers when configured with a 64-bit data bus; when configured with a 32-bit data bus, the 603e performs one-, two-, and eight-beat data transfers. Single-beat operations can transfer from 1 to 8 bytes at a time and can be misaligned; see Section 8.3.2.4, “Effect of Alignment in Data Transfers (64-Bit Bus).” Note that the EC603e microprocessor can transfer from 1 to 4 bytes during single-beat operations. Burst operations always transfer eight words and are aligned on eight-word address boundaries. Burst transfers can achieve significantly higher bus throughput than single-beat operations.

The type of transaction initiated by the 603e depends on whether the code or data is cacheable and, for store operations whether the cache is considered in write-back or write-through mode, which software controls on either a page or block basis. Burst transfers support cacheable operations only; that is, memory structures must be marked as cacheable (and write-back for data store operations) in the respective page or block descriptor to take advantage of burst transfers.

The 603e output $\overline{\text{TBST}}$ indicates to the system whether the current transaction is a single- or four-beat 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 in the cache (and are marked as cacheable and, for stores, write-back in the MMU), the 603e uses the double-word-aligned address associated with the critical code or data that initiated the transaction. This minimizes latency by allowing the critical code or data to be forwarded to the processor before the rest of the cache line is filled. For all other burst operations, however, the cache line is transferred beginning with the oct-word-aligned data.

The 603e does not directly support dynamic interfacing to subsystems with less than a 64-bit data path. It does, however, provide a static 32-bit data bus mode; for more information, see Section 8.1.2.1, “Optional 32-Bit Data Bus Mode.”

8.4.4 Data Transfer Termination

Four signals are used to terminate data bus transactions—$\overline{\text{TA}}$, $\overline{\text{DRTRY}}$ (data retry), $\overline{\text{TEA}}$ (transfer error acknowledge), and $\overline{\text{ARTRY}}$. The $\overline{\text{TA}}$ signal indicates normal termination of data transactions. It must always be asserted on the bus cycle coincident with the data that it is qualifying. It may be withheld by the slave for any number of clocks 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 603e 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. $\overline{\text{TEA}}$ indicates a nonrecoverable bus error event. Upon receiving a final (or only) termination condition, the 603e always negates $\overline{\text{DBB}}$ for one cycle.

If $\overline{\text{DRTRY}}$ is asserted by the memory system 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 will negate $\overline{\text{DRTRY}}$ on that clock. This is to prevent a potential momentary data bus conflict if a write access begins on the following cycle.

The $\overline{\text{TEA}}$ signal is used to signal a nonrecoverable error during the data transaction. It may be asserted on any cycle during $\overline{\text{DBB}}$, or on the cycle after a qualified $\overline{\text{TA}}$ during a read operation, except when no-$\overline{\text{DRTRY}}$ mode is selected (where no-$\overline{\text{DRTRY}}$ mode cancels checking the cycle after $\overline{\text{TA}}$). The assertion of $\overline{\text{TEA}}$ terminates the data tenure immediately even if in the middle of a burst; however, it does not prevent incorrect data that has just been acknowledged with $\overline{\text{TA}}$ from being written into the 603e’s cache or GPRs. The assertion of $\overline{\text{TEA}}$ initiates either a machine check exception or a checkstop condition based on the setting of the MSR.

An assertion of $\overline{\text{ARTRY}}$ causes the data tenure to be terminated immediately if the $\overline{\text{ARTRY}}$ is for the address tenure associated with the data tenure in operation. If $\overline{\text{ARTRY}}$ is connected for the 603e, the earliest allowable assertion of $\overline{\text{TA}}$ to the 603e is directly dependent on the earliest possible assertion of $\overline{\text{ARTRY}}$ to the 603e; see Section 8.3.3, “Address Transfer Termination.”

If the 603e clock is configured for 1:1 or 1.5:1 (processor clock to bus clock ratio) mode and the 603e is performing a burst read into its data cache, at least one wait state must be provided 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, at least two wait states must be provided. The wait states are required due to possible resource contention in the data cache caused by a block replacement (or cast-out) required in connection with the new linefill. These waits states may be provided by withholding the assertion of $\overline{\text{TA}}$ to the 603e for that data tenure, or by withholding $\overline{\text{DBG}}$ to the 603e thereby delaying the start of the data tenure. This restriction applies only to burst reads into the data cache when configured in 1:1 or 1.5:1 clock modes. (It does not apply to instruction fetches, write operations, noncachable read operations, or non-1:1 or 1.5:1 clock modes.)

8.4.4.1 Normal Single-Beat Termination

Normal termination of a single-beat data read operation occurs when $\overline{\text{TA}}$ is asserted by a responding slave. The $\overline{\text{TEA}}$ and $\overline{\text{DRTRY}}$ signals must remain negated during the transfer (see Figure 8-9).

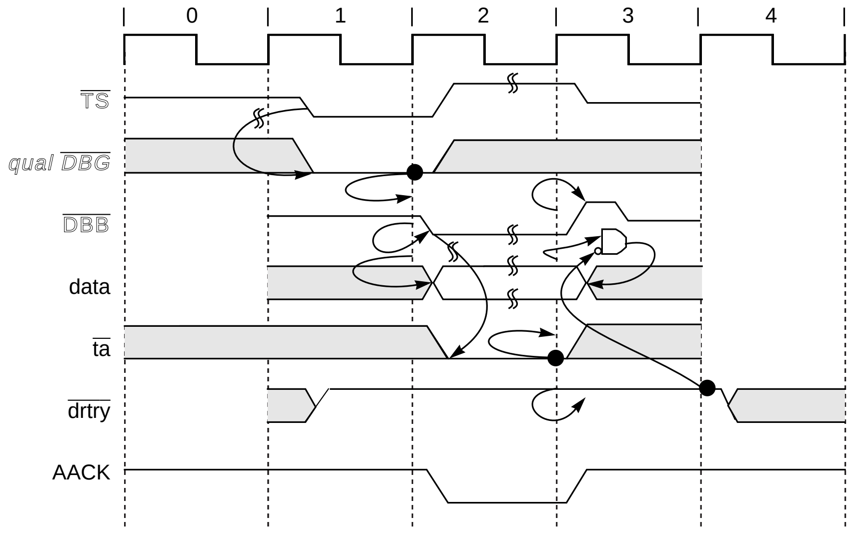


Figure 8-9. Normal Single-Beat Read Termination

The $\overline{\text{DRTRY}}$ signal is not sampled during data writes, as shown in Figure 8-10.

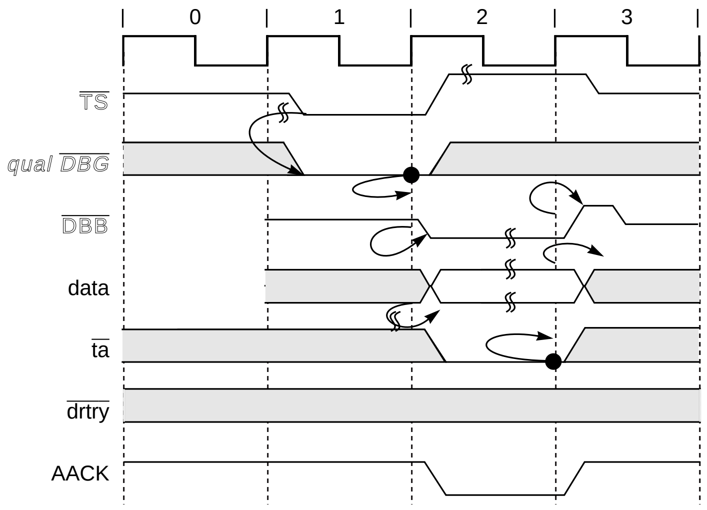


Figure 8-10. Normal Single-Beat Write Termination

Normal termination of a burst transfer occurs when $\overline{\text{TA}}$ is asserted for four bus clock cycles, as shown in Figure 8-11. The bus clock cycles in which $\overline{\text{TA}}$ is asserted need not be consecutive, thus allowing pacing of the data transfer beats. For read bursts to terminate successfully, $\overline{\text{TEA}}$ and $\overline{\text{DRTRY}}$ must remain negated during the transfer. For write bursts, $\overline{\text{TEA}}$ must remain negated for a successful transfer. $\overline{\text{DRTRY}}$ is ignored during data writes.

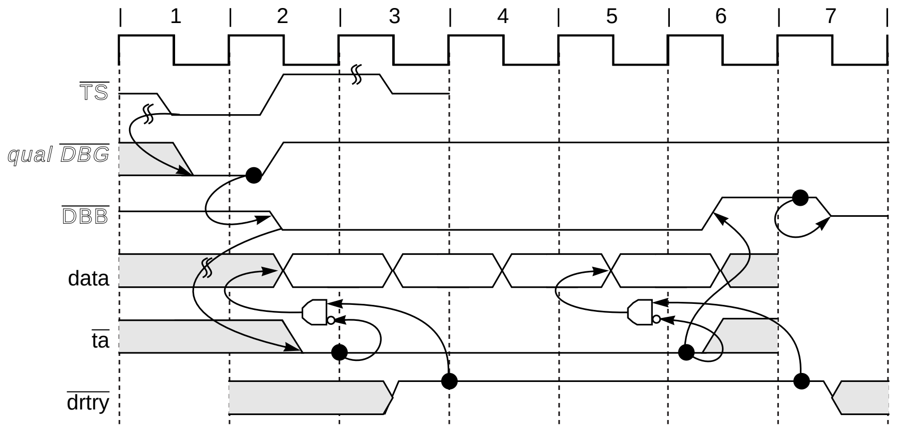


Figure 8-11. Normal Burst Transaction

For read bursts, $\overline{\text{DRTRY}}$ may be asserted one bus clock cycle after $\overline{\text{TA}}$ is asserted to signal that the data presented with $\overline{\text{TA}}$ is invalid and that the processor must wait for the negation of $\overline{\text{DRTRY}}$ before forwarding data to the processor (see Figure 8-12). Thus, a data beat can be terminated by a predicted branch with $\overline{\text{TA}}$ and then one bus clock cycle later confirmed with the negation of $\overline{\text{DRTRY}}$. The $\overline{\text{DRTRY}}$ signal is valid only for read transactions. $\overline{\text{TA}}$ must be asserted on the bus clock cycle before the first bus clock cycle of the assertion of $\overline{\text{DRTRY}}$; otherwise the results are undefined.

The $\overline{\text{DRTRY}}$ signal extends data bus mastership such that other processors cannot use the data bus until $\overline{\text{DRTRY}}$ is negated. Therefore, in the example in Figure 8-12, $\overline{\text{DBB}}$ cannot be asserted until bus clock cycle 5. This is true for both read and write operations even though $\overline{\text{DRTRY}}$ does not extend bus mastership for write operations.

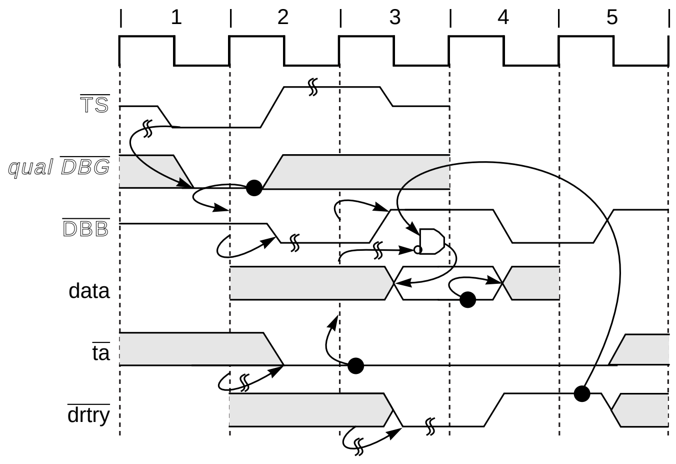


Figure 8-12. Termination with $\overline{\text{DRTRY}}$

Figure 8-13 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. Notice that in bus clock cycle 3 of Figure 8-13, $\overline{\text{TA}}$ is negated for the second data beat. The 603e data pipeline does not proceed until bus clock cycle 4 when the $\overline{\text{TA}}$ is reasserted.

Note that $\overline{\text{DRTRY}}$ is useful for systems that implement predicted forwarding of data such as 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.

Note that $\overline{\text{DRTRY}}$ may not be implemented on other PowerPC processors.

8.4.4.2 Data Transfer Termination Due to a Bus Error

The $\overline{\text{TEA}}$ signal indicates that a bus error occurred. It may be asserted while $\overline{\text{DBB}}$ (and/or $\overline{\text{DRTRY}}$ for read operations) is asserted. Asserting $\overline{\text{TEA}}$ to the 603e terminates the transaction; that is, further assertions of $\overline{\text{TA}}$ and $\overline{\text{DRTRY}}$ are ignored and $\overline{\text{DBB}}$ is negated; see Figure 8-13.

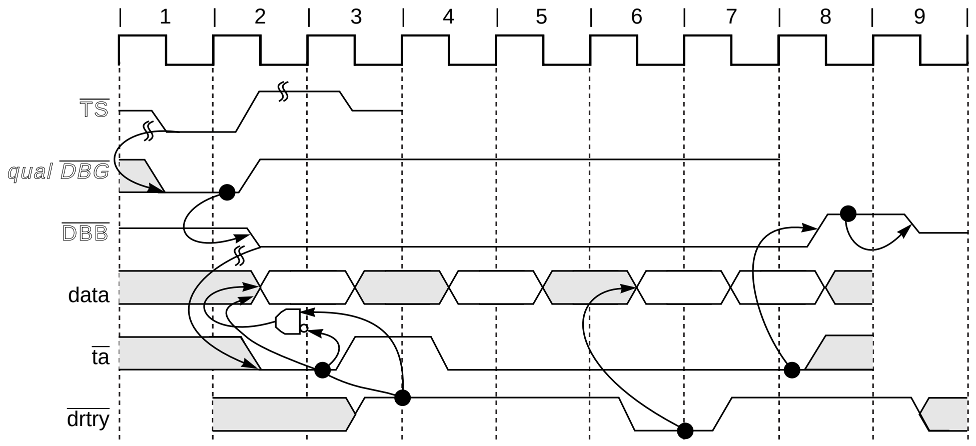


Figure 8-13. Read Burst with $\overline{\text{TA}}$ Wait States and $\overline{\text{DRTRY}}$

Assertion of the $\overline{\text{TEA}}$ signal causes a machine check exception (and possibly a checkstop condition within the 603e). For more information, see Section 4.5.2, "Machine Check Exception (0x00200)." Note also that the 603e does not implement a synchronous error capability for memory accesses. This means that the exception instruction pointer does not point to the memory operation 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 603e. To recover, the exception handler must determine and remedy the cause of the $\overline{\text{TEA}}$, or the 603e must be reset; therefore, this function should only be used to flag fatal system conditions to the processor (such as parity or uncorrectable ECC errors).

After the 603e has committed to run a transaction, that transaction must eventually complete. Address retry causes the transaction to be restarted; $\overline{\text{TA}}$ wait states and $\overline{\text{DRTRY}}$ assertion for reads delay termination of individual data beats. Eventually, however, the system must either terminate the transaction or assert the $\overline{\text{TEA}}$ signal (and vector the 603e into a machine check exception.) For this reason, care must be taken to check for the end of physical memory and the location of certain system facilities to avoid memory accesses that result in the generation of machine check exceptions.

Note that $\overline{\text{TEA}}$ generates a machine check exception depending on the ME bit in the MSR. Clearing the machine check exception enable control bits leads to a true checkstop condition (instruction execution halted and processor clock stopped).

8.4.5 Memory Coherency—MEI Protocol

The 603e provides dedicated hardware to provide memory coherency by snooping bus transactions. The address retry capability enforces the three-state, MEI cache-coherency protocol (see Figure 8-14).

The global ($\overline{\text{GBL}}$) output signal indicates whether the current transaction must be snooped by other snooping devices on the bus. Address bus masters assert $\overline{\text{GBL}}$ to indicate that the current transaction is a global access (that is, an access to memory shared by more than one device). If $\overline{\text{GBL}}$ is not asserted for the transaction, that transaction is not snooped. When other devices detect the $\overline{\text{GBL}}$ input asserted, they must respond by snooping the broadcast address.

Normally, $\overline{\text{GBL}}$ reflects the M bit value specified for the memory reference in the corresponding translation descriptor(s). Note that care must be taken to minimize the number of pages marked as global, because the retry protocol discussed in the previous section is used to enforce coherency and can require significant bus bandwidth.

When the 603e is not the address bus master, $\overline{\text{GBL}}$ is an input. The 603e 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 603e cache occurs if the snooped transaction is not marked global. This includes invalidation cycles.

When the 603e detects a qualified snoop condition, the address associated with the $\overline{\text{TS}}$ is compared against the data cache tags. Snooping completes if no hit is detected. If, however, the address hits in the cache, the 603e reacts according to the MEI protocol shown in Figure 8-14, assuming the WIM bits are set to write-back, caching-allowed, and coherency-enforced modes (WIM = 001).

The 603e's on-chip data cache is implemented as a four-way set-associative cache. To facilitate external monitoring of the internal cache tags, the cache set entry (CSE[0–1]) signals indicate which cache set is being replaced on read operations. Note that these signals are valid only for 603e burst operations; for all other bus operations, the CSE[0–1] signals should be ignored.

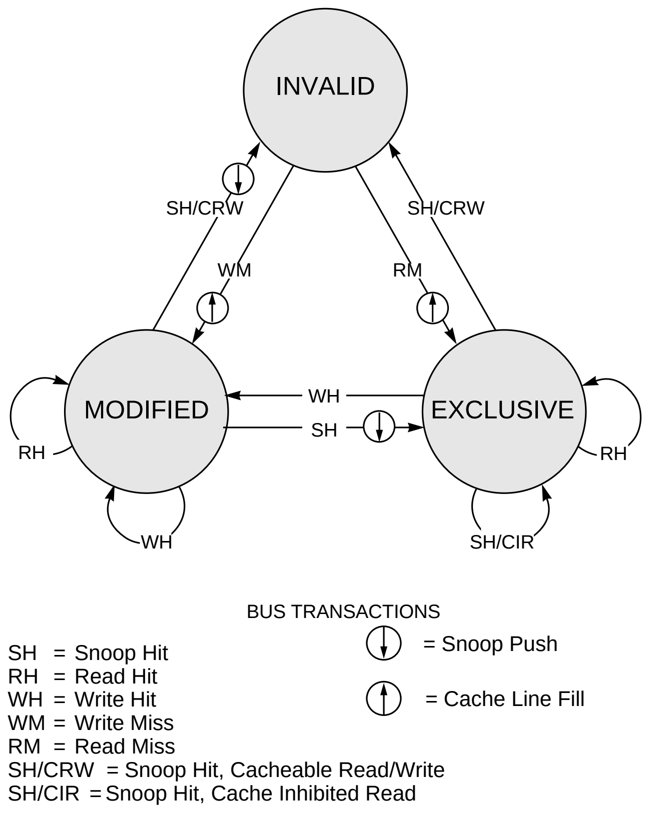


Figure 8-14. MEI Cache Coherency Protocol—State Diagram (WIM = 001)

Table 8-9 shows the CSE encodings.

Table 8-9. CSE[0–1] Signals

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

8.5 Timing Examples

This section shows timing diagrams for various scenarios. Figure 8-15 illustrates the fastest single-beat reads possible for the 603e. This figure shows both minimal latency and maximum single-beat throughput. By delaying the data bus tenure, the latency increases, but, because of split-transaction pipelining, the overall throughput is not affected unless the data bus latency causes the third address tenure to be delayed.

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

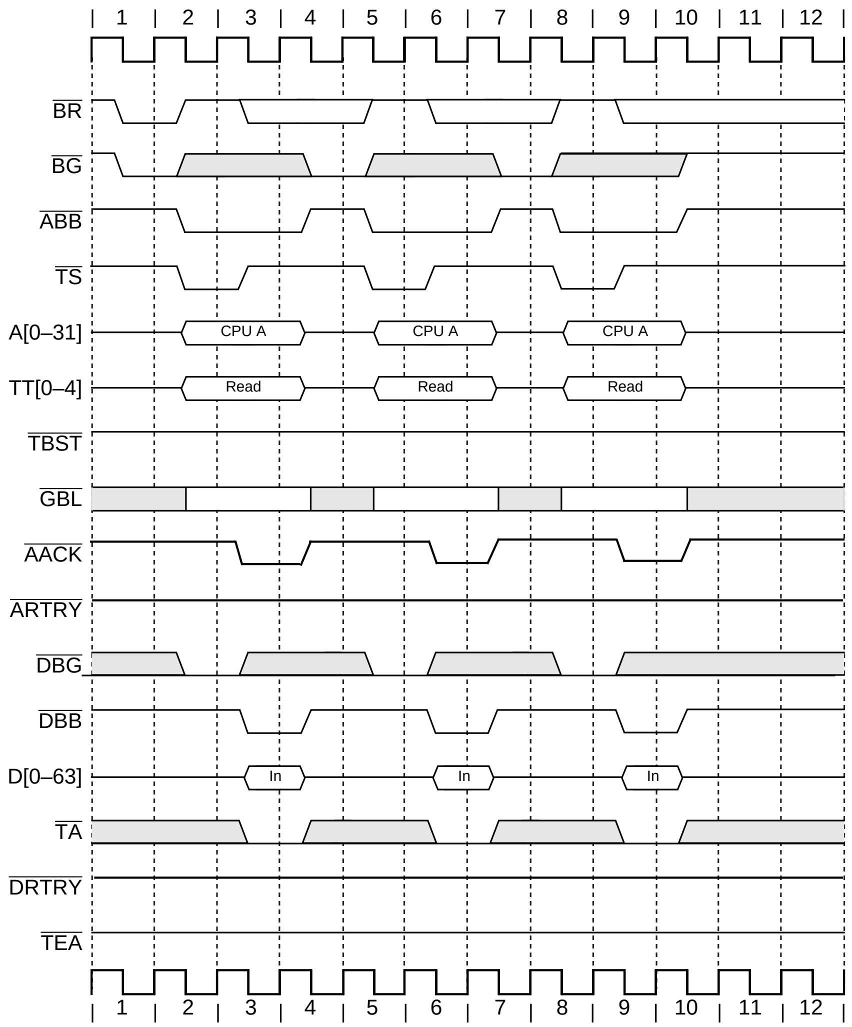


Figure 8-15. Fastest Single-Beat Reads

Figure 8-16 illustrates the fastest single-beat writes supported by the 603e. All bidirectional signals are three-stated between bus tenures.

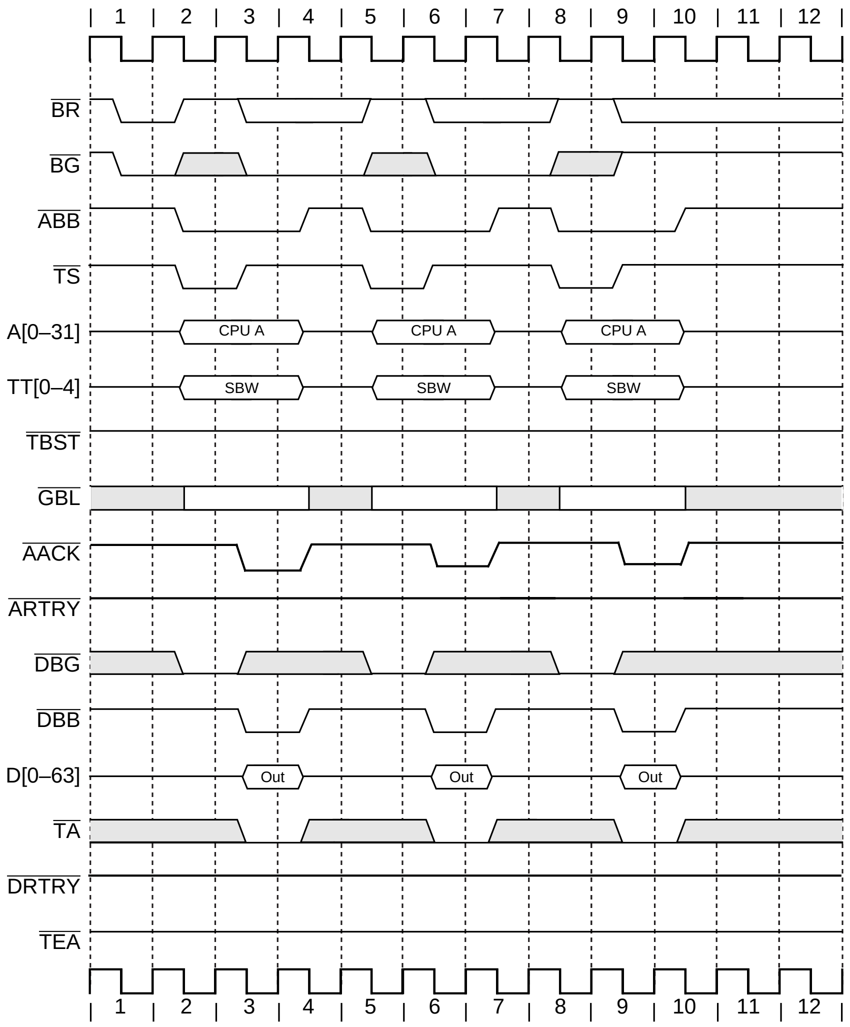


Figure 8-16. Fastest Single-Beat Writes

Figure 8-17 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. The pipelining shown in Figure 8-17 can occur if the second access is not another load (for example, an instruction fetch).

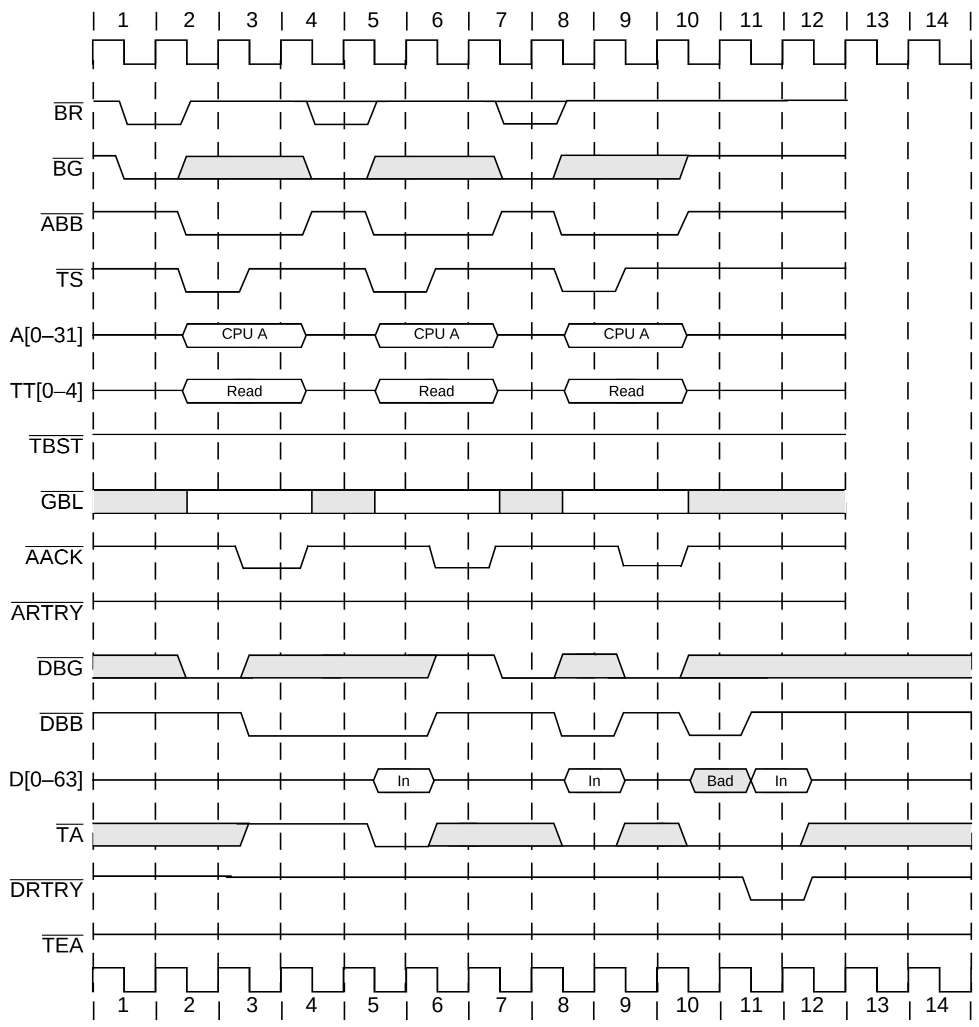


Figure 8-17. Single-Beat Reads Showing Data-Delay Controls

Figure 8-18 shows data-delay controls in a single-beat write operation. Note that all bidirectional signals are three-stated between bus tenures. Data transfers are delayed in the following ways:

- The $\overline{\text{TA}}$ signal is held negated to insert wait states in clocks 3 and 4.
- In clock 6, $\overline{\text{DBG}}$ is held negated, delaying the start of the data tenure.

The last access is not delayed ($\overline{\text{DRTRY}}$ is valid only for read operations).

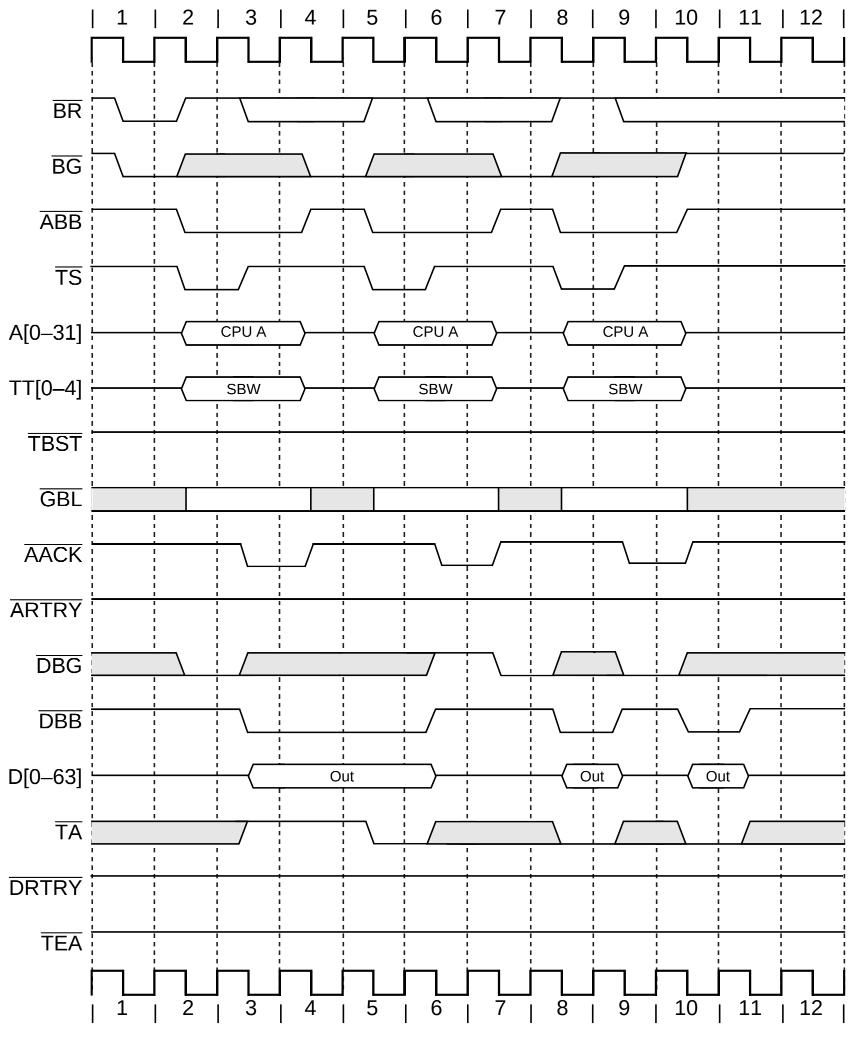


Figure 8-18. Single-Beat Writes Showing Data Delay Controls

Figure 8-19 shows the use of data-delay controls with burst transfers. Note that all bidirectional signals are three-stated between bus tenures. Note the following:

- The first data beat of bursted read data (clock 0) is the critical quad word.
- The write burst shows the use of $\overline{\text{TA}}$ signal negation to delay the third data beat.
- The final read burst shows the use of $\overline{\text{DRTRY}}$ on the third data beat.
- The address for the third transfer is delayed until the first transfer completes.

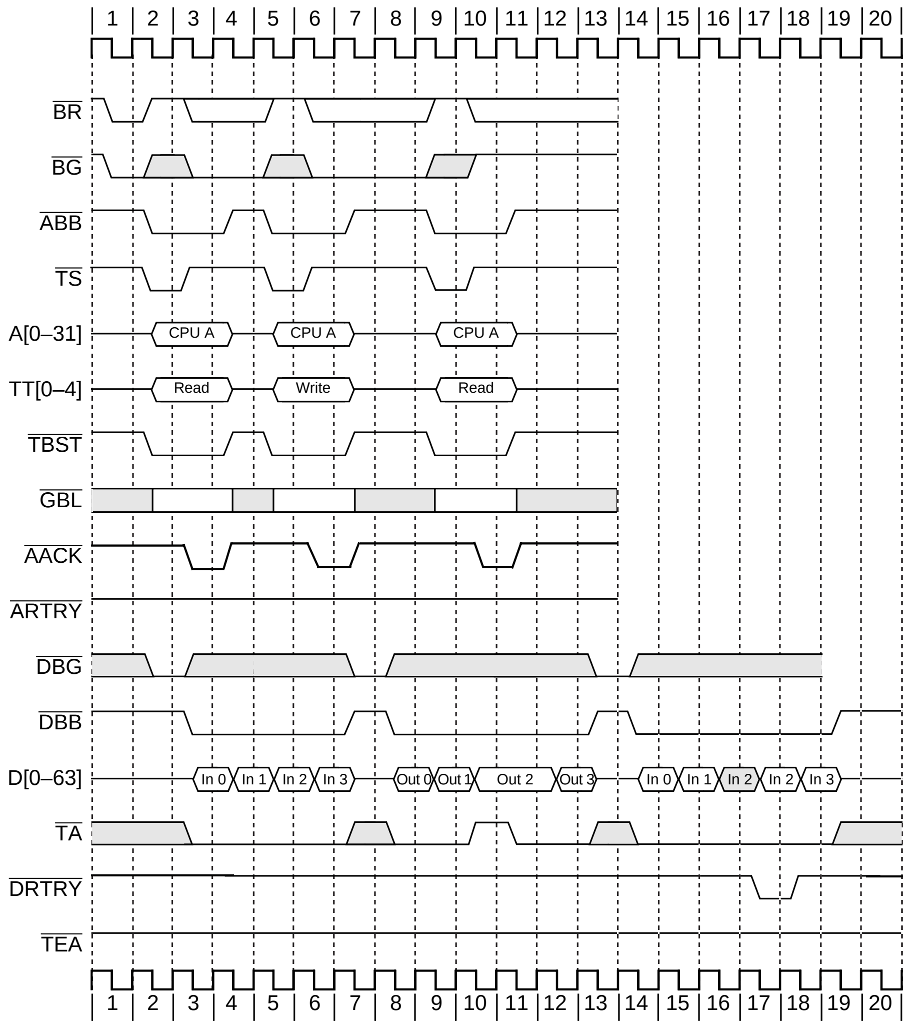


Figure 8-19. Burst Transfers with Data Delay Controls

Figure 8-20 shows the use of the $\overline{\text{TEA}}$ 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 0) is the critical quad word.
- The $\overline{\text{TEA}}$ signal truncates the burst write transfer on the third data beat.
- The 603e eventually causes an exception to be taken on the $\overline{\text{TEA}}$ event.

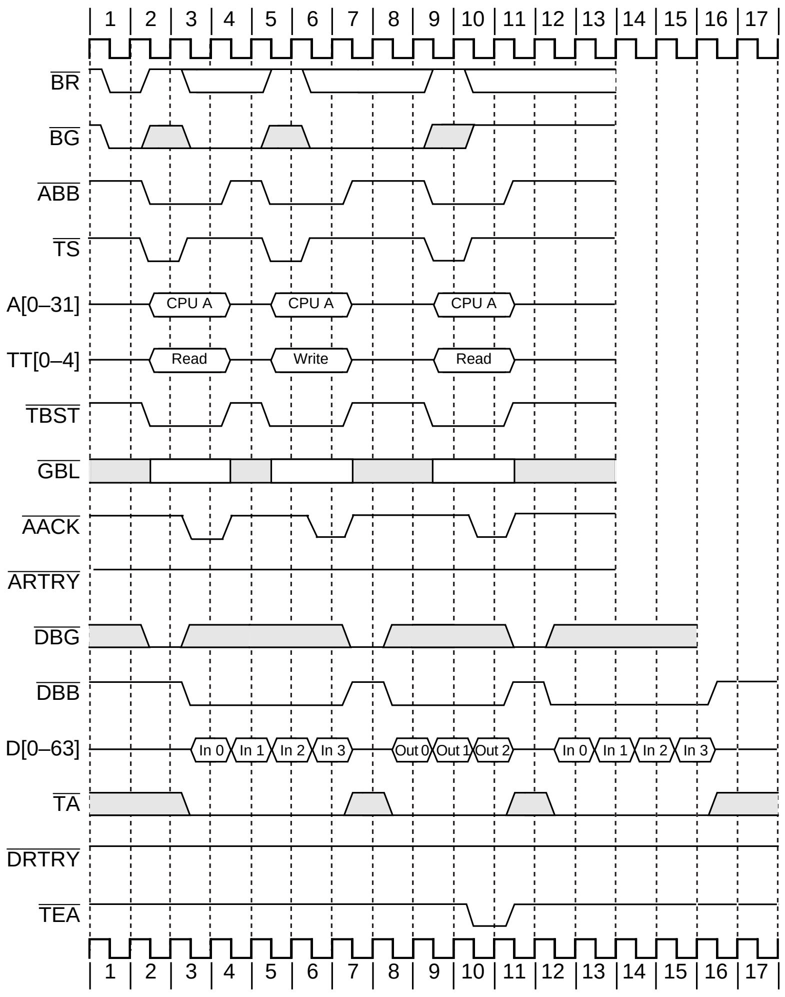


Figure 8-20. Use of Transfer Error Acknowledge ($\overline{\text{TEA}}$)

8.6 Optional Bus Configurations

The 603e supports three optional bus configurations that are selected by the assertion or negation of $\overline{\text{DRTRY}}$, $\overline{\text{TLBISYNC}}$, and $\overline{\text{QACK}}$ signals during the negation of the $\overline{\text{HRESET}}$ signal. The operation and selection of the optional bus configurations are described in the following sections.

8.6.1 32-Bit Data Bus Mode

The 603e supports an optional 32-bit data bus mode. The 32-bit data bus mode operates the same as the 64-bit data bus mode with the exception of the byte lanes involved in the transfer and the number of data beats that are performed. When in 32-bit data bus mode, only byte lanes 0 through 3 are used corresponding to DH0–DH31 and DP0–DP3. Byte lanes 4 through 7 corresponding to DL0–DL31 and DP4–DP7 are never used in this mode. The unused data bus signals are not sampled by the 603e during read operations, and they are driven low during write operations.

The number of data beats required for a data tenure in the 32-bit data bus mode is one, two, or eight beats depending on the size of the program transaction and the cache mode for the address. Data transactions of one or two data beats are performed for caching-inhibited load/store or write-through store operations. These transactions do not assert the $\overline{\text{TBST}}$ signal even though a two-beat burst may be performed (having the same $\overline{\text{TBST}}$ and TSIZ[0–2] encodings as the 64-bit data bus mode). Single-beat data transactions are performed for bus operations of 4 bytes or less, and double-beat data transactions are performed for 8-byte operations only. The 603e only generates an 8-byte operation for a double-word-aligned load or store double operation to or from the floating-point GPRs (not supported on the EC603e microprocessor). All cache-inhibited instruction fetches are performed as word (single-beat) operations.

Data transactions of eight data beats are performed for burst operations that load into or store from the 603e's internal caches. These transactions transfer 32 bytes in the same way as in 64-bit data bus mode, asserting the $\overline{\text{TBST}}$ signal, and signaling a transfer size of 2 (TSIZ(0–2) = 0b010).

The same bus protocols apply for arbitration, transfer, and termination of the address and data tenures in the 32-bit data bus mode as they apply to the 64-bit data bus mode. 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 word or smaller transactions, or on the bus clock after the second data beat is acknowledged (after the second $\overline{\text{TA}}$) for double-word or burst operations (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 603e is in 32-bit data bus mode is shown in Figure 8-21.

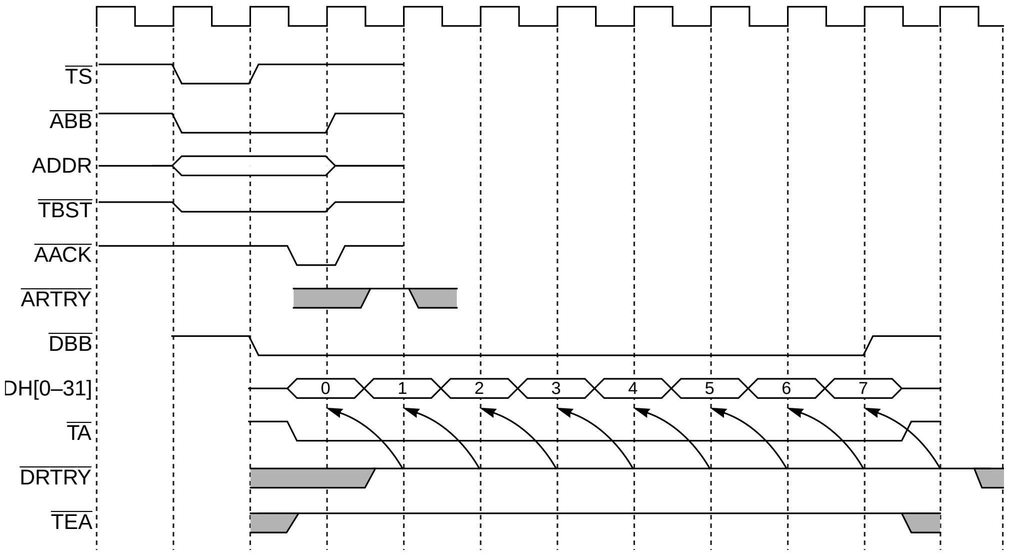


Figure 8-21. 32-Bit Data Bus Transfer (Eight-Beat Burst)

An example of a two-beat data transfer (with $\overline{\text{DRTRY}}$ asserted during each data tenure) is shown in Figure 8-22.

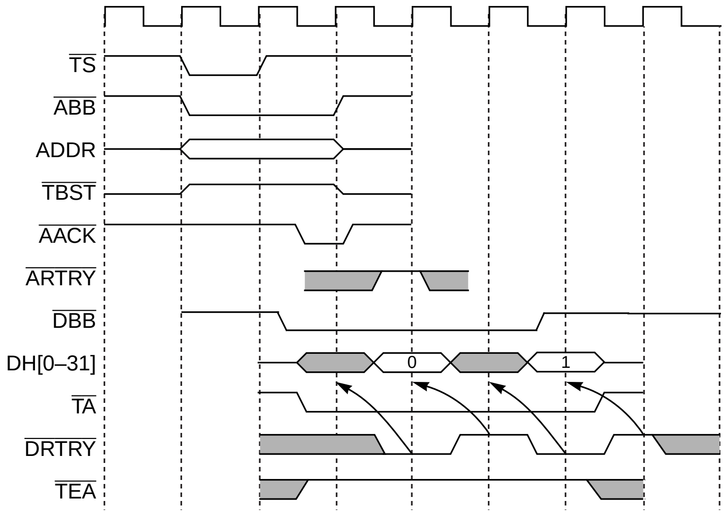


Figure 8-22. 32-Bit Data Bus Transfer (Two-Beat Burst with $\overline{\text{DRTRY}}$)

The 603e selects 64-bit or 32-bit data bus mode at startup by sampling the state of the $\overline{\text{TLBISYNC}}$ signal at the negation of $\overline{\text{HRESET}}$. If the $\overline{\text{TLBISYNC}}$ signal is negated at the negation of $\overline{\text{HRESET}}$, 64-bit data mode is entered by the 603e. If $\overline{\text{TLBISYNC}}$ is asserted at the negation of $\overline{\text{HRESET}}$, 32-bit data mode is entered.

8.6.2 No-$\overline{\text{DRTRY}}$ Mode

The 603e supports an optional mode to disable the use of the data retry function provided through the $\overline{\text{DRTRY}}$ signal. The no-$\overline{\text{DRTRY}}$ mode allows the forwarding of data during load operations to the internal CPU one bus cycle sooner than in the normal bus protocol.

The PowerPC bus protocol specifies that, during load operations, the memory system normally has the capability to cancel data that was read by the master on the bus cycle after $\overline{\text{TA}}$ was asserted. In the 603e implementation, this late cancellation protocol requires the 603e to hold any loaded data at the bus interface for one additional bus clock to verify that the data is valid before forwarding it to the internal CPU. For systems that do not implement the $\overline{\text{DRTRY}}$ function, the 603e provides an optional no-$\overline{\text{DRTRY}}$ mode that eliminates this one-cycle stall during all load operations, and allows for the forwarding of data to the internal CPU immediately when $\overline{\text{TA}}$ is recognized.

When the 603e is in the 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 603e.

When the 603e is following normal bus protocol, data may be cancelled the bus cycle after $\overline{\text{TA}}$ by either of two means—late cancellation by $\overline{\text{DRTRY}}$, or late cancellation 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.

When no-$\overline{\text{DRTRY}}$ mode is selected for the 603e, the system must ensure that $\overline{\text{DRTRY}}$ will not be asserted to the 603e. If it is asserted, it may cause improper operation of the bus interface. The system must also ensure that an assertion of $\overline{\text{ARTRY}}$ by a snooping device must occur before or coincident with the first assertion of $\overline{\text{TA}}$ to the 603e, but not on the cycle after the first assertion of $\overline{\text{TA}}$.

Other than the inability to cancel data that was read by the master on the bus cycle after $\overline{\text{TA}}$ was asserted, the bus protocol for the 603e is identical to that for the basic transfer bus protocols described in this chapter, as well as for 32-bit data bus mode.

The 603e selects the desired $\overline{\text{DRTRY}}$ mode at startup by sampling the state of the $\overline{\text{DRTRY}}$ signal itself at the negation of the $\overline{\text{HRESET}}$ signal. If the $\overline{\text{DRTRY}}$ signal is negated at the negation of $\overline{\text{HRESET}}$, normal operation is selected. If the $\overline{\text{DRTRY}}$ signal is asserted at the negation of $\overline{\text{HRESET}}$, no-$\overline{\text{DRTRY}}$ mode is selected.

8.6.3 Reduced-Pinout Mode

The 603e provides 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], $\overline{\text{APE}}$, $\overline{\text{DPE}}$, and $\overline{\text{RSRV}}$ 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 in the reduced-pinout mode, the bidirectional and output signals disabled are always driven low during the periods when they would normally have been driven by the 603e. The open-drain outputs (APE and DPE) are always three-stated. The bidirectional inputs are always turned-off at the input receivers of the 603e and are not sampled.

The 603e selects either full-pinout or reduced-pinout mode at startup by sampling the state of the QACK signal at the negation of HRESET. If the QACK signal is asserted at the negation of HRESET, full-pinout mode is selected by the 603e. If QACK is negated at the negation of HRESET, reduced-pinout mode is selected.

8.7 Interrupt, Checkstop, and Reset Signals

This section describes external interrupts, checkstop operations, and hard and soft reset inputs.

8.7.1 External Interrupts

The external interrupt input signals (INT, SMI and MCP) of the 603e eventually force the processor to take the external interrupt vector, or the system management interrupt vector if the MSR[EE] is set, or the machine check interrupt if the MSR[ME] bit and the HID0[EMCP] bit are set.

8.7.2 Checkstops

The 603e has two checkstop input signals—CKSTP_IN (non-maskable) and MCP (enabled when MSR[ME] is cleared, and HID0[EMCP] is set), and a checkstop output (CKSTP_OUT). If CKSTP_IN or MCP is asserted, the 603e halts operations by gating off all internal clocks. The 603e asserts CKSTP_OUT if CKSTP_IN is asserted.

If CHECKSTOP is asserted by the 603e, it has entered the checkstop state, and processing has halted internally. The CHECKSTOP signal can be asserted for various reasons including receiving a TEA signal and detection of external parity errors. For more information about checkstop state, see Section 4.5.2.2, “Checkstop State (MSR[ME] = 0).”

8.7.3 Reset Inputs

The 603e has two reset inputs, described as follows:

- HRESET (hard reset)—The HRESET signal is used for power-on reset sequences, or for situations in which the 603e must go through the entire cold-start sequence of internal hardware initializations.
- SRESET (soft reset)—The soft reset input provides warm reset capability. This input can be used to avoid forcing the 603e to complete the cold start sequence.

When either reset input is negated, the processor attempts to fetch code from the system reset exception vector. The vector is located at offset 0x00100 from the exception prefix (all zeros or ones, depending on the setting of the exception prefix bit in the machine state register (MSR[IP]). The IP bit is set for HRESET.

8.7.4 System Quiesce Control Signals

The system quiesce control signals ($\overline{\mathrm{QREQ}}$ and $\overline{\mathrm{QACK}}$) allow the processor to enter a low power state, and bring bus activity to a quiescent state in an orderly fashion.

The system quiesce state is entered by asserting the $\overline{\mathrm{QREQ}}$ signal. This signal allows the system to terminate or pause any bus activities that are normally snooped. When the system is ready to enter the system quiesce state, it asserts the $\overline{\mathrm{QACK}}$ signal. At this time the 603e may enter a quiescent (low power) state. When the 603e is in the quiescent state, it stops snooping bus activity.

8.8 Processor State Signals

This section describes the 603e's support for atomic update and memory through the use of the **lwarx**/**stwcx**. opcode pair, and includes a description of the 603e TLBISYNC input.

8.8.1 Support for the lwarx/stwcx. Instruction Pair

The Load Word and Reserve Indexed (**lwarx**) and the Store Word Conditional Indexed (**stwcx**.) instructions provide a means for atomic memory updating. Memory can be updated atomically by setting a reservation on the load and checking that the reservation is still valid before the store is performed. In the 603e, the reservations are made on behalf of aligned, 32-byte sections of the memory address space.

The reservation ($\overline{\mathrm{RSRV}}$) output signal is driven synchronously with the bus clock and reflects the status of the reservation coherency bit in the reservation address register (see Chapter 3, “Instruction and Data Cache Operation,” for more information). See Section 7.2.9.7.3, “Reservation (RSRV)—Output,” for information about timing.

8.8.2 $\overline{\mathrm{TLBISYNC}}$ Input

The $\overline{\mathrm{TLBISYNC}}$ input allows for the hardware synchronization of changes to MMU tables when the 603e and another DMA master share the same MMU translation tables in system memory. It is asserted by a DMA master when it is using shared addresses that could be changed in the MMU tables by the 603e during the DMA master’s tenure.

The $\overline{\mathrm{TLBISYNC}}$ input, when asserted to the 603e, prevents the 603e from completing any instructions past a **tlbsync** instruction. Generally, during the execution of an **eciwx** or **ecowx** instruction by the 603e, the selected DMA device should assert the 603e’s $\overline{\mathrm{TLBISYNC}}$ signal and maintain it asserted during its DMA tenure if it is using a shared translation address. Subsequent instructions by the 603e should include a **sync** and **tlbsync** instruction before any MMU table changes are performed. This will prevent the 603e from making table changes disruptive to the other master during the DMA period.

8.9 IEEE 1149.1-Compliant Interface

The 603e boundary-scan interface is a fully-compliant implementation of the IEEE 1149.1 standard. This section describes the 603e IEEE 1149.1(JTAG) interface.

8.9.1 IEEE 1149.1 Interface Description

The 603e has five dedicated JTAG signals which are described in Table 8-10. The TDI and TDO scan ports are used to scan instructions as well as data into the various scan registers for JTAG operations. The scan operation is controlled by the test access port (TAP) controller which in turn is controlled by the TMS input sequence. The scan data is latched in at the rising edge of TCK.

Table 8-10. IEEE Interface Pin 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 which is used to reset the TAP controller asynchronously. The $\overline{\text{TRST}}$ signal assures that the JTAG logic does not interfere with the normal operation of the chip, and can be asserted coincident with the $\overline{\text{HRESET}}$.

The PID7v-603e implements the JTAG/COP in the same manner as does the PID6-603e implementation with the exception of the introduction of the 33-bit Run_N counter register in which the most-significant 32 bits form a 32-bit counter. The function of the least-significant bit remains unchanged. The Run_N counter is used by the COP to control the number of processor cycles that the processor runs before halting.

8.10 Using Data Bus Write Only

The 603e supports split-transaction pipelined transactions. It supports a limited out-of-order capability for its own pipelined transactions through the data bus write only ($\overline{\text{DBWO}}$) signal. When recognized on the clock of a qualified $\overline{\text{DBG}}$, the assertion of $\overline{\text{DBWO}}$ directs the 603e to perform the next pending data write tenure (if any), even if a pending read tenure would have normally been performed because of address pipelining. The $\overline{\text{DBWO}}$ signal does not change the order of write tenures with respect to other write tenures from the same 603e. It only allows that a write tenure be performed ahead of a pending read tenure from the same 603e.

In general, an address tenure on the bus is followed strictly in order by its associated data tenure. Transactions pipelined by the 603e complete strictly in order. However, the 603e can run bus transactions out of order only when the external system allows the 603e to perform a cache-line-snoop-push-out operation (or other write transaction, if pending in the 603e write queues) between the address and data tenures of a read operation through the use of $\overline{\text{DBWO}}$. This effectively envelopes the write operation within the read operation. Figure 8-23 shows how the $\overline{\text{DBWO}}$ signal is used to perform an enveloped write transaction.

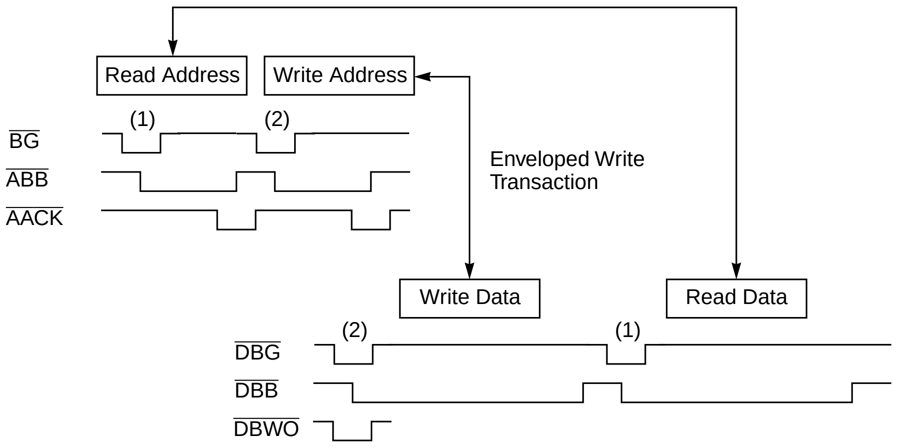


Figure 8-23. Data Bus Write Only Transaction

Note that although the 603e can pipeline any write transaction behind the read transaction, special care should be used when using the enveloped write feature. It is envisioned that most system implementations will not need this capability; for these applications, $\overline{\text{DBWO}}$ should remain negated. In systems where this capability is needed, $\overline{\text{DBWO}}$ should be asserted under the following scenario:

1. The 603e initiates a read transaction (either single-beat or burst) by completing the read address tenure with no address retry.
2. Then, the 603e initiates a write transaction by completing the write address tenure, with no address retry.
3. At this point, if $\overline{\text{DBWO}}$ is asserted with a qualified data bus grant to the 603e, the 603e asserts $\overline{\text{DBB}}$ and drives the write data onto the data bus, out of order with respect to the address pipeline. The write transaction concludes with the 603e negating $\overline{\text{DBB}}$.
4. The next qualified data bus grant signals the 603e to complete the outstanding read transaction by latching the data on the bus. This assertion of $\overline{\text{DBG}}$ should not be accompanied by an asserted $\overline{\text{DBWO}}$.

Any number of bus transactions by other bus masters can be attempted between any of these steps.

Note the following regarding $\overline{\text{DBWO}}$:

- $\overline{\text{DBWO}}$ can be asserted if no data bus read is pending, but it has no effect on write ordering.
- The ordering and presence of data bus writes is determined by the writes in the write queues at the time $\overline{\text{BG}}$ is asserted for the write address (not $\overline{\text{DBG}}$). If a particular write is desired (for example, a cache-line-snoop-push-out operation), then $\overline{\text{BG}}$ must be asserted after that particular write is in the queue and it must be the highest priority write in the queue at that time. A cache-line-snoop-push-out operations may be the highest priority write, but more than one may be queued.
- Because more than one write may be in the write queue when $\overline{\text{DBG}}$ is asserted for the write address, more than one data bus write may be enveloped by a pending data bus read.

The arbiter must monitor bus operations and coordinate the various masters and slaves with respect to the use of the data bus when $\overline{\text{DBWO}}$ is used. Individual $\overline{\text{DBG}}$ signals associated with each bus device should allow the arbiter to synchronize both pipelined and split-transaction bus organizations. Individual $\overline{\text{DBG}}$ and $\overline{\text{DBWO}}$ signals provide a primitive form of source-level tagging for the granting of the data bus.

Note that use of the $\overline{\text{DBWO}}$ signal allows some operation-level tagging with respect to the 603e and the use of the data bus.

Chapter 9
Power Management

The PowerPC 603e microprocessor is the first microprocessor specifically designed for low-power operation. The 603e provides both automatic and program-controllable power reduction modes for progressive reduction of power consumption. This chapter describes the hardware support provided by the 603e for power management.

9.1 Dynamic Power Management

Dynamic power management automatically powers up and down the individual execution units of the 603e, based upon the contents of the instruction stream. For example, if no floating-point instructions are being executed, the floating-point unit is automatically powered down. Power is not actually removed from the execution unit; instead, each execution unit has an independent clock input, which is automatically controlled on a clock-by-clock basis. Since CMOS circuits consume negligible power when they are not switching, stopping the clock to an execution unit effectively eliminates its power consumption. The operation of DPM is completely transparent to software or any external hardware. Dynamic power management is enabled by setting bit 11 in HID0 on power-up, or following $\overline{\text{HRESET}}$.

9.2 Programmable Power Modes

The 603e provides four power modes selectable by setting the appropriate control bits in the machine state register (MSR) and hardware implementation register 0 (HID0) registers. The four power modes are described briefly as follows:

- Full-power—This is the default power state of the 603e. The 603e is fully powered and the internal functional units are operating at the full processor clock speed. If the dynamic power management mode is enabled, functional units that are idle will automatically enter a low-power state without affecting performance, software execution, or external hardware.
- Doze—All the functional units of the 603e are disabled except for the time base/ decrementer registers and the bus snooping logic. When the processor is in doze mode, an external asynchronous interrupt, a system management interrupt, a decrementer exception, a hard or soft reset, or machine check brings the 603e into

the full-power state. The 603e in doze mode maintains the PLL in a fully powered state and locked to the system external clock input (SYSCLK) so a transition to the full-power state takes only a few processor clock cycles.

- Nap—The nap mode further reduces power consumption by disabling bus snooping, leaving only the time base register and the PLL in a powered state. The 603e returns to the full-power state upon receipt of an external asynchronous interrupt, a system management interrupt, a decrementer exception, a hard or soft reset, or a machine check input ($\overline{\text{MCP}}$) signal. A return to full-power state from a nap state takes only a few processor clock cycles.
- Sleep—Sleep mode reduces power consumption to a minimum by disabling all internal functional units, after which external system logic may disable the PLL and SYSCLK. Returning the 603e to the full-power state requires the enabling of the PLL and SYSCLK, followed by the assertion of an external asynchronous interrupt, a system management interrupt, a hard or soft reset, or a machine check input ($\overline{\text{MCP}}$) signal after the time required to relock the PLL.

The PID7v-603e implementation offers the following enhancements to the 603e family:

- Lower-power design
- 2.5-volt core and 3.3-volt I/O

Hardware can enable a power management state through external asynchronous interrupts. The hardware interrupt causes the transfer of program flow to interrupt handler code. The appropriate mode is then set by the software. The 603e provides a separate interrupt and interrupt vector for power management—the system management interrupt (SMI). The 603e also contains a decrement timer which allows it to enter the nap or doze mode for a predetermined amount of time and then return to full power operation through the decrementer interrupt exception. Note that the 603e cannot switch from one power management mode to another without first returning to full-on mode. The nap and sleep modes disable bus snooping; therefore, a hardware handshake is provided to ensure coherency before the 603e enters these power management modes. Table 9-1 summarizes the four power states.

Table 9-1. Programmable Power Modes

PM Mode	Functioning Units	Activation Method	Full-Power Wake Up Method
Full power	All units active	—	—
Full power (with DPM)	Requested logic by demand	By instruction dispatch	—
Doze	• Bus snooping • Data cache as needed • Decrementer timer	Controlled by SW	External asynchronous exceptions* Decrementer interrupt Reset
Nap	Decrementer timer	Controlled by hardware and software	External asynchronous exceptions Decrementer interrupt Reset
Sleep	None	Controlled by hardware and software	External asynchronous exceptions Reset

9.2.1 Power Management Modes

The following sections describe the characteristics of the 603e's power management modes, the requirements for entering and exiting the various modes, and the system capabilities provided by the 603e while the power management modes are active.

9.2.1.1 Full-Power Mode with DPM Disabled

Full-power mode with DPM disabled power mode is selected when the DPM enable bit (bit 11) in HID0 is cleared.

- Default state following power-up and $\overline{\text{HRESET}}$
- All functional units are operating at full processor speed at all times

9.2.1.2 Full-Power Mode with DPM Enabled

Full-power mode with DPM enabled (HID0[11] = 1) provides on-chip power management without affecting the functionality or performance of the 603e.

- Required functional units are operating at full processor speed
- Functional units are clocked only when needed
- No software or hardware intervention required after mode is set
- Software/hardware and performance transparent

9.2.1.3 Doze Mode

Doze mode disables most functional units but maintains cache coherency by enabling the bus interface unit and snooping. A snoop hit will cause the 603e to enable the data cache, copy the data back to memory, disable the cache, and fully return to the doze state.

- Most functional units disabled
- Bus snooping and time base/decrementer still enabled
- Doze mode sequence
  - Set doze bit (HID0[8] = 1)
  - 603e enters doze mode after several processor clocks
- Several methods of returning to full-power mode
  - Assert $\overline{\text{INT}}$, $\overline{\text{SMI}}$, $\overline{\text{MCP}}$ or decrementer interrupts
  - Assert hard reset or soft reset
- Transition to full-power state takes no more than a few processor cycles
- PLL running and locked to SYSCLK

9.2.1.4 Nap Mode

The nap mode disables the 603e but still maintains the phase-locked loop (PLL) and the time base/decrementer. The time base can be used to restore the 603e to full-on state after a programmed amount of time. Because bus snooping is disabled for nap and sleep mode, a hardware handshake using the quiesce request ($\overline{\text{QREQ}}$) and quiesce acknowledge ($\overline{\text{QACK}}$) signals are required to maintain data coherency. The 603e will assert the $\overline{\text{QREQ}}$ signal to indicate that it is ready to disable bus snooping. When the system has ensured that snooping is no longer necessary, it will assert $\overline{\text{QACK}}$ and the 603e will enter the sleep or nap mode.

- Time base/decrementer still enabled
- Most functional units disabled (including bus snooping)
- All nonessential input receivers disabled
- Nap mode sequence
  - Set nap bit (HID0[9] = 1)
  - 603e asserts quiesce request ($\overline{\text{QREQ}}$) signal
  - System asserts quiesce acknowledge ($\overline{\text{QACK}}$) signal
  - 603e enters sleep mode after several processor clocks
- Several methods of returning to full-power mode
  - Assert $\overline{\text{INT}}$, $\overline{\text{SMI}}$, $\overline{\text{MCP}}$ or decrementer interrupts
  - Assert hard reset or soft reset
- Transition to full-power takes no more than a few processor cycles
- PLL running and locked to SYSCLK

9.2.1.5 Sleep Mode

Sleep mode consumes the least amount of power of the four modes since all functional units are disabled. To conserve the maximum amount of power, the PLL may be disabled and the SYSCLK may be removed. Due to the fully static design of the 603e, internal processor state is preserved when no internal clock is present. Because the time base and decrementer are disabled while the 603e is in sleep mode, the 603e’s time base contents will have to be updated from an external time base following sleep mode if accurate time-of-day maintenance is required. Before the 603e enters the sleep mode, the 603e will assert the $\overline{\text{QREQ}}$ signal to indicate that it is ready to disable bus snooping. When the system has ensured that snooping is no longer necessary, it will assert $\overline{\text{QACK}}$ and the 603e will enter the sleep mode.

- All functional units disabled (including bus snooping and time base)
- All nonessential input receivers disabled
  - — Internal clock regenerators disabled
  - — PLL still running (see below)
- Sleep mode sequence
  - — Set sleep bit (HID0[10] = 1)
  - — 603e asserts quiesce request ($\overline{\text{QREQ}}$)
  - — System asserts quiesce acknowledge ($\overline{\text{QACK}}$)
  - — 603e enters sleep mode after several processor clocks
- Several methods of returning to full-power mode
  - — Assert $\overline{\text{INT}}$, $\overline{\text{SMI}}$ or $\overline{\text{MCP}}$ interrupts
  - — Assert hard reset or soft reset
- PLL may be disabled and SYSCLK may be removed while in sleep mode
- Return to full-power mode after PLL and SYSCLK disabled in sleep mode
  - — Enable SYSCLK
  - — Reconfigure PLL into desired processor clock mode
  - — System logic waits for PLL startup and relock time (100 µsec)
  - — System logic asserts one of the sleep recovery signals (for example, INT or SMI)

9.2.2 Power Management Software Considerations

Since the 603e is a dual issue processor with out-of-order execution capability, care must be taken in how the power management mode is entered. Furthermore, nap and sleep modes require all outstanding bus operations to be completed before the power management mode is entered. Normally during system configuration time, one of the power management modes would be selected by setting the appropriate HID0 mode bit. Later on, the power management mode is invoked by setting the MSR[POW] bit. To ensure a clean transition into and out of the power management mode, set the MSR[EE] bit and execute the following code sequence:

```
        sync
        mtmsr[POW = 1]
        isync
loop:   b loop
```

Appendix A
PowerPC Instruction Set Listings

This appendix lists the PowerPC 603e microprocessor's instruction set as well as the additional PowerPC instructions not implemented in the 603e. Instructions are sorted by mnemonic, opcode, function, and form. Also included in this appendix is a quick reference table that contains general information, such as the architecture level, privilege level, and form, and indicates if the instruction is 64-bit and optional.

Note that split fields, that represent the concatenation of sequences from left to right, are shown in lowercase. For more information refer to Chapter 8, "Instruction Set," in *The Programming Environments Manual.*

A.1 Instructions Sorted by Mnemonic

Table A-1 lists the instructions implemented in the PowerPC architecture in alphabetical order by mnemonic.

Table A-1. Complete Instruction List Sorted by Mnemonic

Key:

▢	Reserved bits
▨	Instruction not implemented in the 603e

Name	0	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21	22 23 24 25 26 27 28 29 30	31
add*x*	31	D	A	B	OE	266	Rc
addc*x*	31	D	A	B	OE	10	Rc
adde*x*	31	D	A	B	OE	138	Rc
addi	14	D	A	SIMM			
addic	12	D	A	SIMM			
addic.	13	D	A	SIMM			
addis	15	D	A	SIMM			
addme*x*	31	D	A	0 0 0 0 0	OE	234	Rc
addze*x*	31	D	A	0 0 0 0 0	OE	202	Rc
and*x*	31	S	A	B	28		Rc
andc*x*	31	S	A	B	60		Rc

Name	0	6	9	10	11	16	21	22	30	31
andi.	28	S			A	UIMM				
andis.	29	S			A	UIMM				
b*x*	18	LI							AA	LK
bc*x*	16	BO			BI	BD			AA	LK
bcctr*x*	19	BO			BI	0 0 0 0 0	528			LK
bclr*x*	19	BO			BI	0 0 0 0 0	16			LK
cmp	31	crfD	0	L	A	B	0			0
cmpi	11	crfD	0	L	A	SIMM				
cmpl	31	crfD	0	L	A	B	32			0
cmpli	10	crfD	0	L	A	UIMM				
cntlzd*x* [4]	31	S			A	0 0 0 0 0	58			Rc
cntlzw*x*	31	S			A	0 0 0 0 0	26			Rc
crand	19	crbD			crbA	crbB	257			0
crandc	19	crbD			crbA	crbB	129			0
creqv	19	crbD			crbA	crbB	289			0
crnand	19	crbD			crbA	crbB	225			0
crnor	19	crbD			crbA	crbB	33			0
cror	19	crbD			crbA	crbB	449			0
crorc	19	crbD			crbA	crbB	417			0
crxor	19	crbD			crbA	crbB	193			0
dcbf	31	0 0 0 0 0			A	B	86			0
dcbi [1]	31	0 0 0 0 0			A	B	470			0
dcbst	31	0 0 0 0 0			A	B	54			0
dcbt	31	0 0 0 0 0			A	B	278			0
dcbtst	31	0 0 0 0 0			A	B	246			0
dcbz	31	0 0 0 0 0			A	B	1014			0
divd*x* [4]	31	D			A	B	OE	489		Rc
divdu*x* [4]	31	D			A	B	OE	457		Rc
divw*x*	31	D			A	B	OE	491		Rc
divwu*x*	31	D			A	B	OE	459		Rc
eciwx	31	D			A	B	310			0
ecowx	31	S			A	B	438			0
eieio	31	0 0 0 0 0			0 0 0 0 0	0 0 0 0 0	854			0

Name	0	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
eqv*x*	31	S	A	B	284		Rc
extsb*x*	31	S	A	0 0 0 0 0	954		Rc
extsh*x*	31	S	A	0 0 0 0 0	922		Rc
extsw*x* [4]	31	S	A	0 0 0 0 0	986		Rc
fabs*x*[7]	63	D	0 0 0 0 0	B	264		Rc
fadd*x*	63	D	A	B	0 0 0 0 0	21	Rc
fadds*x*[7]	59	D	A	B	0 0 0 0 0	21	Rc
fcfid*x* [4,7]	63	D	0 0 0 0 0	B	846		Rc
fcmpo[7]	63	crfD 0 0	A	B	32		0
fcmpu [7]	63	crfD 0 0	A	B	0		0
fctid*x* [4,7]	63	D	0 0 0 0 0	B	814		Rc
fctidz*x* [4,7]	63	D	0 0 0 0 0	B	815		Rc
fctiw*x*[7]	63	D	0 0 0 0 0	B	14		Rc
fctiwz*x*[7]	63	D	0 0 0 0 0	B	15		Rc
fdiv*x*[7]	63	D	A	B	0 0 0 0 0	18	Rc
fdivs*x*[7]	59	D	A	B	0 0 0 0 0	18	Rc
fmadd*x*[7]	63	D	A	B	C	29	Rc
fmadds*x*[7]	59	D	A	B	C	29	Rc
fmr*x*[7]	63	D	0 0 0 0 0	B	72		Rc
fmsub*x*[7]	63	D	A	B	C	28	Rc
fmsubs*x*[7]	59	D	A	B	C	28	Rc
fmul*x*[7]	63	D	A	0 0 0 0 0	C	25	Rc
fmuls*x*[7]	59	D	A	0 0 0 0 0	C	25	Rc
fnabs*x*[7]	63	D	0 0 0 0 0	B	136		Rc
fneg*x*[7]	63	D	0 0 0 0 0	B	40		Rc
fnmadd*x* [7]	63	D	A	B	C	31	Rc
fnmadds*x*[7]	59	D	A	B	C	31	Rc
fnmsub*x*[7]	63	D	A	B	C	30	Rc
fnmsubs*x*[7]	59	D	A	B	C	30	Rc
fres*x* [5,7]	59	D	0 0 0 0 0	B	0 0 0 0 0	24	Rc
frsp*x*[7]	63	D	0 0 0 0 0	B	12		Rc
frsqrte*x* [5,7]	63	D	0 0 0 0 0	B	0 0 0 0 0	26	Rc
fsel*x* [5,7]	63	D	A	B	C	23	Rc

Name	0	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
fsqrt*x*[5,7]	63	D	0 0 0 0 0	B	0 0 0 0 0	22	Rc
fsqrts*x*[5,7]	59	D	0 0 0 0 0	B	0 0 0 0 0	22	Rc
fsub*x*[7]	63	D	A	B	0 0 0 0 0	20	Rc
fsubs*x*[7]	59	D	A	B	0 0 0 0 0	20	Rc
icbi	31	0 0 0 0 0	A	B	982		0
isync	19	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	150		0
lbz	34	D	A	d			
lbzu	35	D	A	d			
lbzux	31	D	A	B	119		0
lbzx	31	D	A	B	87		0
ld[4]	58	D	A	ds			0
ldarx[4]	31	D	A	B	84		0
ldu[4]	58	D	A	ds			1
ldux[4]	31	D	A	B	53		0
ldx[4]	31	D	A	B	21		0
lfd[7]	50	D	A	d			
lfdu[7]	51	D	A	d			
lfdux[7]	31	D	A	B	631		0
lfdx[7]	31	D	A	B	599		0
lfs[7]	48	D	A	d			
lfsu[7]	49	D	A	d			
lfsux[7]	31	D	A	B	567		0
lfsx[7]	31	D	A	B	535		0
lha	42	D	A	d			
lhau	43	D	A	d			
lhaux	31	D	A	B	375		0
lhax	31	D	A	B	343		0
lhbrx	31	D	A	B	790		0
lhz	40	D	A	d			
lhzu	41	D	A	d			
lhzux	31	D	A	B	311		0
lhzx	31	D	A	B	279		0
lmw[3]	46	D	A	d			

Name	0	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
lswi [3]	31	D	A	NB	597	0
lswx [3]	31	D	A	B	533	0
lwa [4]	58	D	A	ds		2
lwarx	31	D	A	B	20	0
lwaux [4]	31	D	A	B	373	0
lwax [4]	31	D	A	B	341	0
lwbrx	31	D	A	B	534	0
lwz	32	D	A	d		
lwzu	33	D	A	d		
lwzux	31	D	A	B	55	0
lwzx	31	D	A	B	23	0
mcrf	19	crfD 0 0	crfS 0 0	0 0 0 0 0	0	0
mcrfs[7]	63	crfD 0 0	crfS 0 0	0 0 0 0 0	64	0
mcrxr	31	crfD 0 0	0 0 0 0 0	0 0 0 0 0	512	0
mfcr	31	D	0 0 0 0 0	0 0 0 0 0	19	0
mffs*x*[7]	63	D	0 0 0 0 0	0 0 0 0 0	583	Rc
mfmsr [1]	31	D	0 0 0 0 0	0 0 0 0 0	83	0
mfspr [2]	31	D	spr		339	0
mfsr [1]	31	D	0 SR	0 0 0 0 0	595	0
mfsrin [1]	31	D	0 0 0 0 0	B	659	0
mftb	31	D	tbr		371	0
mtcrf	31	S	0 CRM	0	144	0
mtfsb0*x*[7]	63	crbD	0 0 0 0 0	0 0 0 0 0	70	Rc
mtfsb1*x*[7]	63	crbD	0 0 0 0 0	0 0 0 0 0	38	Rc
mtfsf*x*[7]	63	0 FM	0	B	711	Rc
mtfsfi*x*[7]	63	crfD 0 0	0 0 0 0 0	IMM 0	134	Rc
mtmsr [1]	31	S	0 0 0 0 0	0 0 0 0 0	146	0
mtspr [2]	31	S	spr		467	0
mtsr [1]	31	S	0 SR	0 0 0 0 0	210	0
mtsrin [1]	31	S	0 0 0 0 0	B	242	0
mulhd*x*[4]	31	D	A	B	0 73	Rc
mulhdu*x*[4]	31	D	A	B	0 9	Rc
mulhw*x*	31	D	A	B	0 75	Rc

Name	0	6	11	16	21	22	26	27	30	31
mulhwu*x*	31	D	A	B	0	11				Rc
mulld*x*[4]	31	D	A	B	OE	233				Rc
mulli	7	D	A	SIMM						
mullw*x*	31	D	A	B	OE	235				Rc
nand*x*	31	S	A	B	476					Rc
neg*x*	31	D	A	0 0 0 0 0	OE	104				Rc
nor*x*	31	S	A	B	124					Rc
or*x*	31	S	A	B	444					Rc
orc*x*	31	S	A	B	412					Rc
ori	24	S	A	UIMM						
oris	25	S	A	UIMM						
rfi[1]	19	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	50					0
rldcl*x*[4]	30	S	A	B	mb			8		Rc
rldcr*x*[4]	30	S	A	B	me			9		Rc
rldic*x*[4]	30	S	A	sh	mb			2	sh	Rc
rldicl*x*[4]	30	S	A	sh	mb			0	sh	Rc
rldicr*x*[4]	30	S	A	sh	me			1	sh	Rc
rldimi*x*[4]	30	S	A	sh	mb			3	sh	Rc
rlwimi*x*	20	S	A	SH	MB		ME			Rc
rlwinm*x*	21	S	A	SH	MB		ME			Rc
rlwnm*x*	23	S	A	B	MB		ME			Rc
sc	17	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0					1	0
slbia[1,4,5]	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	498					0
slbie[1,4,5]	31	0 0 0 0 0	0 0 0 0 0	B	434					0
sld*x*[4]	31	S	A	B	27					Rc
slw*x*	31	S	A	B	24					Rc
srad*x*[4]	31	S	A	B	794					Rc
sradi*x*[4]	31	S	A	sh	413				sh	Rc
sraw*x*	31	S	A	B	792					Rc
srawi*x*	31	S	A	SH	824					Rc
srd*x*[4]	31	S	A	B	539					Rc
srw*x*	31	S	A	B	536					Rc
stb	38	S	A	d						

Name	0	6	11	16	21	22	30	31
stbu	39	S	A	d				
stbux	31	S	A	B	247			0
stbx	31	S	A	B	215			0
std[4]	62	S	A	ds			0	
stdcx.[4]	31	S	A	B	214			1
stdu[4]	62	S	A	ds			1	
stdux[4]	31	S	A	B	181			0
stdx[4]	31	S	A	B	149			0
stfd	54	S	A	d				
stfdu	55	S	A	d				
stfdux	31	S	A	B	759			0
stfdx	31	S	A	B	727			0
stfiwx[5]	31	S	A	B	983			0
stfs	52	S	A	d				
stfsu	53	S	A	d				
stfsux	31	S	A	B	695			0
stfsx	31	S	A	B	663			0
sth	44	S	A	d				
sthbrx	31	S	A	B	918			0
sthu	45	S	A	d				
sthux	31	S	A	B	439			0
sthx	31	S	A	B	407			0
stmw[3]	47	S	A	d				
stswi[3]	31	S	A	NB	725			0
stswx[3]	31	S	A	B	661			0
stw	36	S	A	d				
stwbrx	31	S	A	B	662			0
stwcx.	31	S	A	B	150			1
stwu	37	S	A	d				
stwux	31	S	A	B	183			0
stwx	31	S	A	B	151			0
subf*x*	31	D	A	B	OE	40		Rc
subfc*x*	31	D	A	B	OE	8		Rc

Name	0	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21	22 23 24 25 26 27 28 29 30	31
subfe*x*	31	D	A	B	OE	136	Rc
subfic	08	D	A	SIMM			
subfme*x*	31	D	A	0 0 0 0 0	OE	232	Rc
subfze*x*	31	D	A	0 0 0 0 0	OE	200	Rc
sync	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	598		0
td [4]	31	TO	A	B	68		0
tdi [4]	02	TO	A	SIMM			
tlbia [1,5]	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	370		0
tlbie [1,5]	31	0 0 0 0 0	0 0 0 0 0	B	306		0
tlbld [1,6]	31	0 0 0 0 0	0 0 0 0 0	B	978		0
tlbli [1,6]	31	0 0 0 0 0	0 0 0 0 0	B	1010		0
tlbsync[1,5]	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	566		0
tw	31	TO	A	B	4		0
twi	03	TO	A	SIMM			
xor*x*	31	S	A	B	316		Rc
xori	26	S	A	UIMM			
xoris	27	S	A	UIMM			

[1] Supervisor-level instruction
[2] Supervisor- and user-level instruction
[3] Load and store string or multiple instruction
[4] 64-bit instruction
[5] Optional in the PowerPC architecture
[6] Implementation-specific instruction
[7] Floating-point instructions are not supported by the EC603e microprocessor and are trapped by the floating-point unavailable exception vector.

A.2 Instructions Sorted by Opcode

Table A-2 lists the instructions defined in the PowerPC architecture in numeric order by opcode.

Key:

□	Reserved bits
▨	Instruction not implemented in the 603e

Table A-2. Complete Instruction List Sorted by Opcode

Name	0 5	6 7 8	9	10	11 12 13	14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29	30	31
tdi [4]	0 0 0 0 1 0	TO			A		SIMM				
twi	0 0 0 0 1 1	TO			A		SIMM				
mulli	0 0 0 1 1 1	D			A		SIMM				
subfic	0 0 1 0 0 0	D			A		SIMM				
cmpli	0 0 1 0 1 0	crfD	0	L	A		UIMM				
cmpi	0 0 1 0 1 1	crfD	0	L	A		SIMM				
addic	0 0 1 1 0 0	D			A		SIMM				
addic.	0 0 1 1 0 1	D			A		SIMM				
addi	0 0 1 1 1 0	D			A		SIMM				
addis	0 0 1 1 1 1	D			A		SIMM				
bc*x*	0 1 0 0 0 0	BO			BI		BD			AA	LK
sc	0 1 0 0 0 1	0 0 0 0 0			0 0 0 0 0		0 0 0 0 0 0 0 0 0 0 0 0 0 0			1	0
b*x*	0 1 0 0 1 0	LI								AA	LK
mcrf	0 1 0 0 1 1	crfD	0 0		crfS	0 0	0 0 0 0 0	0 0 0 0 0 0 0 0 0 0			0
bclr*x*	0 1 0 0 1 1	BO			BI		0 0 0 0 0	0 0 0 0 0 1 0 0 0 0			LK
crnor	0 1 0 0 1 1	crbD			crbA		crbB	0 0 0 0 1 0 0 0 0 1			0
rfi	0 1 0 0 1 1	0 0 0 0 0			0 0 0 0 0		0 0 0 0 0	0 0 0 0 1 1 0 0 1 0			0
crandc	0 1 0 0 1 1	crbD			crbA		crbB	0 0 1 0 0 0 0 0 0 1			0
isync	0 1 0 0 1 1	0 0 0 0 0			0 0 0 0 0		0 0 0 0 0	0 0 1 0 0 1 0 1 1 0			0
crxor	0 1 0 0 1 1	crbD			crbA		crbB	0 0 1 1 0 0 0 0 0 1			0
crnand	0 1 0 0 1 1	crbD			crbA		crbB	0 0 1 1 1 0 0 0 0 1			0
crand	0 1 0 0 1 1	crbD			crbA		crbB	0 1 0 0 0 0 0 0 0 1			0
creqv	0 1 0 0 1 1	crbD			crbA		crbB	0 1 0 0 1 0 0 0 0 1			0
crorc	0 1 0 0 1 1	crbD			crbA		crbB	0 1 1 0 1 0 0 0 0 1			0
cror	0 1 0 0 1 1	crbD			crbA		crbB	0 1 1 1 0 0 0 0 0 1			0
bcctr*x*	0 1 0 0 1 1	BO			BI		0 0 0 0 0	1 0 0 0 0 1 0 0 0 0			LK
rlwimi*x*	0 1 0 1 0 0	S			A		SH	MB	ME		Rc

Name	0–5	6–10	11–15	16–20	21–30	31
rlwinm*x*	0 1 0 1 0 1	S	A	SH	MB ME	Rc
rlwnm*x*	0 1 0 1 1 1	S	A	B	MB ME	Rc
ori	0 1 1 0 0 0	S	A	UIMM		
oris	0 1 1 0 0 1	S	A	UIMM		
xori	0 1 1 0 1 0	S	A	UIMM		
xoris	0 1 1 0 1 1	S	A	UIMM		
andi.	0 1 1 1 0 0	S	A	UIMM		
andis.	0 1 1 1 0 1	S	A	UIMM		
rldicl*x* [4]	0 1 1 1 1 0	S	A	sh	mb 0 0 0 sh	Rc
rldicr*x* [4]	0 1 1 1 1 0	S	A	sh	me 0 0 1 sh	Rc
rldic*x* [4]	0 1 1 1 1 0	S	A	sh	mb 0 1 0 sh	Rc
rldimi*x* [4]	0 1 1 1 1 0	S	A	sh	mb 0 1 1 sh	Rc
rldcl*x* [4]	0 1 1 1 1 0	S	A	B	mb 0 1 0 0 0	Rc
rldcr*x* [4]	0 1 1 1 1 0	S	A	B	me 0 1 0 0 1	Rc
cmp	0 1 1 1 1 1	crfD 0 L	A	B	0 0 0 0 0 0 0 0 0 0	0
tw	0 1 1 1 1 1	TO	A	B	0 0 0 0 0 0 0 1 0 0	0
subfc*x*	0 1 1 1 1 1	D	A	B	OE 0 0 0 0 0 0 1 0 0 0	Rc
mulhdu*x* [4]	0 1 1 1 1 1	D	A	B	0 0 0 0 0 0 0 1 0 0 1	Rc
addc*x*	0 1 1 1 1 1	D	A	B	OE 0 0 0 0 0 0 1 0 1 0	Rc
mulhwu*x*	0 1 1 1 1 1	D	A	B	0 0 0 0 0 0 0 1 0 1 1	Rc
mfcr	0 1 1 1 1 1	D	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0 1 0 0 1 1	0
lwarx	0 1 1 1 1 1	D	A	B	0 0 0 0 0 1 0 1 0 0	0
ldx [4]	0 1 1 1 1 1	D	A	B	0 0 0 0 0 1 0 1 0 1	0
lwzx	0 1 1 1 1 1	D	A	B	0 0 0 0 0 1 0 1 1 1	0
slw*x*	0 1 1 1 1 1	S	A	B	0 0 0 0 0 1 1 0 0 0	Rc
cntlzw*x*	0 1 1 1 1 1	S	A	0 0 0 0 0	0 0 0 0 0 1 1 0 1 0	Rc
sld*x* [4]	0 1 1 1 1 1	S	A	B	0 0 0 0 0 1 1 0 1 1	Rc
and*x*	0 1 1 1 1 1	S	A	B	0 0 0 0 0 1 1 1 0 0	Rc
cmpl	0 1 1 1 1 1	crfD 0 L	A	B	0 0 0 0 1 0 0 0 0 0	0
subf*x*	0 1 1 1 1 1	D	A	B	OE 0 0 0 0 1 0 1 0 0 0	Rc
ldux [4]	0 1 1 1 1 1	D	A	B	0 0 0 0 1 1 0 1 0 1	0
dcbst	0 1 1 1 1 1	0 0 0 0 0	A	B	0 0 0 0 1 1 0 1 1 0	0
lwzux	0 1 1 1 1 1	D	A	B	0 0 0 0 1 1 0 1 1 1	0

Name	0 … 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21	22 23 24 25 26 27 28 29 30	31
cntlzd*x*[4]	0 1 1 1 1 1	S	A	0 0 0 0 0		0 0 0 0 1 1 1 0 1 0	Rc
andc*x*	0 1 1 1 1 1	S	A	B		0 0 0 0 1 1 1 1 0 0	Rc
td[4]	0 1 1 1 1 1	TO	A	B		0 0 0 1 0 0 0 1 0 0	0
mulhd*x*[4]	0 1 1 1 1 1	D	A	B	0	0 0 0 1 0 0 1 0 0 1	Rc
mulhw*x*	0 1 1 1 1 1	D	A	B	0	0 0 0 1 0 0 1 0 1 1	Rc
mfmsr	0 1 1 1 1 1	D	0 0 0 0 0	0 0 0 0 0		0 0 0 1 0 1 0 0 1 1	0
ldarx[4]	0 1 1 1 1 1	D	A	B		0 0 0 1 0 1 0 1 0 0	0
dcbf	0 1 1 1 1 1	0 0 0 0 0	A	B		0 0 0 1 0 1 0 1 1 0	0
lbzx	0 1 1 1 1 1	D	A	B		0 0 0 1 0 1 0 1 1 1	0
neg*x*	0 1 1 1 1 1	D	A	0 0 0 0 0	OE	0 0 0 1 1 0 1 0 0 0	Rc
lbzux	0 1 1 1 1 1	D	A	B		0 0 0 1 1 1 0 1 1 1	0
nor*x*	0 1 1 1 1 1	S	A	B		0 0 0 1 1 1 1 1 0 0	Rc
subfe*x*	0 1 1 1 1 1	D	A	B	OE	0 0 1 0 0 0 1 0 0 0	Rc
adde*x*	0 1 1 1 1 1	D	A	B	OE	0 0 1 0 0 0 1 0 1 0	Rc
mtcrf	0 1 1 1 1 1	S	0 / CRM	CRM / 0		0 0 1 0 0 1 0 0 0 0	0
mtmsr	0 1 1 1 1 1	S	0 0 0 0 0	0 0 0 0 0		0 0 1 0 0 1 0 0 1 0	0
stdx[4]	0 1 1 1 1 1	S	A	B		0 0 1 0 0 1 0 1 0 1	0
stwcx.	0 1 1 1 1 1	S	A	B		0 0 1 0 0 1 0 1 1 0	1
stwx	0 1 1 1 1 1	S	A	B		0 0 1 0 0 1 0 1 1 1	0
stdux[4]	0 1 1 1 1 1	S	A	B		0 0 1 0 1 1 0 1 0 1	0
stwux	0 1 1 1 1 1	S	A	B		0 0 1 0 1 1 0 1 1 1	0
subfze*x*	0 1 1 1 1 1	D	A	0 0 0 0 0	OE	0 0 1 1 0 0 1 0 0 0	Rc
addze*x*	0 1 1 1 1 1	D	A	0 0 0 0 0	OE	0 0 1 1 0 0 1 0 1 0	Rc
mtsr	0 1 1 1 1 1	S	0 / SR	0 0 0 0 0		0 0 1 1 0 1 0 0 1 0	0
stdcx.[4]	0 1 1 1 1 1	S	A	B		0 0 1 1 0 1 0 1 1 0	1
stbx	0 1 1 1 1 1	S	A	B		0 0 1 1 0 1 0 1 1 1	0
subfme*x*	0 1 1 1 1 1	D	A	0 0 0 0 0	OE	0 0 1 1 1 0 1 0 0 0	Rc
mulld[4]	0 1 1 1 1 1	D	A	B	OE	0 0 1 1 1 0 1 0 0 1	Rc
addme*x*	0 1 1 1 1 1	D	A	0 0 0 0 0	OE	0 0 1 1 1 0 1 0 1 0	Rc
mullw*x*	0 1 1 1 1 1	D	A	B	OE	0 0 1 1 1 0 1 0 1 1	Rc
mtsrin	0 1 1 1 1 1	S	0 0 0 0 0	B		0 0 1 1 1 1 0 0 1 0	0
dcbtst	0 1 1 1 1 1	0 0 0 0 0	A	B		0 0 1 1 1 1 0 1 1 0	0
stbux	0 1 1 1 1 1	S	A	B		0 0 1 1 1 1 0 1 1 1	0

Name	0 … 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
add*x*	0 1 1 1 1 1	D	A	B	OE 0 1 0 0 0 0 1 0 1 0	Rc
dcbt	0 1 1 1 1 1	0 0 0 0 0	A	B	0 1 0 0 0 1 0 1 1 0	0
lhzx	0 1 1 1 1 1	D	A	B	0 1 0 0 0 1 0 1 1 1	0
eqv*x*	0 1 1 1 1 1	S	A	B	0 1 0 0 0 1 1 1 0 0	Rc
tlbie [1,5]	0 1 1 1 1 1	0 0 0 0 0	0 0 0 0 0	B	0 1 0 0 1 1 0 0 1 0	0
eciwx	0 1 1 1 1 1	D	A	B	0 1 0 0 1 1 0 1 1 0	0
lhzux	0 1 1 1 1 1	D	A	B	0 1 0 0 1 1 0 1 1 1	0
xor*x*	0 1 1 1 1 1	S	A	B	0 1 0 0 1 1 1 1 0 0	Rc
mfspr [2]	0 1 1 1 1 1	D	spr		0 1 0 1 0 1 0 0 1 1	0
lwax [4]	0 1 1 1 1 1	D	A	B	0 1 0 1 0 1 0 1 0 1	0
lhax	0 1 1 1 1 1	D	A	B	0 1 0 1 0 1 0 1 1 1	0
tlbia [1,5]	0 1 1 1 1 1	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 1 0 1 1 1 0 0 1 0	0
mftb	0 1 1 1 1 1	D	tbr		0 1 0 1 1 1 0 0 1 1	0
lwaux [4]	0 1 1 1 1 1	D	A	B	0 1 0 1 1 1 0 1 0 1	0
lhaux	0 1 1 1 1 1	D	A	B	0 1 0 1 1 1 0 1 1 1	0
sthx	0 1 1 1 1 1	S	A	B	0 1 1 0 0 1 0 1 1 1	0
orc*x*	0 1 1 1 1 1	S	A	B	0 1 1 0 0 1 1 1 0 0	Rc
sradi*x* [4]	0 1 1 1 1 1	S	A	sh	1 1 0 0 1 1 1 0 1 1 sh	Rc
slbie [1,4,5]	0 1 1 1 1 1	0 0 0 0 0	0 0 0 0 0	B	0 1 1 0 1 1 0 0 1 0	0
ecowx	0 1 1 1 1 1	S	A	B	0 1 1 0 1 1 0 1 1 0	0
sthux	0 1 1 1 1 1	S	A	B	0 1 1 0 1 1 0 1 1 1	0
or*x*	0 1 1 1 1 1	S	A	B	0 1 1 0 1 1 1 1 0 0	Rc
divdu*x* [4]	0 1 1 1 1 1	D	A	B	OE 0 1 1 1 0 0 1 0 0 1	Rc
divwu*x*	0 1 1 1 1 1	D	A	B	OE 0 1 1 1 0 0 1 0 1 1	Rc
mtspr [2]	0 1 1 1 1 1	S	spr		0 1 1 1 0 1 0 0 1 1	0
dcbi	0 1 1 1 1 1	0 0 0 0 0	A	B	0 1 1 1 0 1 0 1 1 0	0
nand*x*	0 1 1 1 1 1	S	A	B	0 1 1 1 0 1 1 1 0 0	Rc
divd*x* [4]	0 1 1 1 1 1	D	A	B	OE 0 1 1 1 1 0 1 0 0 1	Rc
divw*x*	0 1 1 1 1 1	D	A	B	OE 0 1 1 1 1 0 1 0 1 1	Rc
slbia [1,4,5]	0 1 1 1 1 1	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 1 1 1 1 1 0 0 1 0	0
mcrxr	0 1 1 1 1 1	crfD 0 0	0 0 0 0 0	0 0 0 0 0	1 0 0 0 0 0 0 0 0 0	0
lswx [3]	0 1 1 1 1 1	D	A	B	1 0 0 0 0 1 0 1 0 1	0
lwbrx	0 1 1 1 1 1	D	A	B	1 0 0 0 0 1 0 1 1 0	0

Name	0 … 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
lfsx[7]	0 1 1 1 1 1	D	A	B	1 0 0 0 0 1 0 1 1 1	0
srw*x*	0 1 1 1 1 1	S	A	B	1 0 0 0 0 1 1 0 0 0	Rc
srd*x* [4]	0 1 1 1 1 1	S	A	B	1 0 0 0 0 1 1 0 1 1	Rc
tlbsync [1,5]	0 1 1 1 1 1	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	1 0 0 0 1 1 0 1 1 0	0
lfsux[7]	0 1 1 1 1 1	D	A	B	1 0 0 0 1 1 0 1 1 1	0
mfsr	0 1 1 1 1 1	D	0 | SR	0 0 0 0 0	1 0 0 1 0 1 0 0 1 1	0
lswi [3]	0 1 1 1 1 1	D	A	NB	1 0 0 1 0 1 0 1 0 1	0
sync	0 1 1 1 1 1	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	1 0 0 1 0 1 0 1 1 0	0
lfdx[7]	0 1 1 1 1 1	D	A	B	1 0 0 1 0 1 0 1 1 1	0
lfdux[7]	0 1 1 1 1 1	D	A	B	1 0 0 1 1 1 0 1 1 1	0
mfsrin [1]	0 1 1 1 1 1	D	0 0 0 0 0	B	1 0 1 0 0 1 0 0 1 1	0
stswx [3]	0 1 1 1 1 1	S	A	B	1 0 1 0 0 1 0 1 0 1	0
stwbrx	0 1 1 1 1 1	S	A	B	1 0 1 0 0 1 0 1 1 0	0
stfsx	0 1 1 1 1 1	S	A	B	1 0 1 0 0 1 0 1 1 1	0
stfsux	0 1 1 1 1 1	S	A	B	1 0 1 0 1 1 0 1 1 1	0
stswi [3]	0 1 1 1 1 1	S	A	NB	1 0 1 1 0 1 0 1 0 1	0
stfdx[7]	0 1 1 1 1 1	S	A	B	1 0 1 1 0 1 0 1 1 1	0
stfdux[7]	0 1 1 1 1 1	S	A	B	1 0 1 1 1 1 0 1 1 1	0
lhbrx	0 1 1 1 1 1	D	A	B	1 1 0 0 0 1 0 1 1 0	0
sraw*x*	0 1 1 1 1 1	S	A	B	1 1 0 0 0 1 1 0 0 0	Rc
srad*x* [4]	0 1 1 1 1 1	S	A	B	1 1 0 0 0 1 1 0 1 0	Rc
srawi*x*	0 1 1 1 1 1	S	A	SH	1 1 0 0 1 1 1 0 0 0	Rc
eieio	0 1 1 1 1 1	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	1 1 0 1 0 1 0 1 1 0	0
sthbrx	0 1 1 1 1 1	S	A	B	1 1 1 0 0 1 0 1 1 0	0
extsh*x*	0 1 1 1 1 1	S	A	0 0 0 0 0	1 1 1 0 0 1 1 0 1 0	Rc
extsb*x*	0 1 1 1 1 1	S	A	0 0 0 0 0	1 1 1 0 1 1 1 0 1 0	Rc
tlbld [1,6]	0 1 1 1 1 1	0 0 0 0 0	0 0 0 0 0	B	1 1 1 1 0 1 0 0 1 0	0
icbi	0 1 1 1 1 1	0 0 0 0 0	A	B	1 1 1 1 0 1 0 1 1 0	0
stfiwx [5]	0 1 1 1 1 1	S	A	B	1 1 1 1 0 1 0 1 1 1	0
extsw [4]	0 1 1 1 1 1	S	A	0 0 0 0 0	1 1 1 1 0 1 1 0 1 0	Rc
tlbli [1,6]	0 1 1 1 1 1	0 0 0 0 0	0 0 0 0 0	B	1 1 1 1 1 1 0 0 1 0	0
dcbz	0 1 1 1 1 1	0 0 0 0 0	A	B	1 1 1 1 1 1 0 1 1 0	0
lwz	1 0 0 0 0 0	D	A	d		

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
lwzu	1 0 0 0 0 1	D	A	d			
lbz	1 0 0 0 1 0	D	A	d			
lbzu	1 0 0 0 1 1	D	A	d			
stw	1 0 0 1 0 0	S	A	d			
stwu	1 0 0 1 0 1	S	A	d			
stb	1 0 0 1 1 0	S	A	d			
stbu	1 0 0 1 1 1	S	A	d			
lhz	1 0 1 0 0 0	D	A	d			
lhzu	1 0 1 0 0 1	D	A	d			
lha	1 0 1 0 1 0	D	A	d			
lhau	1 0 1 0 1 1	D	A	d			
sth	1 0 1 1 0 0	S	A	d			
sthu	1 0 1 1 0 1	S	A	d			
lmw [3]	1 0 1 1 1 0	D	A	d			
stmw [3]	1 0 1 1 1 1	S	A	d			
lfs[7]	1 1 0 0 0 0	D	A	d			
lfsu[7]	1 1 0 0 0 1	D	A	d			
lfd[7]	1 1 0 0 1 0	D	A	d			
lfdu[7]	1 1 0 0 1 1	D	A	d			
stfs[7]	1 1 0 1 0 0	S	A	d			
stfsu[7]	1 1 0 1 0 1	S	A	d			
stfd[7]	1 1 0 1 1 0	S	A	d			
stfdu[7]	1 1 0 1 1 1	S	A	d			
ld [4]	1 1 1 0 1 0	D	A	ds			0 0
ldu [4]	1 1 1 0 1 0	D	A	ds			0 1
lwa [4]	1 1 1 0 1 0	D	A	ds			1 0
fdivs*x*[7]	1 1 1 0 1 1	D	A	B	0 0 0 0 0	1 0 0 1 0	Rc
fsubs*x*[7]	1 1 1 0 1 1	D	A	B	0 0 0 0 0	1 0 1 0 0	Rc
fadds*x*[7]	1 1 1 0 1 1	D	A	B	0 0 0 0 0	1 0 1 0 1	Rc
fsqrts*x* [5,7]	1 1 1 0 1 1	D	0 0 0 0 0	B	0 0 0 0 0	1 0 1 1 0	Rc
fres*x* [5,7]	1 1 1 0 1 1	D	0 0 0 0 0	B	0 0 0 0 0	1 1 0 0 0	Rc
fmuls*x*[7]	1 1 1 0 1 1	D	A	0 0 0 0 0	C	1 1 0 0 1	Rc
fmsubs*x*[7]	1 1 1 0 1 1	D	A	B	C	1 1 1 0 0	Rc

Name	0 … 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
fmadds*x*[7]	1 1 1 0 1 1	D	A	B	C	1 1 1 0 1	Rc
fnmsubs*x*[7]	1 1 1 0 1 1	D	A	B	C	1 1 1 1 0	Rc
fnmadds*x*[7]	1 1 1 0 1 1	D	A	B	C	1 1 1 1 1	Rc
std [4]	1 1 1 1 1 0	S	A	ds			0 0
stdu [4]	1 1 1 1 1 0	S	A	ds			0 1
fcmpu[7]	1 1 1 1 1 1	crfD 0 0	A	B	0 0 0 0 0 0 0 0 0 0		0
frsp*x*[7]	1 1 1 1 1 1	D	0 0 0 0 0	B	0 0 0 0 0 0 1 1 0 0		Rc
fctiw*x*[7]	1 1 1 1 1 1	D	0 0 0 0 0	B	0 0 0 0 0 0 1 1 1 0		
fctiwz*x*[7]	1 1 1 1 1 1	D	0 0 0 0 0	B	0 0 0 0 0 0 1 1 1 1		Rc
fdiv*x*[7]	1 1 1 1 1 1	D	A	B	0 0 0 0 0	1 0 0 1 0	Rc
fsub*x*[7]	1 1 1 1 1 1	D	A	B	0 0 0 0 0	1 0 1 0 0	Rc
fadd*x*[7]	1 1 1 1 1 1	D	A	B	0 0 0 0 0	1 0 1 0 1	Rc
fsqrt*x* [5,7]	1 1 1 1 1 1	D	0 0 0 0 0	B	0 0 0 0 0	1 0 1 1 0	Rc
fsel*x* [5,7]	1 1 1 1 1 1	D	A	B	C	1 0 1 1 1	Rc
fmul*x*[7]	1 1 1 1 1 1	D	A	0 0 0 0 0	C	1 1 0 0 1	Rc
frsqrte*x* [5,7]	1 1 1 1 1 1	D	0 0 0 0 0	B	0 0 0 0 0	1 1 0 1 0	Rc
fmsub*x*[7]	1 1 1 1 1 1	D	A	B	C	1 1 1 0 0	Rc
fmadd*x*[7]	1 1 1 1 1 1	D	A	B	C	1 1 1 0 1	Rc
fnmsub*x*[7]	1 1 1 1 1 1	D	A	B	C	1 1 1 1 0	Rc
fnmadd*x* [7]	1 1 1 1 1 1	D	A	B	C	1 1 1 1 1	Rc
fcmpo [7]	1 1 1 1 1 1	crfD 0 0	A	B	0 0 0 0 1 0 0 0 0 0		0
mtfsb1*x* [7]	1 1 1 1 1 1	crbD	0 0 0 0 0	0 0 0 0 0	0 0 0 0 1 0 0 1 1 0		Rc
fneg*x*[7]	1 1 1 1 1 1	D	0 0 0 0 0	B	0 0 0 0 1 0 1 0 0 0		Rc
mcrfs[7]	1 1 1 1 1 1	crfD 0 0	crfS 0 0	0 0 0 0 0	0 0 0 1 0 0 0 0 0 0		0
mtfsb0*x*[7]	1 1 1 1 1 1	crbD	0 0 0 0 0	0 0 0 0 0	0 0 0 1 0 0 0 1 1 0		Rc
fmr*x*[7]	1 1 1 1 1 1	D	0 0 0 0 0	B	0 0 0 1 0 0 1 0 0 0		Rc
mtfsfi*x*[7]	1 1 1 1 1 1	crfD 0 0	0 0 0 0 0	IMM 0	0 0 1 0 0 0 0 1 1 0		Rc
fnabs*x*[7]	1 1 1 1 1 1	D	0 0 0 0 0	B	0 0 1 0 0 0 1 0 0 0		Rc
fabs*x*[7]	1 1 1 1 1 1	D	0 0 0 0 0	B	0 1 0 0 0 0 1 0 0 0		Rc
mffs*x*[7]	1 1 1 1 1 1	D	0 0 0 0 0	0 0 0 0 0	1 0 0 1 0 0 0 1 1 1		Rc
mtfsf*x*[7]	1 1 1 1 1 1	0 FM	0	B	1 0 1 1 0 0 0 1 1 1		Rc
fctid*x* [4,7]	1 1 1 1 1 1	D	0 0 0 0 0	B	1 1 0 0 1 0 1 1 1 0		Rc

Name	0 … 5	6 … 10	11 … 15	16 … 20	21 … 30	31
fctidz*x* [4,7]	1 1 1 1 1 1	D	0 0 0 0 0	B	1 1 0 0 1 0 1 1 1 1	Rc
fcfid*x* [4,7]	1 1 1 1 1 1	D	0 0 0 0 0	B	1 1 0 1 0 0 1 1 1 0	Rc

[1] Supervisor-level instruction
[2] Supervisor- and user-level instruction
[3] Load and store string or multiple instruction
[4] 64-bit instruction
[5] Optional in the PowerPC architecture
[6] 603e-implementation specific instruction
[7] Floating-point instructions are not supported by the EC603e microprocessor and are trapped by the floating-point unavailable exception vector.

A.3 Instructions Grouped by Functional Categories

Table A-3 through Table A-30 list the PowerPC instructions grouped by function.

Key:

☐	Reserved bits
▨	Instruction not implemented in the 603e

Table A-3. Integer Arithmetic Instructions

Name	0 … 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21	22 23 24 25 26 27 28 29 30	31
add*x*	31	D	A	B	OE	266	Rc
addc*x*	31	D	A	B	OE	10	Rc
adde*x*	31	D	A	B	OE	138	Rc
addi	14	D	A	SIMM			
addic	12	D	A	SIMM			
addic.	13	D	A	SIMM			
addis	15	D	A	SIMM			
addme*x*	31	D	A	0 0 0 0 0	OE	234	Rc
addze*x*	31	D	A	0 0 0 0 0	OE	202	Rc
divd*x* [4]	31	D	A	B	OE	489	Rc
divdu*x* [4]	31	D	A	B	OE	457	Rc
divw*x*	31	D	A	B	OE	491	Rc
divwu*x*	31	D	A	B	OE	459	Rc
mulhd*x* [4]	31	D	A	B	0	73	Rc
mulhdu*x* [4]	31	D	A	B	0	9	Rc
mulhw*x*	31	D	A	B	0	75	Rc
mulhwu*x*	31	D	A	B	0	11	Rc
mulld [4]	31	D	A	B	OE	233	Rc
mulli	07	D	A	SIMM			
mullw*x*	31	D	A	B	OE	235	Rc
neg*x*	31	D	A	0 0 0 0 0	OE	104	Rc
subf*x*	31	D	A	B	OE	40	Rc
subfc*x*	31	D	A	B	OE	8	Rc
subfic*x*	08	D	A	SIMM			
subfe*x*	31	D	A	B	OE	136	Rc
subfme*x*	31	D	A	0 0 0 0 0	OE	232	Rc
subfze*x*	31	D	A	0 0 0 0 0	OE	200	Rc

Table A-4. Integer Compare Instructions

Name	0–5	6–8	9	10	11–15	16–20	21–30	31
cmp	31	crfD	0	L	A	B	0 0 0 0 0 0 0 0 0 0	0
cmpi	11	crfD	0	L	A	SIMM		
cmpl	31	crfD	0	L	A	B	32	0
cmpli	10	crfD	0	L	A	UIMM		

Table A-5. Integer Logical Instructions

Name	0–5	6–10	11–15	16–20	21–30	31
and*x*	31	S	A	B	28	Rc
andc*x*	31	S	A	B	60	Rc
andi.	28	S	A	UIMM		
andis.	29	S	A	UIMM		
cntlzd*x*[4]	31	S	A	0 0 0 0 0	58	Rc
cntlzw*x*	31	S	A	0 0 0 0 0	26	Rc
eqv*x*	31	S	A	B	284	Rc
extsb*x*	31	S	A	0 0 0 0 0	954	Rc
extsh*x*	31	S	A	0 0 0 0 0	922	Rc
extsw*x*[4]	31	S	A	0 0 0 0 0	986	Rc
nand*x*	31	S	A	B	476	Rc
nor*x*	31	S	A	B	124	Rc
or*x*	31	S	A	B	444	Rc
orc*x*	31	S	A	B	412	Rc
ori	24	S	A	UIMM		
oris	25	S	A	UIMM		
xor*x*	31	S	A	B	316	Rc
xori	26	S	A	UIMM		
xoris	27	S	A	UIMM		

Table A-6. Integer Rotate Instructions

Name	0–5	6–10	11–15	16–20	21–26	27–29	30	31
rldcl*x*[4]	30	S	A	B	mb	8		Rc
rldcr*x*[4]	30	S	A	B	me	9		Rc
rldic*x*[4]	30	S	A	sh	mb	2	sh	Rc
rldicl*x*[4]	30	S	A	sh	mb	0	sh	Rc
rldicr*x*[4]	30	S	A	sh	me	1	sh	Rc

Table A-6. Integer Rotate Instructions (Continued)

rldimi*x*[4]	30	S	A	sh	mb	3	sh	Rc
rlwimi*x*	22	S	A	SH	MB	ME		Rc
rlwinm*x*	20	S	A	SH	MB	ME		Rc
rlwnm*x*	21	S	A	SH	MB	ME		Rc

Table A-7. Integer Shift Instructions

Name	0–5	6–10	11–15	16–20	21–29	30	31
sld*x*[4]	31	S	A	B	27		Rc
slw*x*	31	S	A	B	24		Rc
srad*x*[4]	31	S	A	B	794		Rc
sradi*x*[4]	31	S	A	sh	413	sh	Rc
sraw*x*	31	S	A	B	792		Rc
srawi*x*	31	S	A	SH	824		Rc
srd*x*[4]	31	S	A	B	539		Rc
srw*x*	31	S	A	B	536		Rc

Table A-8. Floating-Point Arithmetic Instructions[7]

Name	0–5	6–10	11–15	16–20	21–25	26–30	31
fadd*x*	63	D	A	B	0 0 0 0 0	21	Rc
fadds*x*	59	D	A	B	0 0 0 0 0	21	Rc
fdiv*x*	63	D	A	B	0 0 0 0 0	18	Rc
fdivs*x*	59	D	A	B	0 0 0 0 0	18	Rc
fmul*x*	63	D	A	0 0 0 0 0	C	25	Rc
fmuls*x*	59	D	A	0 0 0 0 0	C	25	Rc
fres*x*[5]	59	D	0 0 0 0 0	B	0 0 0 0 0	24	Rc
frsqrte*x*[5]	63	D	0 0 0 0 0	B	0 0 0 0 0	26	Rc
fsub*x*	63	D	A	B	0 0 0 0 0	20	Rc
fsubs*x*	59	D	A	B	0 0 0 0 0	20	Rc
fsel*x*[5]	63	D	A	B	C	23	Rc
fsqrt*x*[5]	63	D	0 0 0 0 0	B	0 0 0 0 0	22	Rc
fsqrts*x*[5]	59	D	0 0 0 0 0	B	0 0 0 0 0	22	Rc

Table A-9. Floating-Point Multiply-Add Instructions[7]

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
fmadd*x*	63	D	A	B	C	29	Rc
fmadds*x*	59	D	A	B	C	29	Rc
fmsub*x*	63	D	A	B	C	28	Rc
fmsubs*x*	59	D	A	B	C	28	Rc
fnmadd*x*	63	D	A	B	C	31	Rc
fnmadds*x*	59	D	A	B	C	31	Rc
fnmsub*x*	63	D	A	B	C	30	Rc
fnmsubs*x*	59	D	A	B	C	30	Rc

Table A-10. Floating-Point Rounding and Conversion Instructions[7]

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
fcfid*x* [4]	63	D	0 0 0 0 0	B	846	Rc
fctid*x* [4]	63	D	0 0 0 0 0	B	814	Rc
fctidz*x* [4]	63	D	0 0 0 0 0	B	815	Rc
fctiw*x*	63	D	0 0 0 0 0	B	14	Rc
fctiwz*x*	63	D	0 0 0 0 0	B	15	Rc
frsp*x*	63	D	0 0 0 0 0	B	12	Rc

Table A-11. Floating-Point Compare Instructions[7]

Name	0 5	6 7 8	9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
fcmpo	63	crfD	0 0	A	B	32	0
fcmpu	63	crfD	0 0	A	B	0	0

Table A-12. Floating-Point Status and Control Register Instructions[7]

Name	0 5	6	7 8	9 10	11 12 13	14	15	16 17 18 19	20	21 22 23 24 25 26 27 28 29 30	31
mcrfs	63	crfD		0 0	crfS	0 0		0 0 0 0 0		64	0
mffs*x*	63	D			0 0 0 0 0			0 0 0 0 0		583	Rc
mtfsb0*x*	63	crbD			0 0 0 0 0			0 0 0 0 0		70	Rc
mtfsb1*x*	63	crbD			0 0 0 0 0			0 0 0 0 0		38	Rc
mtfsf*x*	31	0	FM				0	B		711	Rc
mtfsfi*x*	63	crfD		0 0	0 0 0 0 0			IMM	0	134	Rc

Table A-13. Integer Load Instructions

Name	0 … 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
lbz	34	D	A	d		
lbzu	35	D	A	d		
lbzux	31	D	A	B	119	0
lbzx	31	D	A	B	87	0
ld [4]	58	D	A	ds		0
ldu [4]	58	D	A	ds		1
ldux [4]	31	D	A	B	53	0
ldx [4]	31	D	A	B	21	0
lha	42	D	A	d		
lhau	43	D	A	d		
lhaux	31	D	A	B	375	0
lhax	31	D	A	B	343	0
lhz	40	D	A	d		
lhzu	41	D	A	d		
lhzux	31	D	A	B	311	0
lhzx	31	D	A	B	279	0
lwa [4]	58	D	A	ds		2
lwaux [4]	31	D	A	B	373	0
lwax [4]	31	D	A	B	341	0
lwz	32	D	A	d		
lwzu	33	D	A	d		
lwzux	31	D	A	B	55	0
lwzx	31	D	A	B	23	0

Table A-14. Integer Store Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
stb	38	S	A	d		
stbu	39	S	A	d		
stbux	31	S	A	B	247	0
stbx	31	S	A	B	215	0
std[4]	62	S	A	ds		0
stdu[4]	62	S	A	ds		1
stdux[4]	31	S	A	B	181	0
stdx[4]	31	S	A	B	149	0
sth	44	S	A	d		
sthu	45	S	A	d		
sthux	31	S	A	B	439	0
sthx	31	S	A	B	407	0
stw	36	S	A	d		
stwu	37	S	A	d		
stwux	31	S	A	B	183	0
stwx	31	S	A	B	151	0

Table A-15. Integer Load and Store with Byte-Reverse Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
lhbrx	31	D	A	B	790	0
lwbrx	31	D	A	B	534	0
sthbrx	31	S	A	B	918	0
stwbrx	31	S	A	B	662	0

Table A-16. Integer Load and Store Multiple Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
lmw[3]	46	D	A	d		
stmw[3]	47	S	A	d		

Table A-17. Integer Load and Store String Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
lswi [3]	31	D	A	NB	597	0
lswx [3]	31	D	A	B	533	0
stswi [3]	31	S	A	NB	725	0
stswx [3]	31	S	A	B	661	0

Table A-18. Memory Synchronization Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
eieio	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	854	0
isync	19	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	150	0
ldarx [4]	31	D	A	B	84	0
lwarx	31	D	A	B	20	0
stdcx.[4]	31	S	A	B	214	1
stwcx.	31	S	A	B	150	1
sync	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	598	0

Table A-19. Floating-Point Load Instructions[7]

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
lfd	50	D	A	d		
lfdu	51	D	A	d		
lfdux	31	D	A	B	631	0
lfdx	31	D	A	B	599	0
lfs	48	D	A	d		
lfsu	49	D	A	d		
lfsux	31	D	A	B	567	0
lfsx	31	D	A	B	535	0

Table A-20. Floating-Point Store Instructions[7]

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
stfd	54	S	A	d		
stfdu	55	S	A	d		
stfdux	31	S	A	B	759	0
stfdx	31	S	A	B	727	0
stfiwx [5]	31	S	A	B	983	0
stfs	52	S	A	d		
stfsu	53	S	A	d		
stfsux	31	S	A	B	695	0
stfsx	31	S	A	B	663	0

Table A-21. Floating-Point Move Instructions[7]

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
fabs*x*	63	D	0 0 0 0 0	B	264	Rc
fmr*x*	63	D	0 0 0 0 0	B	72	Rc
fnabs*x*	63	D	0 0 0 0 0	B	136	Rc
fneg*x*	63	D	0 0 0 0 0	B	40	Rc

Table A-22. Branch Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29	30	31
b*x*	18	LI				AA	LK
bc*x*	16	BO	BI	BD		AA	LK
bcctr*x*	19	BO	BI	0 0 0 0 0	528		LK
bclr*x*	19	BO	BI	0 0 0 0 0	16		LK

Table A-23. Condition Register Logical Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
crand	19	crbD	crbA	crbB	257	0
crandc	19	crbD	crbA	crbB	129	0
creqv	19	crbD	crbA	crbB	289	0
crnand	19	crbD	crbA	crbB	225	0
crnor	19	crbD	crbA	crbB	33	0

Table A-23. Condition Register Logical Instructions (Continued)

cror	19	crbD		crbA		crbB	449	0
crorc	19	crbD		crbA		crbB	417	0
crxor	19	crbD		crbA		crbB	193	0
mcrf	19	crfD	0 0	crfS	0 0	0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	0

Table A-24. System Linkage Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29	30	31
rfi [1]	19	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	50		0
sc	17	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0		1	0

Table A-25. Trap Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
td [4]	31	TO	A	B	68	0
tdi [4]	03	TO	A	SIMM		
tw	31	TO	A	B	4	0
twi	03	TO	A	SIMM		

Table A-26. Processor Control Instructions

Name	0 5	6 7 8	9 10	11	12 13 14 15	16 17 18 19	20	21 22 23 24 25 26 27 28 29 30	31
mcrxr	31	crfS	0 0	0 0 0 0 0		0 0 0 0 0		512	0
mfcr	31	D		0 0 0 0 0		0 0 0 0 0		19	0
mfmsr [1]	31	D		0 0 0 0 0		0 0 0 0 0		83	0
mfspr [2]	31	D		spr				339	0
mftb	31	D		tpr				371	0
mtcrf	31	S		0	CRM		0	144	0
mtmsr [1]	31	S		0 0 0 0 0		0 0 0 0 0		146	0
mtspr [2]	31	D		spr				467	0

Table A-27. Cache Management Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
dcbf	31	0 0 0 0 0	A	B	86	0
dcbi [1]	31	0 0 0 0 0	A	B	470	0
dcbst	31	0 0 0 0 0	A	B	54	0
dcbt	31	0 0 0 0 0	A	B	278	0
dcbtst	31	0 0 0 0 0	A	B	246	0
dcbz	31	0 0 0 0 0	A	B	1014	0
icbi	31	0 0 0 0 0	A	B	982	0

Table A-28. Segment Register Manipulation Instructions

Name	0 5	6 7 8 9 10	11	12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
mfsr [1]	31	D	0	SR	0 0 0 0 0	595	0
mfsrin [1]	31	D	0 0 0 0 0		B	659	0
mtsr [1]	31	S	0	SR	0 0 0 0 0	210	0
mtsrin [1]	31	S	0 0 0 0 0		B	242	0

Table A-29. Lookaside Buffer Management Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
slbia [1,4,5]	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	498	0
slbie [1,4,5]	31	0 0 0 0 0	0 0 0 0 0	B	434	0
tlbia [1,5]	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	370	0
tlbie [1,5]	31	0 0 0 0 0	0 0 0 0 0	B	306	0
tlbld [1,6]	31	0 0 0 0 0	0 0 0 0 0	B	978	0
tlbli [1,6]	31	0 0 0 0 0	0 0 0 0 0	B	1010	0
tlbsync [1,5]	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	566	0

Table A-30. External Control Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
eciwx	31	D	A	B	310	0
ecowx	31	S	A	B	438	0

[1] Supervisor-level instruction
[2] Supervisor- and user-level instruction
[3] Load and store string or multiple instruction
[4] 64-bit instruction
[5] Optional in the PowerPC architecture
[6] 603e-implementation specific instruction
[7] Floating-point instructions are not supported by the EC603e microprocessor and are trapped by the floating-point unavailable exception vector.

A.4 Instructions Sorted by Form

Table A-31 through Table A-45 list the PowerPC instructions grouped by form.

Key:

□	Reserved bits
▨	Instruction not implemented in the 603e

Table A-31. I-Form

OPCD	LI	AA	LK

Specific Instruction

Name	0 5	6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29	30	31
b*x*	18	LI	AA	LK

Table A-32. B-Form

OPCD	BO	BI	BD	AA	LK

Specific Instruction

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29	30	31
bc*x*	16	BO	BI	BD	AA	LK

Table A-33. SC-Form

OPCD	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	0

Specific Instruction

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29	30	31
sc	17	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	0

Table A-34. D-Form

OPCD	D			A	d
OPCD	D			A	SIMM
OPCD	S			A	d
OPCD	S			A	UIMM
OPCD	crfD	0	L	A	SIMM
OPCD	crfD	0	L	A	UIMM
OPCD	TO			A	SIMM

Specific Instructions

Name	0 – 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
addi	14	D	A	SIMM
addic	12	D	A	SIMM
addic.	13	D	A	SIMM
addis	15	D	A	SIMM
andi.	28	S	A	UIMM
andis.	29	S	A	UIMM
cmpi	11	crfD | 0 | L	A	SIMM
cmpli	10	crfD | 0 | L	A	UIMM
lbz	34	D	A	d
lbzu	35	D	A	d
lfd[7]	50	D	A	d
lfdu [7]	51	D	A	d
lfs[7]	48	D	A	d
lfsu[7]	49	D	A	d
lha	42	D	A	d
lhau	43	D	A	d
lhz	40	D	A	d
lhzu	41	D	A	d
lmw [3]	46	D	A	d
lwz	32	D	A	d
lwzu	33	D	A	d
mulli	7	D	A	SIMM
ori	24	S	A	UIMM
oris	25	S	A	UIMM
stb	38	S	A	d
stbu	39	S	A	d
stfd[7]	54	S	A	d
stfdu[7]	55	S	A	d
stfs[7]	52	S	A	d
stfsu[7]	53	S	A	d
sth	44	S	A	d
sthu	45	S	A	d

stmw [3]	47	S	A	d
stw	36	S	A	d
stwu	37	S	A	d
subfic	08	D	A	SIMM
tdi [4]	02	TO	A	SIMM
twi	03	TO	A	SIMM
xori	26	S	A	UIMM
xoris	27	S	A	UIMM

Table A-35. DS-Form

OPCD	D	A	ds	XO
OPCD	S	A	ds	XO

Specific Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29	30 31
ld [4]	58	D	A	ds	0
ldu [4]	58	D	A	ds	1
lwa [4]	58	D	A	ds	2
std [4]	62	S	A	ds	0
stdu [4]	62	S	A	ds	1

Table A-36. X-Form

OPCD	D	A	B	XO	0
OPCD	D	A	NB	XO	0
OPCD	D	0 0 0 0 0	B	XO	0
OPCD	D	0 0 0 0 0	0 0 0 0 0	XO	0
OPCD	D	0 SR	0 0 0 0 0	XO	0
OPCD	S	A	B	XO	Rc
OPCD	S	A	B	XO	1
OPCD	S	A	B	XO	0
OPCD	S	A	NB	XO	0
OPCD	S	A	0 0 0 0 0	XO	Rc
OPCD	S	0 0 0 0 0	B	XO	0
OPCD	S	0 0 0 0 0	0 0 0 0 0	XO	0
OPCD	S	0 SR	0 0 0 0 0	XO	0
OPCD	S	A	SH	XO	Rc
OPCD	crfD 0 L	A	B	XO	0

OPCD	crfD	0 0		A		B		XO	0
OPCD	crfD	0 0		crfS	0 0	0 0 0 0 0		XO	0
OPCD	crfD	0 0		0 0 0 0 0		0 0 0 0 0		XO	0
OPCD	crfD	0 0		0 0 0 0 0		IMM	0	XO	Rc
OPCD	TO			A		B		XO	0
OPCD	D			0 0 0 0 0		B		XO	Rc
OPCD	D			0 0 0 0 0		0 0 0 0 0		XO	Rc
OPCD	crbD			0 0 0 0 0		0 0 0 0 0		XO	Rc
OPCD	0 0 0 0 0			A		B		XO	0
OPCD	0 0 0 0 0			0 0 0 0 0		B		XO	0
OPCD	0 0 0 0 0			0 0 0 0 0		0 0 0 0 0		XO	0

Specific Instructions

and*x*	31	S			A	B	28	Rc
andc*x*	31	S			A	B	60	Rc
cmp	31	crfD	0	L	A	B	0	0
cmpl	31	crfD	0	L	A	B	32	0
cntlzd*x* [4]	31	S			A	0 0 0 0 0	58	Rc
cntlzw*x*	31	S			A	0 0 0 0 0	26	Rc
dcbf	31	0 0 0 0 0			A	B	86	0
dcbi [1]	31	0 0 0 0 0			A	B	470	0
dcbst	31	0 0 0 0 0			A	B	54	0
dcbt	31	0 0 0 0 0			A	B	278	0
dcbtst	31	0 0 0 0 0			A	B	246	0
dcbz	31	0 0 0 0 0			A	B	1014	0
eciwx	31	D			A	B	310	0
ecowx	31	S			A	B	438	0
eieio	31	0 0 0 0 0			0 0 0 0 0	0 0 0 0 0	854	0
eqv*x*	31	S			A	B	284	Rc
extsb*x*	31	S			A	0 0 0 0 0	954	Rc
extsh*x*	31	S			A	0 0 0 0 0	922	Rc
extsw*x* [4]	31	S			A	0 0 0 0 0	986	Rc
fabs*x*[7]	63	D			0 0 0 0 0	B	264	Rc
fcfid*x* [4,7]	63	D			0 0 0 0 0	B	846	Rc
fcmpo [7]	63	crfD	0 0		A	B	32	0

fcmpu[7]	63	crfD	0 0	A		B	0	0
fctid*x*[4,7]	63	D		0 0 0 0 0		B	814	Rc
fctidz*x*[4,7]	63	D		0 0 0 0 0		B	815	Rc
fctiw*x*[7]	63	D		0 0 0 0 0		B	14	Rc
fctiwz*x*[7]	63	D		0 0 0 0 0		B	15	Rc
fmr*x*[7]	63	D		0 0 0 0 0		B	72	Rc
fnabs*x*[7]	63	D		0 0 0 0 0		B	136	Rc
fneg*x*[7]	63	D		0 0 0 0 0		B	40	Rc
frsp*x*[7]	63	D		0 0 0 0 0		B	12	Rc
icbi	31	0 0 0 0 0		A		B	982	0
lbzux	31	D		A		B	119	0
lbzx	31	D		A		B	87	0
ldarx[4]	31	D		A		B	84	0
ldux[4]	31	D		A		B	53	0
ldx[4]	31	D		A		B	21	0
lfdux[7]	31	D		A		B	631	0
lfdx[7]	31	D		A		B	599	0
lfsux[7]	31	D		A		B	567	0
lfsx[7]	31	D		A		B	535	0
lhaux	31	D		A		B	375	0
lhax	31	D		A		B	343	0
lhbrx	31	D		A		B	790	0
lhzux	31	D		A		B	311	0
lhzx	31	D		A		B	279	0
lswi[3]	31	D		A		NB	597	0
lswx[3]	31	D		A		B	533	0
lwarx	31	D		A		B	20	0
lwaux[4]	31	D		A		B	373	0
lwax[4]	31	D		A		B	341	0
lwbrx	31	D		A		B	534	0
lwzux	31	D		A		B	55	0
lwzx	31	D		A		B	23	0
mcrfs	63	crfD	0 0	crfS	0 0	0 0 0 0 0	64	0
mcrxr	31	crfD	0 0	0 0 0 0 0		0 0 0 0 0	512	0
mfcr	31	D		0 0 0 0 0		0 0 0 0 0	19	0

mffs*x*[7]	63	D	0 0 0 0 0	0 0 0 0 0	583	Rc
mfmsr[1]	31	D	0 0 0 0 0	0 0 0 0 0	83	0
mfsr[1]	31	D	0 SR	0 0 0 0 0	595	0
mfsrin[1]	31	D	0 0 0 0 0	B	659	0
mtfsb0*x*[7]	63	crbD	0 0 0 0 0	0 0 0 0 0	70	Rc
mtfsb1*x*[7]	63	crfD	0 0 0 0 0	0 0 0 0 0	38	Rc
mtfsfi*x*[7]	63	crbD 0 0	0 0 0 0 0	IMM 0	134	Rc
mtmsr[1]	31	S	0 0 0 0 0	0 0 0 0 0	146	0
mtsr[1]	31	S	0 SR	0 0 0 0 0	210	0
mtsrin[1]	31	S	0 0 0 0 0	B	242	0
nand*x*	31	S	A	B	476	Rc
nor*x*	31	S	A	B	124	Rc
or*x*	31	S	A	B	444	Rc
orc*x*	31	S	A	B	412	Rc
slbia[1,4,5]	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	498	0
slbie[1,4,5]	31	0 0 0 0 0	0 0 0 0 0	B	434	0
sld*x*[4]	31	S	A	B	27	Rc
slw*x*	31	S	A	B	24	Rc
srad*x*[4]	31	S	A	B	794	Rc
sraw*x*	31	S	A	B	792	Rc
srawi*x*	31	S	A	SH	824	Rc
srd*x*[4]	31	S	A	B	539	Rc
srw*x*	31	S	A	B	536	Rc
stbux	31	S	A	B	247	0
stbx	31	S	A	B	215	0
stdcx.[4]	31	S	A	B	214	1
stdux[4]	31	S	A	B	181	0
stdx[4]	31	S	A	B	149	0
stfdux[7]	31	S	A	B	759	0
stfdx[7]	31	S	A	B	727	0
stfiwx[5,7]	31	S	A	B	983	0
stfsux[7]	31	S	A	B	695	0
stfsx[7]	31	S	A	B	663	0
sthbrx	31	S	A	B	918	0
sthux	31	S	A	B	439	0

sthx	31	S	A	B	407	0
stswi [3]	31	S	A	NB	725	0
stswx [3]	31	S	A	B	661	0
stwbrx	31	S	A	B	662	0
stwcx.	31	S	A	B	150	1
stwux	31	S	A	B	183	0
stwx	31	S	A	B	151	0
sync	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	598	0
td [4]	31	TO	A	B	68	0
tlbia [1,5]	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	370	0
tlbie [1,5]	31	0 0 0 0 0	0 0 0 0 0	B	306	0
tlbld [1,6]	31	0 0 0 0 0	0 0 0 0 0	B	978	0
tlbli [1,6]	31	0 0 0 0 0	0 0 0 0 0	B	1010	0
tlbsync[1,5]	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	566	0
tw	31	TO	A	B	4	0
xor*x*	31	S	A	B	316	Rc

Table A-37. XL-Form

OPCD	BO		BI		0 0 0 0 0	XO	LK
OPCD	crbD		crbA		crbB	XO	0
OPCD	crfD	0 0	crfS	0 0	0 0 0 0 0	XO	0
OPCD	0 0 0 0 0		0 0 0 0 0		0 0 0 0 0	XO	0

Specific Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
bcctr*x*	19	BO	BI	0 0 0 0 0	528	LK
bclr*x*	19	BO	BI	0 0 0 0 0	16	LK
crand	19	crbD	crbA	crbB	257	0
crandc	19	crbD	crbA	crbB	129	0
creqv	19	crbD	crbA	crbB	289	0
crnand	19	crbD	crbA	crbB	225	0
crnor	19	crbD	crbA	crbB	33	0
cror	19	crbD	crbA	crbB	449	0
crorc	19	crbD	crbA	crbB	417	0
crxor	19	crbD	crbA	crbB	193	0

Name	0–5	6–8	9–10	11–13	14–15	16–20	21–30	31
isync	19	0 0 0 0 0		0 0 0 0 0		0 0 0 0 0	150	0
mcrf	19	crfD	0 0	crfS	0 0	0 0 0 0 0	0	0
rfi [1]	19	0 0 0 0 0		0 0 0 0 0		0 0 0 0 0	50	0

Table A-38. XFX-Form

OPCD	D	spr			XO	0
OPCD	D	0	CRM	0	XO	0
OPCD	S	spr			XO	0
OPCD	D	tbr			XO	0

Specific Instructions

Name	0–5	6–10	11	12–19	20	21–30	31
mfspr [2]	31	D	spr			339	0
mftb	31	D	tbr			371	0
mtcrf	31	S	0	CRM	0	144	0
mtspr [2]	31	D	spr			467	0

Table A-39. XFL-Form

OPCD	0	FM	0	B	XO	Rc

Specific Instructions

Name	0–5	6	7–14	15	16–20	21–30	31
mtfsf*x*[7]	63	0	FM	0	B	711	Rc

Table A-40. XS-Form

OPCD	S	A	sh	XO	sh	Rc

Specific Instructions

Name	0–5	6–10	11–15	16–20	21–29	30	31
sradi*x* [4]	31	S	A	sh	413	sh	Rc

Table A-41. XO-Form

OPCD	D	A	B	OE	XO	Rc
OPCD	D	A	B	0	XO	Rc
OPCD	D	A	0 0 0 0 0	OE	XO	Rc

Specific Instructions

Name	0–5	6–10	11–15	16–20	21	22–30	31
add*x*	31	D	A	B	OE	266	Rc
addc*x*	31	D	A	B	OE	10	Rc

adde*x*	31	D	A	B	OE	138	Rc
addme*x*	31	D	A	0 0 0 0 0	OE	234	Rc
addze*x*	31	D	A	0 0 0 0 0	OE	202	Rc
divd*x* [4]	31	D	A	B	OE	489	Rc
divdu*x* [4]	31	D	A	B	OE	457	Rc
divw*x*	31	D	A	B	OE	491	Rc
divwu*x*	31	D	A	B	OE	459	Rc
mulhd*x* [4]	31	D	A	B	0	73	Rc
mulhdu*x* [4]	31	D	A	B	0	9	Rc
mulhw*x*	31	D	A	B	0	75	Rc
mulhwu*x*	31	D	A	B	0	11	Rc
mulld*x* [4]	31	D	A	B	OE	233	Rc
mullw*x*	31	D	A	B	OE	235	Rc
neg*x*	31	D	A	0 0 0 0 0	OE	104	Rc
subf*x*	31	D	A	B	OE	40	Rc
subfc*x*	31	D	A	B	OE	8	Rc
subfe*x*	31	D	A	B	OE	136	Rc
subfme*x*	31	D	A	0 0 0 0 0	OE	232	Rc
subfze*x*	31	D	A	0 0 0 0 0	OE	200	Rc

Table A-42. A-Form

OPCD	D	A	B	0 0 0 0 0	XO	Rc
OPCD	D	A	B	C	XO	Rc
OPCD	D	A	0 0 0 0 0	C	XO	Rc
OPCD	D	0 0 0 0 0	B	0 0 0 0 0	XO	Rc

Specific Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
fadd*x*[7]	63	D	A	B	0 0 0 0 0	21	Rc
fadds*x*[7]	59	D	A	B	0 0 0 0 0	21	Rc
fdiv*x*[7]	63	D	A	B	0 0 0 0 0	18	Rc
fdivs*x*[7]	59	D	A	B	0 0 0 0 0	18	Rc
fmadd*x*[7]	63	D	A	B	C	29	Rc
fmadds*x*[7]	59	D	A	B	C	29	Rc
fmsub*x*[7]	63	D	A	B	C	28	Rc
fmsubs*x*[7]	59	D	A	B	C	28	Rc

fmul*x*[7]	63	D	A	0 0 0 0 0	C	25	Rc
fmuls*x*[7]	59	D	A	0 0 0 0 0	C	25	Rc
fnmadd*x*[7]	63	D	A	B	C	31	Rc
fnmadds*x*[7]	59	D	A	B	C	31	Rc
fnmsub*x*[7]	63	D	A	B	C	30	Rc
fnmsubs*x*[7]	59	D	A	B	C	30	Rc
fres*x*[5'7]	59	D	0 0 0 0 0	B	0 0 0 0 0	24	Rc
frsqrte*x*[5,7]	63	D	0 0 0 0 0	B	0 0 0 0 0	26	Rc
fsel*x*[5.7]	63	D	A	B	C	23	Rc
fsqrt*x*[5,7]	63	D	0 0 0 0 0	B	0 0 0 0 0	22	Rc
fsqrts*x*[5,7]	59	D	0 0 0 0 0	B	0 0 0 0 0	22	Rc
fsub*x*[7]	63	D	A	B	0 0 0 0 0	20	Rc
fsubs*x*[7]	59	D	A	B	0 0 0 0 0	20	Rc

Table A-43. M-Form

OPCD	S	A	SH	MB	ME	Rc
OPCD	S	A	B	MB	ME	Rc

Specific Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
rlwimi*x*	20	S	A	SH	MB	ME	Rc
rlwinm*x*	21	S	A	SH	MB	ME	Rc
rlwnm*x*	23	S	A	B	MB	ME	Rc

Table A-44. MD-Form

OPCD	S	A	sh	mb	XO	sh	Rc
OPCD	S	A	sh	me	XO	sh	Rc

Specific Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26	27 28 29	30	31
rldic*x*[4]	30	S	A	sh	mb	2	sh	Rc
rldicl*x*[4]	30	S	A	sh	mb	0	sh	Rc
rldicr*x*[4]	30	S	A	sh	me	1	sh	Rc
rldimi*x*[4]	30	S	A	sh	mb	3	sh	Rc

Table A-45. MDS-Form

OPCD	S	A	B	mb	XO	Rc
OPCD	S	A	B	me	XO	Rc

Specific Instructions

Name	0 ... 5	6 ... 10	11 ... 15	16 ... 20	21 ... 26	27 ... 30	31
rldcl*x* [4]	30	S	A	B	mb	8	Rc
rldcr*x* [4]	30	S	A	B	me	9	Rc

[1] Supervisor-level instruction
[2] Supervisor- and user-level instruction
[3] Load and store string or multiple instruction
[4] 64-bit instruction
[5] Optional in the PowerPC architecture
[6] 603e-implementation specific instruction
[7] Floating-point instructions are not supported by the EC603e microprocessor and are trapped by the floating-point unavailable exception vector.

A.5 Instruction Set Legend

Table A-46 provides general information on the PowerPC instruction set (such as the architectural level, privilege level, and form).

Key:

☐	Reserved bits
▨	Instruction not implemented in the 603e

Table A-46. PowerPC Instruction Set Legend

	UISA	VEA	OEA	Supervisor Level	64-Bit	Optional	Form
add*x*	√						XO
addc*x*	√						XO
adde*x*	√						XO
addi	√						D
addic	√						D
addic.	√						D
addis	√						D
addme*x*	√						XO
addze*x*	√						XO
and*x*	√						X
andc*x*	√						X
andi.	√						D
andis.	√						D
b*x*	√						I
bc*x*	√						B
bcctr*x*	√						XL
bclr*x*	√						XL
cmp	√						X
cmpi	√						D
cmpl	√						X
cmpli	√						D
cntlzd*x* [4]	√				√		X
cntlzw*x*	√						X
crand	√						XL
crandc	√						XL
creqv	√						XL

crnand	√						XL
crnor	√						XL
cror	√						XL
crorc	√						XL
crxor	√						XL
dcbf		√					X
dcbi [1]			√	√			X
dcbst		√					X
dcbt		√					X
dcbtst		√					X
dcbz		√					X
divd*x* [4]	√				√		XO
divdu*x* [4]	√				√		XO
divw*x*	√						XO
divwu*x*	√						XO
eciwx		√				√	X
ecowx		√				√	X
eieio		√					X
eqv*x*	√						X
extsb*x*	√						X
extsh*x*	√						X
extsw*x* [4]	√				√		X
fabs*x*[7]	√						X
fadd*x*[7]	√						A
fadds*x*[7]	√						A
fcfid*x* [4,7]	√				√		X
fcmpo[7]	√						X
fcmpu[7]	√						X
fctid*x* [4,7]	√				√		X
fctidz*x* [7,4]	√				√		X
fctiw*x*[7]	√						X
fctiwz*x*[7]	√						X
fdiv*x*[7]	√						A
fdivs*x*[7]	√						A
fmadd*x*[7]	√						A

fmadds*x*[7]	√						A
fmr*x*[7]	√						X
fmsub*x*[7]	√						A
fmsubs*x*[7]	√						A
fmul*x*[7]	√						A
fmuls*x*[7]	√						A
fnabs*x*[7]	√						X
fneg*x*[7]	√						X
fnmadd*x*[7]	√						A
fnmadds*x*[7]	√						A
fnmsub*x*[7]	√						A
fnmsubs*x*[7]	√						A
fres*x*[5,7]	√					√	A
frsp*x*[7]	√						X
frsqrte*x*[5,7]	√					√	A
fsel*x*[5,7]	√					√	A
fsqrt*x*[5,7]	√					√	A
fsqrts*x*[5,7]	√					√	A
fsub*x*[7]	√						A
fsubs*x*[7]	√						A
icbi		√					X
isync		√					XL
lbz	√						D
lbzu	√						D
lbzux	√						X
lbzx	√						X
ld[4]	√				√		DS
ldarx[4]	√				√		X
ldu[4]	√				√		DS
ldux[4]	√				√		X
ldx[4]	√				√		X
lfd[7]	√						D
lfdu[7]	√						D
lfdux[7]	√						X
lfdx[7]	√						X

	UISA	VEA	OEA	Supervisor Level	64-bit	Optional	Form
lfs[7]	√						D
lfsu[7]	√						D
lfsux[7]	√						X
lfsx[7]	√						X
lha	√						D
lhau	√						D
lhaux	√						X
lhax	√						X
lhbrx	√						X
lhz	√						D
lhzu	√						D
lhzux	√						X
lhzx	√						X
lmw [3]	√						D
lswi [3]	√						X
lswx [3]	√						X
lwa [4]	√				√		DS
lwarx	√						X
lwaux [4]	√				√		X
lwax [4]	√				√		X
lwbrx	√						X
lwz	√						D
lwzu	√						D
lwzux	√						X
lwzx	√						X
mcrf	√						XL
mcrfs[7]	√						X
mcrxr	√						X
mfcr	√						X
mffs*x*[7]	√						X
mfmsr [1]			√	√			X
mfspr [2]	√		√	√			XFX
mfsr [1]			√	√			X
mfsrin [1]			√	√			X

	UISA	VEA	OEA	Supervisor Level	64-bit	Optional	Form
mftb		√					XFX
mtcrf	√						XFX
mtfsb0*x*[7]	√						X
mtfsb1*x*[7]	√						X
mtfsf*x*[7]	√						XFL
mtfsfi*x*[7]	√						X
mtmsr[1]			√	√			X
mtspr[2]	√		√	√			XFX
mtsr[1]			√	√			X
mtsrin[1]			√	√			X
mulhd*x*[4]	√				√		XO
mulhdu*x*[4]	√				√		XO
mulhw*x*	√						XO
mulhwu*x*	√						XO
mulld*x*[4]	√				√		XO
mulli	√						D
mullw*x*	√						XO
nand*x*	√						X
neg*x*	√						XO
nor*x*	√						X
or*x*	√						X
orc*x*	√						X
ori	√						D
oris	√						D
rfi[1]			√	√			XL
rldcl*x*[4]	√				√		MDS
rldcr*x*[4]	√				√		MDS
rldic*x*[4]	√				√		MD
rldicl*x*[4]	√				√		MD
rldicr*x*[4]	√				√		MD
rldimi*x*[4]	√				√		MD
rlwimi*x*	√						M
rlwinm*x*	√						M
rlwnm*x*	√						M

	UISA	VEA	OEA	Supervisor Level	64-bit	Optional	Form
sc	√		√				SC
slbia [1,4,5]			√	√	√	√	X
slbie [1,4,5]			√	√	√	√	X
sld*x* [4]	√				√		X
slw*x*	√						X
srad*x* [4]	√				√		X
sradi*x* [4]	√				√		XS
sraw*x*	√						X
srawi*x*	√						X
srd*x* [4]	√				√		X
srw*x*	√						X
stb	√						D
stbu	√						D
stbux	√						X
stbx	√						X
std [4]	√				√		DS
stdcx. [4]	√				√		X
stdu [4]	√				√		DS
stdux [4]	√				√		X
stdx [4]	√				√		X
stfd[7]	√						D
stfdu[7]	√						D
stfdux[7]	√						X
stfdx[7]	√						X
stfiwx [5,7]	√					√	X
stfs[7]	√						D
stfsu[7]	√						D
stfsux[7]	√						X
stfsx[7]	√						X
sth	√						D
sthbrx	√						X
sthu	√						D
sthux	√						X
sthx	√						X

stmw [3]	√						D
stswi [3]	√						X
stswx [3]	√						X
stw	√						D
stwbrx	√						X
stwcx.	√						X
stwu	√						D
stwux	√						X
stwx	√						X
subf*x*	√						XO
subfc*x*	√						XO
subfe*x*	√						XO
subfic	√						D
subfme*x*	√						XO
subfze*x*	√						XO
sync	√						X
td [4]	√				√		X
tdi [4]	√				√		D
tlbia [1,5]			√	√		√	X
tlbie [1,5]			√	√		√	X
tlbld [1,6]				√			X
tlbli [1,6]				√			X
tlbsync [1,5]			√	√			X
tw	√						X
twi	√						D
xor*x*	√						X
xori	√						D
xoris	√						D

[1] Supervisor-level instruction
[2] Supervisor- and user-level instruction
[3] Load and store string or multiple instruction
[4] 64-bit instruction
[5] Optional in the PowerPC architecture
[6] 603e-implementation specific instruction
[7] Floating-point instructions are not supported by the EC603e microprocessor andare trapped by the floating-point unavailable exception vector.

Appendix B
Instructions Not Implemented

This appendix provides a list of the 32-bit and 64-bit PowerPC instructions that are not implemented in the PowerPC 603e microprocessor. It also provides a list of the floating-point instructions that are not supported by the EC603e microprocessor and the 64-bit SPR encoding that is not implemented by the 603e. Note that any attempt to execute instructions that are not implemented on the 603e will generate an illegal instruction exception. Note that exceptions are referred to as interrupts in the architecture specification.

Table B-1 provides the 32-bit PowerPC instructions that are optional to the PowerPC architecture but not implemented by the 603e.

Table B-1. 32-Bit Instructions Not Implemented by the PowerPC 603e

Mnemonic	Instruction
fsqrt	Floating Square Root (Double-Precision)
fsqrts	Floating Square Root Single
tlbia	TLB Invalidate All

Table B-2 provides a list of 64-bit instructions that are not implemented by the 603e and EC603e microprocessors.

Table B-2. 64-Bit Instructions Not Implemented

Mnemonic	Instruction
cntlzd	Count Leading Zeros Double Word
divd	Divide Double Word
divdu	Divide Double Word Unsigned
extsw	Extend Sign Word
fcfid	Floating Convert From Integer Double Word
fctid	Floating Convert to Integer Double Word
fctidz	Floating Convert to Integer Double Word with Round toward Zero
ld	Load Double Word
ldarx	Load Double Word and Reserve Indexed

Table B-2. 64-Bit Instructions Not Implemented (Continued)

Mnemonic	Instruction
ldu	Load Double Word with Update
ldux	Load Double Word with Update Indexed
ldx	Load Double Word Indexed
lwa	Load Word Algebraic
lwaux	Load Word Algebraic with Update Indexed
lwax	Load Word Algebraic Indexed
mulld	Multiply Low Double Word
mulhd	Multiply High Double Word
mulhdu	Multiply High Double Word Unsigned
rldcl	Rotate Left Double Word then Clear Left
rldcr	Rotate Left Double Word then Clear Right
rldic	Rotate Left Double Word Immediate then Clear
rldicl	Rotate Left Double Word Immediate then Clear Left
rldicr	Rotate Left Double Word Immediate then Clear Right
rldimi	Rotate Left Double Word Immediate then Mask Insert
slbia	SLB Invalidate All
slbie	SLB Invalidate Entry
sld	Shift Left Double Word
srad	Shift Right Algebraic Double Word
sradi	Shift Right Algebraic Double Word Immediate
srd	Shift Right Double Word
std	Store Double Word
stdcx.	Store Double Word Conditional Indexed
stdu	Store Double Word with Update
stdux	Store Double Word Indexed with Update
stdx	Store Double Word Indexed
td	Trap Double Word
tdi	Trap Double Word Immediate

Table B-3 lists floating-point instructions that are not supported by the EC603e microprocessor. The EC603e microprocessor does not support the floating-point unit; therefore, floating-point instructions are trapped by the floating-point unavailable exception vector but they may be emulated by software.

Table B-3. Floating-Point Instructions Not Supported by the EC603e Microprocessor

Mnemonic	Instruction
fabs	Floating Absolute
fadd	Floating Add
fadds	Floating Add Single
fcmpo	Floating Compare Ordered
fcmpu	Floating Compare Unordered
fctiw	Floating Convert to Integer Word
fctiwz	Floating Convert to Integer Word with Round toward Zero
fdiv	Floating Divide
fdivs	Floating Divide Single
fmadd	Floating Multiply Add
fmadds	Floating Multipy Add Single
fmr	Floating Move Register
fmsub	Floating Multiply Subtract
fmsubs	Floating Multiply Subtract Single
fmul	Floating Multiply
fmuls	Floating Multiply Single
fnabs	Floating Negative Absolute
fneg	Floating Negative
fnmadd	Floating Negative Multiply-Add (Double-Precision)
fnmadds	Floating Negatve Multiply-Add Single
fnmsub	Floating Negative Multiply-Subtract (Double-Precision)
fnmsubs	Floating Negative Multiply -Subtract Single
fres	Floating Reciprocal Estimate Single
frsp	Floating Round to Single
frsqrte	Floating Reciprocal Square Root Estimate
fsel	Floating Select
fsqrt	Floating Square Root (Double-Precision)
fsqrts	Floating Square Root Single

Table B-3. Floating-Point Instructions Not Supported by the EC603e Microprocessor (Continued)

Mnemonic	Instruction
fsub	Floating Subtract
fsubs	Floating Subtract Single
lfd	Load Floating-Point Double
lfdu	Load Floating-Point Double with Update
lfdux	Load Floating-Point Double with Update Indexed
lfdx	Load Floating-Point
lfs	Load Floating-Point Single
lfsu	Load Floating-Point Single with Update
lfsux	Load Floating-Point Single with Update Indexed
lfsx	Load Floating-Point Indexed
mcrfs	Move to Condition Register from FPSCR
mffs	Move from FPSCR
mtfsb0	Move to FPSCR Bit 0
mtfsb1	Move to FPSCR Bit 1
mtfsf	Move to FPSCR Fields
mtfsfi	Move to FPSCR Field Immediate
stfd	Store Floating-Point Double
stfdu	Store Floating-Point Double with Update
stfdux	Store Floating-Point Double with Update Indexed
stfdx	Store Floating-Point Double Indexed
stfiwx	Store Floating-Point as Integer Word Indexed
stfs	Store Floating-Point Single
stfsu	Store Floating-Point Single with Update
stfsux	Store Floating-Point Single with Update Indexed
stfsx	Store Floating-Point Single Indexed
tlbia	TLB Invalidate All

Table B-4 provides the 64-bit SPR encoding that is not implemented by the 603e and

EC603e microprocessor.

Table B-4. 64-Bit SPR Encoding Not Implemented

SPR			Register Name	Access
Decimal	spr[5–9]	spr[0–4]		
280	01000	11000	ASR	Supervisor

Appendix C PowerPC 603 Processor System Design and Programming Considerations

While the PowerPC 603 microprocessor shares most of the attributes of the PowerPC 603e microprocessor, the system designer or programmer should keep in mind the 603 hardware and software differences, described in the following sections, that can require modifications to accommodate the 603 in systems designed for the 603e.

C.1 PowerPC 603 Microprocessor Hardware Considerations

The 603's hardware implementation differs from the 603e in the following ways:

- $\overline{\text{XATS}}$ signal replaces CSE1 signal
- Hardware support for access to direct-store segments
- Bus clock multipliers of 1:1, 2:1, 3:1, and 4:1 only
- 8-Kbyte, two-way set associative instruction and data caches
- HID1 register not implemented in 603

The following sections provide further information on the operation of some of the hardware features specific to the 603.

C.1.1 Hardware Support for Direct-Store Accesses

The 603 provides hardware support for direct-store bus accesses through the provision of the extended address transfer start ($\overline{\text{XATS}}$) signal, and support for direct-store accesses in the bus interface unit. Direct-store accesses are invoked when a segment register T bit is set to 1.

The operation of the $\overline{\text{XATS}}$ signal is described in the following section. The $\overline{\text{XATS}}$ signal is in the same location as the CSE1 signal on the 603e.

C.1.1.1 Extended Address Transfer Start ($\overline{\text{XATS}}$)

The $\overline{\text{XATS}}$ signal is both an input and an output signal on the 603.

C.1.1.1.1 Extended Address Transfer Start ($\overline{\text{XATS}}$)—Output

Following are the state meaning and timing comments for the $\overline{\text{XATS}}$ output signal.

State Meaning	Asserted—Indicates that the 603 has begun a direct-store operation and that the first address cycle is valid. When asserted with the appropriate XATC signals it is also an implied data bus request for certain direct-store operation (unless it is an address-only operation). Negated—Is negated during an entire memory transaction.
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—Coincides with the negation of $\overline{\text{ABB}}$.

C.1.1.1.2 Extended Address Transfer Start ($\overline{\text{XATS}}$)—Input

Following are the state meaning and timing comments for the $\overline{\text{XATS}}$ input signal.

State Meaning	Asserted—Indicates that the 603 must check for a direct-store operation reply. Negated—Indicates that there is no need to check for a direct-store operation reply.
Timing Comments	Assertion—May occur while $\overline{\text{ABB}}$ is asserted. Negation—Must occur one bus clock cycle after $\overline{\text{XATS}}$ is asserted.

C.1.2 Direct-Store Protocol Operation

The 603 defines separate memory-mapped and I/O address spaces, or segments, distinguished by the corresponding segment register T bit in the address translation logic of the 603. If the T bit is cleared, the memory reference is a normal memory-mapped access and can use the virtual memory management hardware of the 603. If the T bit is set, the memory reference is a direct-store access.

The following points should be considered for direct-store accesses:

- The use of direct-store segment accesses may have a significant impact on the performance of the 603. The provision of direct-store segment access capability by the 603 is to provide compatibility with earlier hardware I/O controllers and may not be provided in future derivatives of the 603 family.
- Direct-store accesses are strongly ordered; for example, these accesses occur on the bus strictly in order with respect to the instruction stream.
- Direct-store accesses provide synchronous error reporting.

The 603 has a single bus interface to support both memory accesses and direct-store segment accesses.

The direct-store protocol for the 603 allows for the transfer of 1 to 128 bytes of data between the 603 and the bus unit controller (BUC) for each single load or store request issued by the program. The block of data is transferred by the 603 as multiple single-beat bus transactions (individual address and data tenure for each transaction) until completion. The program waits for the sequence of bus transactions to be completed so that a final completion status (error or no error) can be reported precisely with respect to the program flow. The completion status is snooped by the 603 from a bus transaction run by the BUC.

The system recognizes the assertion of the $\overline{\text{TS}}$ signal as the start of a memory-mapped access. The assertion of $\overline{\text{XATS}}$ indicates a direct-store access. This allows memory-mapped devices to ignore direct-store transactions. If $\overline{\text{XATS}}$ is asserted, the access is to a direct-store space and the following extensions to the memory access protocol apply:

- A new set of bus operations are defined. The transfer type, transfer burst, and transfer size signals are redefined for direct-store operations; they convey the opcode for the I/O transaction (see Table C-1).
- There are two beats of address for each direct-store transfer. The first beat (packet 0) provides basic address information such as the segment register and the sender tag and several control bits; the second beat (packet 1) provides additional addressing bits from the segment register and the logical address.
- The TT[0–3], $\overline{\text{TBST}}$, and TSIZ[0–2] signals are remapped to form an 8-bit extended transfer code (XATC) which specifies a command and transfer size for the transaction. The XATC field is driven and snooped by the 603 during direct-store transactions.
- Only the data signals such as DH[0–31] and DP[0–3] are used. The lower half of the data bus and parity is ignored.
- The sender that initiated the transaction must wait for a reply from the receiver bus unit controller (BUC) before starting a new operation.
- The 603 does not burst direct-store transactions. All direct-store transactions generated by the 603 are single-beat transactions of 4 bytes or less (single data beat tenure per address tenure).

Direct-store transactions use separate arbitration for the split address and data buses and define address-only and single-beat transactions. The address-retry vehicle is identical, although there is no hardware coherency support for direct-store transactions. The $\overline{\text{ARTRY}}$ signal is useful, however, for pacing 603 transactions, effectively indicating to the 603 that the BUC is in a queue-full condition and cannot accept new data.

In addition to the extensions noted above, there are fundamental differences between memory-mapped and direct-store operations. For example, only half of the 64-bit data path is available for 603 direct-store transactions. This lowers the pin count for I/O interfaces but generally results in substantially less bandwidth than memory-mapped accesses. Additionally, load/store instructions that address direct-store segments cannot complete successfully without an error-free reply from the addressed BUC. Because normal direct-store accesses involve multiple I/O transactions (streaming), they are likely to be very long latency instructions; therefore, direct-store operations usually stall 603 instruction issue.

Figure C-1 shows a direct-store tenure. Note that the I/O device response is an address-only bus transaction.

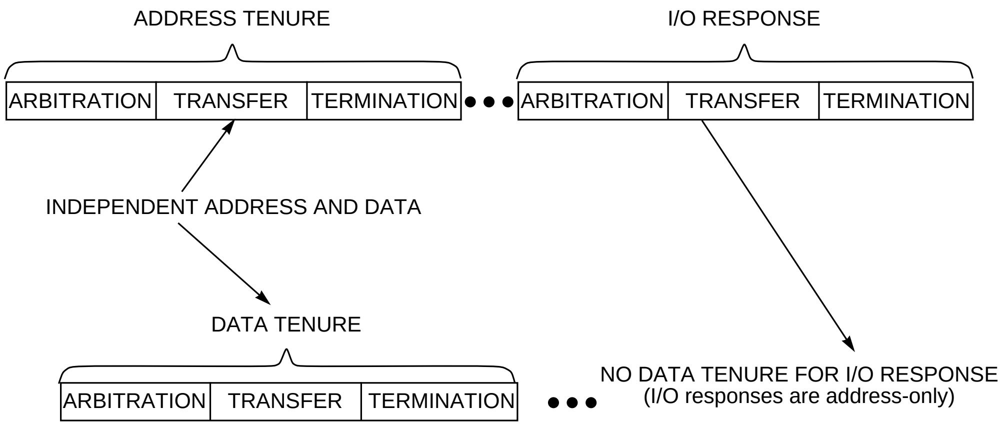


Figure C-1. Direct-Store Tenures

It should be noted that in the best case, the use of the 603 direct-store protocol degrades performance and requires the addressed controllers to implement 603 bus master capability to generate the reply transactions.

C.1.2.1 Direct-Store Transactions

The 603 defines seven direct-store transaction operations, as shown in Table C-1. These operations permit communication between the 603 and BUCs. A single 603 store or load instruction (that translates to a direct-store access) generates one or more direct-store operations (two or more direct-store operations for loads) from the 603 and one reply operation from the addressed BUC.

Table C-1. Direct-Store Bus Operations

Operation	Address Only	Direction	XATC Encoding
Load start (request)	Yes	603 ⇒ IO	0100 0000
Load immediate	No	603 ⇒ IO	0101 0000
Load last	No	603 ⇒ IO	0111 0000
Store immediate	No	603 ⇒ IO	0001 0000
Store last	No	603 ⇒ IO	0011 0000
Load reply	Yes	IO ⇒ 603	1100 0000
Store reply	Yes	IO ⇒ 603	1000 0000

For the first beat of the address bus, the extended address transfer code (XATC) contains the I/O opcode as shown in Table C-1; the opcode is formed by concatenating the transfer type, transfer burst, and transfer size signals defined as follows:

XATC = TT[0:3]||$\overline{\text{TBST}}$||TSIZ[0–2]

C.1.2.1.1 Store Operations

There are three operations defined for direct-store store operations from the 603 to the BUC, defined as follows:

1. Store immediate operations transfer up to 32 bits of data each from the 603 to the BUC.
2. Store last operations transfer up to 32 bits of data each from the 603 to the BUC.
3. Store reply from the BUC reveals the success/failure of that direct-store access to the 603.

A direct-store store access consists of one or more data transfer operations followed by the I/O store reply operation from the BUC. 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, as shown in the following sequence:

STORE LAST (from 603)
•
•
STORE REPLY (from BUC)

However, if more data is involved in the direct-store access, there will be one or more store immediate operations. The BUC can detect when the last data is being transferred by looking for the store last opcode, as shown in the following sequence:

STORE IMMEDIATE(s)
•
•
STORE LAST
•
•
STORE REPLY

C.1.2.1.2 Load Operations

Direct-store load accesses are similar to store operations, except that the 603 latches data from the addressed BUC rather than supplying the data to the BUC. As with memory accesses, the 603 is the master on both load and store operations; the external system must provide the data bus grant to the 603 when the BUC is ready to supply the data to the 603.

The load request direct-store operation has no analogous store operation; it informs the addressed BUC of the total number of bytes of data that the BUC must provide to the 603 on the subsequent load immediate/load last operations. For direct-store load accesses, the simplest, 32-bit (or fewer) data transfer sequence is as follows:

LOAD REQUEST

•

•

LOAD LAST

•

•

LOAD REPLY (from BUC)

However, if more data is involved in the direct-store access, there will be one or more load immediate operations. The BUC can detect when the last data is being transferred by looking for the load last opcode, as seen in the following sequence:

LOAD REQUEST

•

•

LOAD IMM(s)

•

•

LOAD LAST

•

•

LOAD REPLY

Note that three of the seven defined operations are address-only transactions and do not use the data bus. However, unlike the memory transfer protocol, these transactions are not broadcast from one master to all snooping devices. The direct-store address-only transaction protocol strictly controls communication between the 603 and the BUC.

C.1.2.2 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 C-2 as two packets. The first packet, packet 0, is then expanded to depict the XATC and address bus information in detail.

C.1.2.2.1 Packet 0

Figure C-2 shows the organization of the first packet in a direct-store transaction.

The XATC contains the I/O opcode, as discussed earlier and as shown in Table C-1. The address bus contains the following:

Key bit || segment register || sender tag

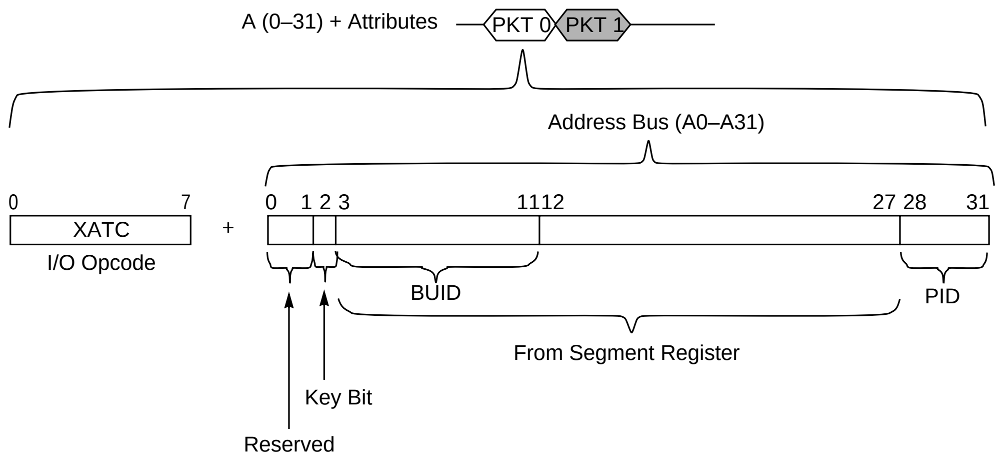


Figure C-2. Direct-Store Operation—Packet 0

This information is organized as follows:

- Bits 0 and 1 of the address bus are reserved—the 603 always drives these bits to zero.
- Key bit—Bit 2 is the key bit from the segment register (either SR[Kp] or SR[Ks]). Kp indicates user-level access and Ks indicates supervisor-level access. The 603 multiplexes the correct key bit into this position according to the current operating context (user or supervisor). (Note that user- and supervisor-level refer to problem and privileged state, respectively, in the architecture specification.)
- Segment register—Address bits 3–27 correspond to bits 3–27 of the selected segment register. Note that address bits 3–11 form the 9-bit receiver tag. Software must initialize these bits in the segment register to the ID of the BUC to be addressed; they are referred to as the BUID (bus unit ID) bits.
- PID (sender tag)—Address bits 28–31 form the 4-bit sender tag. The 603 PID (processor ID) comes from bits 28–31 of the 603's processor ID register. The 4-bit PID 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, for example, it is envisioned that the second-level cache (or equivalent logic) can append a larger processor tag as needed. The BUC addressed by the receiver tag should latch the sender address required by the subsequent I/O reply operation.

C.1.2.2.2 Packet 1

The second address beat, packet 1, transfers byte counts and the physical address for the transaction, as shown in Figure C-3.

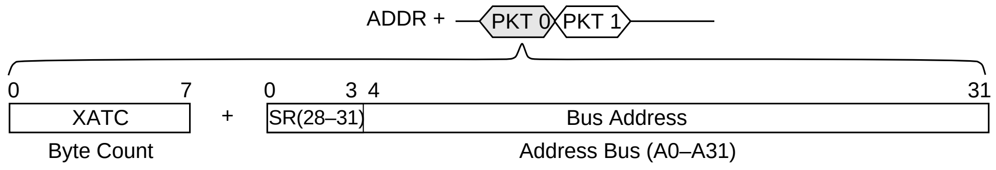


Figure C-3. Direct-Store Operation—Packet 1

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 603).
- Immediate/last (load or store) operations—XATC contains the current transfer byte count (1 to 4 bytes).

Address bits 0–31 contain the physical address of the transaction. The physical address is generated by concatenating segment register bits 28–31 with bits 4–31 of the effective address, as follows:

Segment register (bits 28–31) || effective address (bits 4–31)

While the 603 provides the address of the transaction to the BUC, the BUC must maintain a valid address pointer for the reply.

C.1.2.3 I/O Reply Operations

BUCs must respond to 603 direct-store transactions with an I/O reply operation, as shown in Figure C-4. The purpose of this reply operation is to inform the 603 of the success or failure of the attempted direct-store access. This requires the system direct-store slave to have 603 bus mastership capability—a substantially more complex design task than bus slave implementations that use memory-mapped I/O access.

Reply operations from the BUC to the 603 are address-only transactions. As with packet 0 of the address bus on 603 direct-store operations, the XATC contains the opcode for the operation (see Table C-1). Additionally, the I/O reply operation transfers the sender/receiver tags in the first beat.

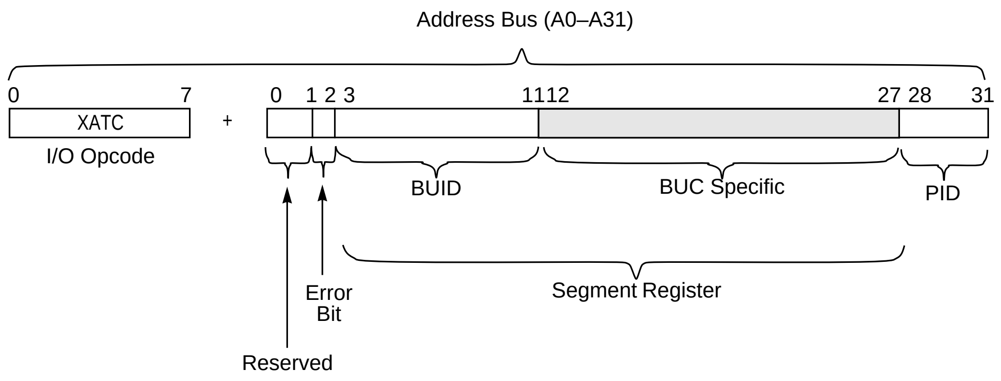


Figure C-4. I/O Reply Operation

The address bits are described in Table C-2.

Table C-2. Address Bits for I/O Reply Operations

Address 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 603 segment registers.
12–27	Address bits 12–27 are BUC-specific and are ignored by the 603.
28–31	PID (receiver tag). The 603 effectively snoops operations on the bus and, on reply operations, compares this field to bits 28–31 of the PID register 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 the 603 detects an error bit set on an I/O reply operation:

1. The 603 completes the instruction that initiated the access.
2. If the instruction is a load, the data is forwarded to the register file(s)/sequencer.
3. A direct-store error exception is generated, which transfers 603 control to the direct-store error exception handler to recover from the error.

If the error bit is not set, the 603 instruction that initiated the access completes and instruction execution resumes.

System designers should note the following:

- “Misplaced” reply operations (that match the processor tag and arrive unexpectedly) are ignored by the 603.
- External logic must assert $\overline{\text{AACK}}$ for the 603, even though it is the receiver of the reply operation. $\overline{\text{AACK}}$ is an input-only signal to the 603.
- The 603 monitors address parity when enabled by software and $\overline{\text{XATS}}$ and reply operations (load or store).

C.1.2.4 Direct-Store Operation Timing

The following timing diagrams show the sequence of events in a typical 603 direct-store load access (Figure C-5) and a typical 603 direct-store store access (Figure C-6). All arbitration signals except for $\overline{\text{ABB}}$ and $\overline{\text{DBB}}$ have been omitted for clarity, although they are still required. Note that, for either case, the number of immediate operations depends on the amount and the alignment of data to be transferred. If no more than 4 bytes are being transferred, and the data is double-word-aligned (that is, does not straddle an 8-byte address boundary), there will be no immediate operation as shown in the figures.

The 603 can transfer as many as 128 bytes of data in one load or store instruction (requiring more than 33 immediate operations in the case of misaligned operands).

In Figure C-5, $\overline{\text{XATS}}$ is asserted with the same timing relationship as $\overline{\text{TS}}$ in a memory access. Notice, however, that the address bus (and XATC) transition on the next bus clock cycle. The first of the two beats on the address bus is valid for one bus clock cycle window only, and that window is defined by the assertion of $\overline{\text{XATS}}$. The second address bus beat, however, can be extended by delaying the assertion of $\overline{\text{AACK}}$ until the system has latched the address.

The load request and load reply operations, shown in Figure C-5, are address-only transactions as denoted by the negated TT3 signal during their respective address tenures. Note that other types of bus operations can occur between the individual direct-store operations on the bus. The 603 involved in this transaction, however, does not initiate any other direct-store load or store operations once the first direct-store operation has begun address tenure; however, if the I/O operation is retried, other higher-priority operations can occur.

Notice that, in this example (zero wait states), 13 bus clock cycles are required to transfer no more than 8 bytes of data.

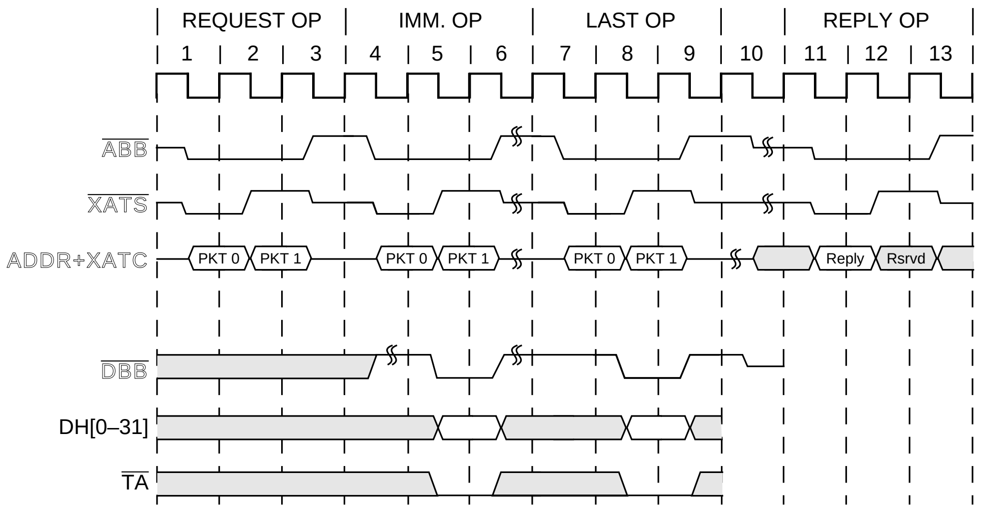


Figure C-5. Direct-Store Interface Load Access Example

Figure C-6 shows a direct-store store access comprised of three direct-store operations. As with the example in Figure C-5, notice that data is transferred only on the 32 bits of the DH bus. As opposed to Figure C-5, there is no request operation since the 603 has the data ready for the BUC.

The assertion of the $\overline{\text{TEA}}$ signal during a direct-store operation indicates that an unrecoverable error has occurred. If the $\overline{\text{TEA}}$ signal is asserted during a direct-store operation, the $\overline{\text{TEA}}$ action will be delayed and the following direct-store transactions will continue until all data transfers from the direct store segment had been completed. The bus agent that asserts $\overline{\text{TEA}}$ is responsible for asserting the $\overline{\text{TEA}}$ signal for every direct-store transaction tenure including the last one. The direct-store reply, in this case, is not required and will be ignored by the processor. The processor will take a machine check exception after the last direct-store data tenure has been terminated by the assertion of $\overline{\text{TEA}}$, and not before.

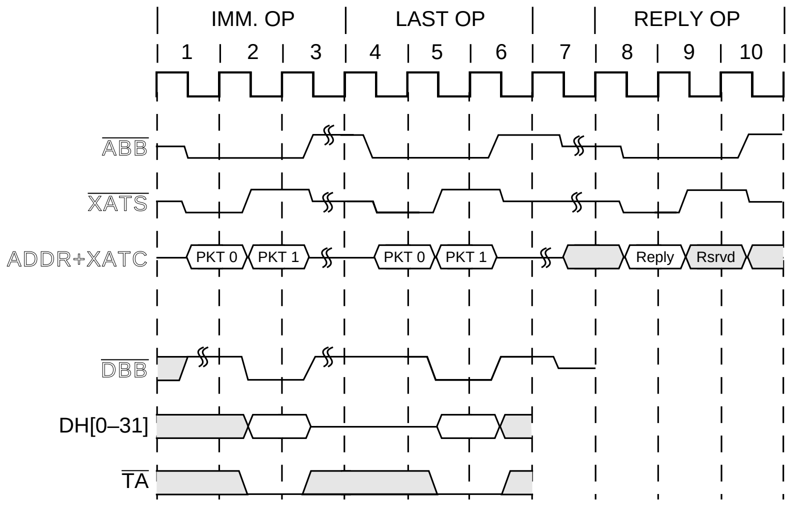


Figure C-6. Direct-Store Interface Store Access Example

C.1.3 CSE Signal

The 603 employs two-way set associativity for both the instruction and data caches, in place of the four-way set associativity of the 603e. The CSE signal indicates which cache set is being loaded during a cache line fill.

Table C-3 shows the CSE signal encoding indicating the cache set selected during a cache load operation.

Table C-3. CSE Signal Encoding

CSE	Cache Set Element
0	Set 0
1	Set 1

C.1.4 PowerPC 603 Processor Bus Clock Multiplier Configuration

The 603 provides support for bus clock multipliers of 1:1, 2:1, 3:1, and 4:1. The bus clock multipliers are selected through the setting of the PLL_CFG[0–3] signals as shown in Table C-4.

Table C-4. PowerPC 603 Microprocessor PLL Configuration

	Bus, CPU, and PLL Frequencies							
PLL_CFG 0–3	**CPU/ SYSCLK Ratio**	**Bus 16.6 MHz**	**Bus 20 MHz**	**Bus 25 MHz**	**Bus 33.3 MHz**	**Bus 40 MHz**	**Bus 50 MHz**	**Bus 66.6 MHz**
00 00	1:1	—	—	—	—	—	—	66.6 (133)
0001	1:1	—	—	—	33.3 (133)	40 (160)	50 (200)	—
0010	1:1	16.6 (133)	20 (160)	25 (200)	—	—	—	—
0100	2:1	—	—	—	66.6 (133)	80 (160)	100 (200)	—
0101	2:1	33.3 (133)	40 (160)	50 (200)	—	—	—	—
1000	3:1	—	—	75 (150)	100 (200)	—	—	—
1001	3:1	50 (200)	—	—	—	—	—	—
1100	4:1	66.6 (133)	80 (160)	100 (200)	—	—	—	—
0011	PLL bypass							
1111	Clock off							

Notes:

1. Some PLL configurations may select bus, CPU, or PLL frequencies which are not useful, not supported, or not tested for by the 603. PLL frequencies (shown in parenthesis in) should not fall below 133 MHz, and should not exceed 200 MHz.
2. In PLL bypass mode, the SYSCLK input signal clocks the internal processor directly, the PLL is disabled, and the bus mode is set for 1:1 mode operation. This mode is intended for factory use only. Note that the AC timing specifications given in this document do not apply in PLL bypass mode.
3. In clock-off mode, no clocking occurs inside the 603 regardless of the SYSCLK input.4. PLL_CFG0–PLL_CFG1 signals select the CPU-to-bus ratio (1:1, 2:1, 3:1, 4:1), PLL_CFG2–PLL_CFG3 signals select the CPU-to-PLL multiplier (x2, x4, x8).

C.1.5 PowerPC 603 Processor Cache Organization

The 603 provides two 8-Kbyte, two-way set associative caches to allow the registers and execution units rapid access to instructions and data. The instruction and data caches are configured as 128 sets of two blocks. The operation of the 603's instruction and data caches is consistent with the caches in the 603e, with the exception of the reduced cache size and set associativity.

C.1.5.1 Instruction Cache Organization

The organization of the instruction cache is shown in Figure C-7. Each cache block contains 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.

Note that address bits A20–A26 provide an index to select a set. Bits A27–A31 select a byte within a block. 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. Note that the replacement algorithm is strictly an LRU algorithm; that is, the least recently used block is filled with new instructions on a cache miss.

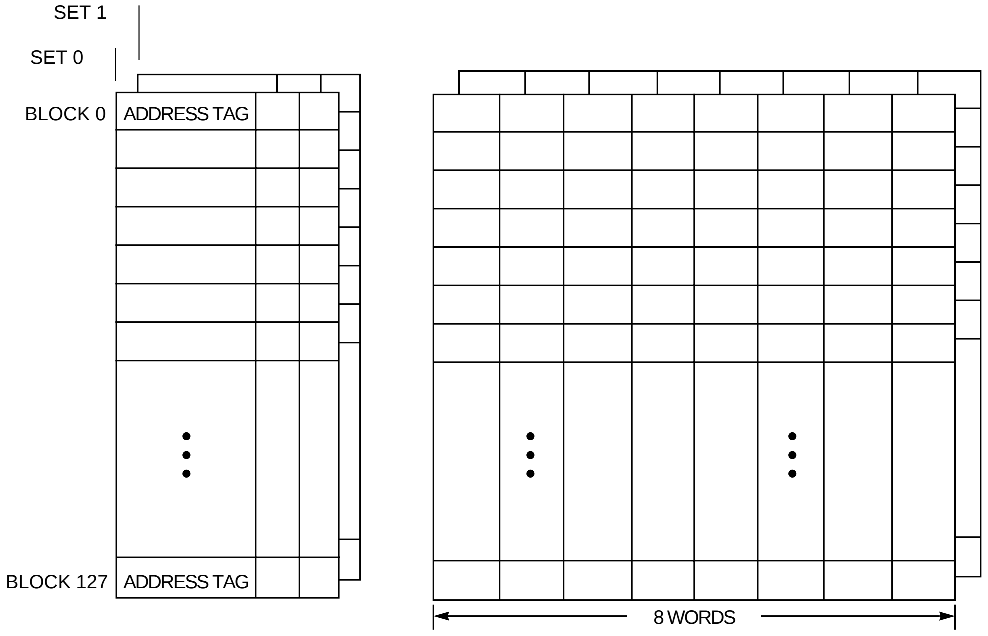


Figure C-7. Instruction Cache Organization

C.1.5.2 Data Cache Organization

The organization of the data cache is shown in Figure C-8. Each cache block contains 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.

Note that address bits A20–A26 provide an index to select a set. Bits A27–A31 select a byte within a block. 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. Note that the replacement algorithm is strictly an LRU algorithm; that is, the least recently used block is filled with new data on a cache miss.

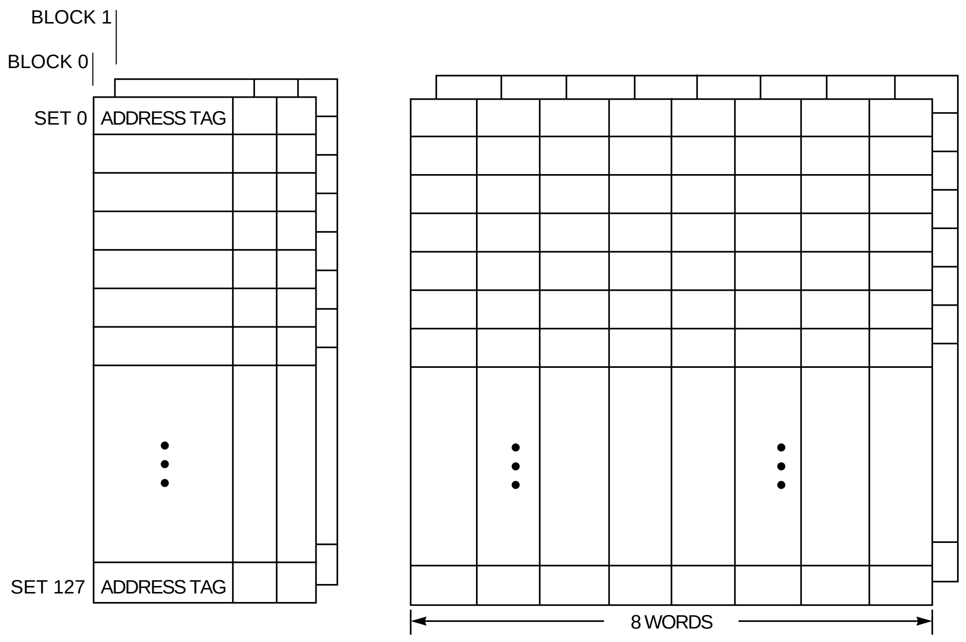


Figure C-8. Data Cache Organization

C.1.6 PLL Configuration (PLL_CFG[0–3])—Input

The 603 operates as described in Section 7.2.12.3, “PLL Configuration (PLL_CFG[0–3])—Input,” except for the following:

To avoid incorrect operation of the PLL, the clock input to the SYSCLK signal input should be stable and within the frequency range specified for the selected PLL_CFG configuration during power-up, during normal operation, or when exiting the sleep power-saving mode.

C.1.7 Address Pipelining and Split-Bus Transactions

The 603 operates as described in Section 8.2.2, “Address Pipelining and Split-Bus Transactions,” except for the following:

Note that in multiprocessor systems, addresses associated with cache line loads are not snooped between the third and fourth beat during the data tenure when the system is configured for 64-bit bus operation.

When configured for 32-bit bus operation, cache line loads are not snooped between the sixth and eighth beats. To ensure memory coherency, multiprocessor systems should avoid pipelined operation, or disallow snooping during the last data beat of a cache load operation.

C.1.8 Data Bus Arbitration

The 603 operates as described in Section 8.4.1, “Data Bus Arbitration,” except for the following:

When the 603 is configured for 1:1 processor to bus clock operation and $\overline{\text{DBG}}$ is always held asserted, multiple single-beat writes will cause incorrect data to be written to memory. The $\overline{\text{DBG}}$ signal should only be asserted when the data tenure can be started on the following bus cycle.

C.2 PowerPC 603 Processor Software Considerations

When developing software for the 603, the programmer should note the following differences from the 603e:

- The 603 supports direct-store accesses; setting T = 1 in a segment register does not result in a DSI exception.
- Store instructions have two-cycle latency and two-cycle throughput.
- The 603 does not perform integer add or compare instructions in the SRU.
- The 603 does not implement the key bit (bit 12) in SRR1 to provide information about memory protection violations prior to page table search operations.
- HID1 is not implemented by the 603; no read-only access to the PLL_CFG signal configuration is provided.
- The PVR value for the 603 is 0x0003.

The following sections provide further information on the 603 attributes that may affect software written for the 603e.

C.2.1 Direct-Store Interface Address Translation

With address translation enabled, all memory accesses generated by the 603 map to a segment descriptor in the segment table. If T = 1 for the selected segment descriptor and there are no BAT hits, the access maps to the direct-store interface, invoking a specific bus protocol for accessing some special-purpose I/O devices. Direct-store segments are provided for POWER compatibility. As the direct-store interface is present only for compatibility with existing I/O devices that used this interface and the direct-store interface protocol is not optimized for performance, its use is discouraged. The selection of address translation type differs for instruction and data accesses only in that instruction accesses are not allowed from direct-store segments; attempting to fetch an instruction from a direct-store segment causes an ISI exception.Applications that require low latency load/store access to external address space should use memory-mapped I/O, rather than the direct-store interface. Refer to Chapter 5, “Memory Management” for additional information about address translation and memory accesses.

C.2.1.1 Direct-Store Segment Translation Summary Flow

Figure C-9 shows the flow used by the MMU when direct-store segment address translation is selected. In the case of a floating-point load or store operation to a direct-store segment, other implementations may not take an alignment exception, as is allowed by the PowerPC architecture. In the case of an **eciwx**, **ecowx**, **lwarx**, or **stwcx.** instruction, the 603 sets the DSISR register as shown and causes the DSI exception.

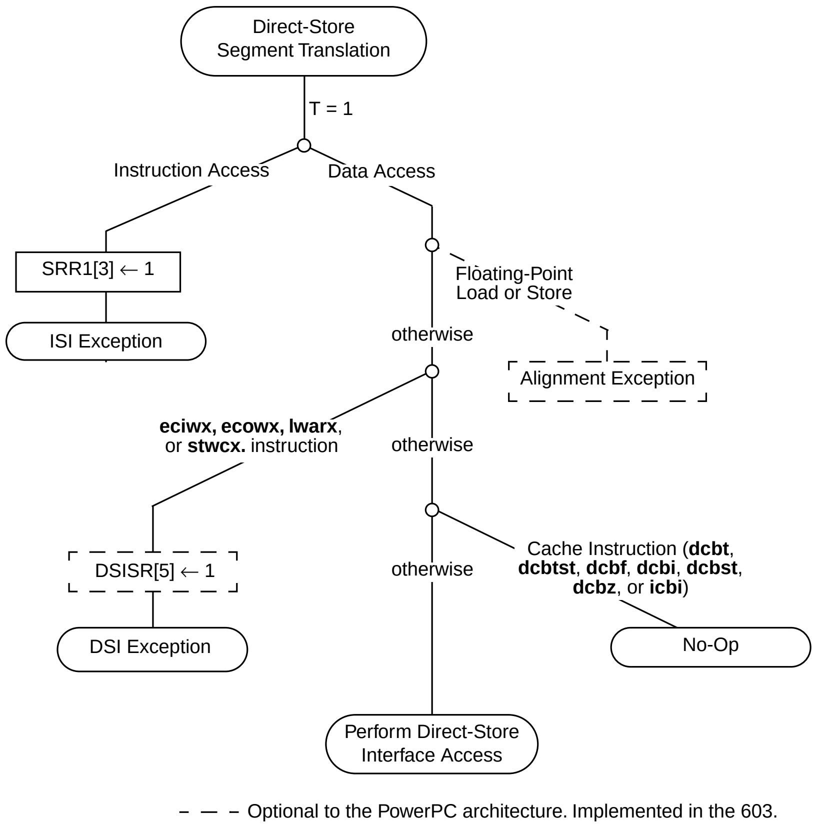

Direct-Store Segment Translation
T = 1
Instruction Access
Data Access
SRR1[3] ← 1
ISI Exception
Floating-Point Load or Store
otherwise
Alignment Exception
eciwx, ecowx, lwarx, or stwcx. instruction
otherwise
DSISR[5] ← 1
DSI Exception
Cache Instruction (dcbt, dcbtst, dcbf, dcbi, dcbst, dcbz, or icbi)
otherwise
No-Op
Perform Direct-Store Interface Access
– — – Optional to the PowerPC architecture. Implemented in the 603.

Figure C-9. Direct-Store Segment Translation Flow

A direct-store access occurs when a data access is initiated and SR[T] is set. In the 603, MSR[DR] is a don't care for this case. The following apply for direct-store accesses:

- Floating-point loads and stores to direct-store segments always cause an alignment exception, regardless of operand alignment.
- **lwarx** or **stwcx.** instructions that map into a direct-store segment always cause a DSI exception. However, if the instruction crosses a segment boundary, an alignment exception is taken instead.

C.2.1.2 Direct-Store Interface Accesses

When the address translation process determines that the segment descriptor has T = 1, direct-store interface address translation is selected and no reference is made to the page tables and referenced and changed bits are not updated. These accesses are performed as if the WIMG bits were 0b0101; that is, caching is inhibited, the accesses bypass the cache, hardware-enforced coherency is not required, and the accesses are considered guarded.

The specific protocol invoked to perform these accesses involves the transfer of address and data information in packets; however, the PowerPC OEA does not define the exact hardware protocol used for direct-store interface accesses. Some instructions cause multiple address/data transactions to occur on the bus. In this case, the address for each transaction is handled individually with respect to the DMMU.

The following data is sent by the 603 to the memory controller in the protocol (two packets consisting of address-only cycles).

- Packet 0
  - One of the Kx bits (Ks or Kp) is selected to be the key as follows:
    - For supervisor accesses (MSR[PR] = 0), the Ks bit is used and Kp is ignored.
    - For user accesses (MSR[PR] = 1), the Kp bit is used and Ks is ignored.
  - The contents of bits 3–31 of the segment register, which is the BUID field concatenated with the “controller-specific” field.
- Packet 1—SR[28–31] concatenated with the 28 lower-order bits of the effective address, EA4–EA31.

C.2.1.3 Direct-Store Segment Protection

Page-level memory protection as described in Section 5.4.2, “Page Memory Protection,” is not provided for direct-store segments. The appropriate key bit (Ks or Kp) from the segment descriptor is sent to the memory controller, and the memory controller implements any protection required. Frequently, no such mechanism is provided; the fact that a direct-store segment is mapped into the address space of a process may be regarded as sufficient authority to access the segment.

C.2.1.4 Instructions Not Supported in Direct-Store Segments

The following instructions are not supported at all and cause a DSI exception in the 603 (with DSISR[5] set) when issued with an effective address that selects a segment descriptor that has T = 1 (or when MSR[DR] = 0):

- **lwarx**
- **stwcx.**
- **eciwx**
- **ecowx**

C.2.1.5 Instructions with No Effect in Direct-Store Segments

The following instructions are executed as no-ops by the 603 when issued with an effective address that selects a segment where T = 1:

- **dcbt**
- **dcbtst**
- **dcbf**
- **dcbi**
- **dcbst**
- **dcbz**
- **icbi**

C.2.2 Store Instruction Latency

The store instructions executed by the 603 execute with 2-cycle latency, and 2-cycle throughput, in contrast to the 2-cycle latency and 1-cycle throughput of the 603e. Table C-5 provides the latencies for the store instructions executed by the 603.

Table C-5. Store Instruction Timing

Primary	Extended	Mnemonic	Unit	Cycles
31	151	**stwx**	LSU	2:2
31	183	**stwux**	LSU	2:2
31	215	**stbx**	LSU	2:2
31	247	**stbux**	LSU	2:2
31	407	**sthx**	LSU	2:2
31	438	**ecowx**	LSU	2:2
31	439	**sthux**	LSU	2:2
31	662	**stwbrx**	LSU	2:2
31	663	**stfsx**	LSU	2:2
31	695	**stfsux**	LSU	2:2
31	727	**stfdx**	LSU	2:2

Table C-5. Store Instruction Timing (Continued)

Primary	Extended	Mnemonic	Unit	Cycles
31	918	**sthbrx**	LSU	2:2
31	983	**stfiwx**	LSU	2:2
36	---	**stw**	LSU	2:2
37	---	**stwu**	LSU	2:2
38	---	**stb**	LSU	2:2
39	---	**stbu**	LSU	2:2
44	---	**sth**	LSU	2:2
45	---	**sthu**	LSU	2:2
52	---	**stfs**	LSU	2:2
53	---	**stfsu**	LSU	2:2
54	---	**stfd**	LSU	2:2
55	---	**stfdu**	LSU	2:2

C.2.3 Instruction Execution by System Register Unit

Unlike the 603e, the 603's SRU does not execute integer add and compare instructions. Table C-6 lists the instructions executed by the 603's SRU, and the number of cycles required for execution.

Table C-6. System Register Instructions

Primary	Extended	Mnemonic	Unit	Cycles
17	- -1	**sc**	SRU	3
19	050	**rfi**	SRU	3
19	150	**isync**	SRU	1&
31	083	**mfmsr**	SRU	1
31	146	**mtmsr**	SRU	2
31	210	**mtsr**	SRU	2
31	242	**mtsrin**	SRU	2
31	339	**mfspr** (not I/DBATs)	SRU	1
31	339	**mfspr** (DBATs)	SRU	3&
31	339	**mfspr** (IBATs)	SRU	3&
31	467	**mtspr** (not IBATs)	SRU	2 (XER-&)
31	467	**mtspr** (IBATs)	SRU	2&
31	595	**mfsr**	SRU	3&
31	598	**sync**	SRU	1&

Table C-6. System Register Instructions (Continued)

Primary	Extended	Mnemonic	Unit	Cycles
31	659	**mfsrin**	SRU	3&
31	854	**eieio**	SRU	1
31	371	**mftb**	SRU	1
31	467	**mttb**	SRU	1

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

C.2.4 Machine Check Exception (0x00200)

The 603 operates as described in Section 4.5.2, "Machine Check Exception (0x00200)," with the exception of the following:

To ensure memory coherency following the assertion of $\overline{\text{TEA}}$, the instruction cache should be invalidated by setting and clearing HID0[ICFI], and flushing the data cache before performing any load or store operations, or executing any data cache management instructions other than **dcbf**.

Note that an assertion of $\overline{\text{TEA}}$ during an instruction fetch will result in an immediate instruction refetch before the machine check exception is taken, which will result in a second assertion of the $\overline{\text{TEA}}$ signal. The second assertion of $\overline{\text{TEA}}$ while the machine check exception is pending from the previous $\overline{\text{TEA}}$ assertion will result in the 603 entering the checkstop state instead of taking the machine check exception.

C.2.5 Instruction Address Breakpoint Exception (0x01400)

The 603 operates as described in Section 4.5.15, "Instruction Address Breakpoint Exception (0x01300)," with the exception of the following:

To avoid spurious IABR exceptions, the IABR special-purpose register should not be loaded with an address that falls within the same cache line as a disabled, but matching IABR address.

C.2.6 Cache Control Instructions

The 603 operates as described in Section 3.7, "Cache Control Instructions," with the exception of the following:

Note that loop structures that contain long sequences of **dcbz** or **dcbi** instructions may cause snoop performance degradation. Programmers can improve snoop performance by inserting no-op instructions (**ori** 0,0,0) between **dcbz** or **dcbi** instructions, replacing the **dcbz** or **dcbi** instructions with a sequence of write-through store operations, using the decrementer to generate a periodic exception to allow snoop activity, or mapping the address space where the **dcbz** or **dcbi** instructions execute as global (M = 1).

Note that the use of the **dcbz** instruction in a multiprocessor system can result in loss of data coherency if the **dcbz** instruction is executed in memory space marked as global (M = 1). Programmers should use software coherency protocols to ensure that no processor can perform a kill operation to memory used by another processor.

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

Atomic. 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 PowerPC 603e microprocessor initiates the read and write separately, but signals the memory system that it is attempting an atomic operation. If the operation fails, status is kept so that the 603e can try again. The 603e implements atomic accesses through the **lwarx**/**stwcx.** instruction pair.

B

Beat. A single state on the 603e bus interface that may extend across multiple bus cycles. A 603e transaction can be composed of multiple address or data *beats*.

Biased exponent. The sum of the exponent and a constant (bias) chosen to make the biased exponent's range non-negative.

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.

Boundedly undefined. The results of attempting to execute a given instruction are said to be *boundedly undefined* if they 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. Boundedly undefined results for a given instruction may vary between implementations, and between execution attempts in the same implementation.

Branch folding. A technique of removing the branch instruction from the instruction sequence.

Burst. A multiple beat data transfer whose total size is typically equal to a cache block (in the 603e, a 32-byte 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 containing recently accessed data and/or instructions (subset of main memory).

Cache block. The cacheable unit for a PowerPC processor. The size of a cache block may vary among processors. For the 603e, it is one cache line (8 words).

Cache coherency. 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.

Cast-outs. Cache block that must be written to memory when a snoop miss causes the least recently used block with modified data to be replaced.

Context synchronization. Context synchronization is the result of specific instructions (such as **sc** or **rfi**) or when certain events occur (such as an exception). During context synchronization, all instructions in execution complete past the point where they can produce an exception; all instructions in execution complete in the context in which they began execution; all subsequent instructions are fetched and executed in the new context.

Copy-back operation. A cache operation in which a cache line is copied back to memory to enforce cache coherency. Copy-back operations consist of snoop push-out operations and cache cast-out operations.

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-store segment access. An access to an I/O address space. The 603 defines separate memory-mapped and I/O address spaces, or segments, distinguished by the corresponding segment register T bit in the address translation logic of the 603. If the T bit is cleared, the memory reference is a normal memory-mapped access and can use the virtual memory management hardware of the 603. If the T bit is set, the memory reference is a direct-store access.

E

E**xception**. An unusual or error condition encountered by the processor that results in special processing.

Exception handler. A software routine that executes when an exception occurs. Normally, the exception handler corrects the condition that caused the exception, or performs some other meaningful task (such as aborting the program that caused the exception). The addresses of the exception handlers are defined by a two-word exception vector that is branched to automatically when an exception occurs.

Exclusive state. EMI state (E) in which only one caching device contains data that is also in system memory.

Execution synchronization. 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. The component of a binary floating-point number that normally signifies the integer power to which two is raised in determining the value of the represented number. Occasionally the exponent is called the signed or unbiased exponent.

F

Feed-forwarding. A 603e 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.

Floating-point unit. The functional unit in the 603e processor responsible for executing all floating-point instructions.
(Not supported on the EC603e microprocessor)

Flush. An operation that causes a modified cache block to be invalidated and the data to be written to memory.

Fraction. The field of the significand that lies to the right of its implied binary point.

G

General-purpose register. Any of the 32 registers in the 603e register file. These registers provide the source operands and destination results for all 603e data manipulation instructions. Load instructions move data from memory to registers, and store instructions move data from registers to memory.

I

IEEE 754. A standard written by the Institute of Electrical and Electronics Engineers that defines operations of binary floating-point arithmetic and representations of binary floating-point numbers.

Instruction queue. A holding place for instructions fetched from the current instruction stream.

Integer unit. The functional unit in the 603e responsible for executing all integer instructions.

Interrupt. An external signal that causes the 603e to suspend current execution and take a predefined exception.

Invalid state. EMI state (I) that indicates that the cache block does not contain valid data.

K

Kill. An operation that causes a cache block to be invalidated.

L

Latency. The number of clock cycles necessary to execute an instruction and make ready the results of that instruction.

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.

Livelock. A state in which processors interact in a way such that no processor makes progress.

M

Mantissa. The decimal part of logarithm.

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, EMI 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-forced I/O controller interface access. These accesses are made to memory space. They do not use the extensions to the memory protocol described for I/O controller interface accesses, and they bypass the page- and block-translation and protection mechanisms.

Memory management unit. The functional unit in the 603e that translates the logical address bits to physical address bits.

Modified state. EMI state (M) in which one, and only one, caching device has the valid data for that address. The data at this address in external memory is not valid.

N

NaN. An abbreviation for not a number; a symbolic entity encoded in floating-point format. There are two types of NaNs—signaling NaNs and quiet NaNs.

No-op. No-operation. A single-cycle operation that does not affect registers or generate bus activity.

O

Out-of-order. An operation is said to be out-of-order when it is not guaranteed to be required by the sequential execution model, such as the execution of an instruction that follows another instruction that may alter the instruction flow. For example, execution of instructions in an unresolved branch is said to be out-of-order, as is the execution of an instruction behind another instruction that may yet cause an exception. The results of operations that are performed out-of-order are not committed to architected resources until it can be ensured that these results adhere to the in-order, or sequential execution model.

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 added, the sum may require 33 bits due to carry. Since the 32-bit registers of the 603e cannot represent this sum, an overflow condition occurs.

P

Packet. A term used in the 603 with respect to direct store operations.

Page. A 4-Kbyte area of memory, aligned on a 4-Kbyte boundary.

Park. The act of allowing a bus master to maintain mastership of the bus without having to arbitrate.

Pipelining. A technique that breaks instruction execution into distinct steps so that multiple steps can be performed at the same time.

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.

Q

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**.

Quiet NaNs. Propagate through almost every arithmetic operation without signaling exceptions. These are used to represent the results of certain invalid operations, such as invalid arithmetic operations on infinities or on NaNs, when invalid.

S

Scan interface. The 603e's test interface.

Shadowing. Shadowing allows a register to be updated by instructions that are executed out of order without destroying machine state information.

Signaling NaNs. Signal the invalid operation exception when they are specified as arithmetic operands

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.

Static branch prediction. Mechanism by which software (for example, compilers) can give a hint to the machine hardware about the direction the branch is likely to take.

Superscalar machine. A machine that can issue multiple instructions concurrently from a conventional linear instruction stream.

Supervisor mode. The privileged operation state of the 603e. In supervisor mode, software can access all control registers and can access the supervisor memory space, among other privileged operations.

T

Tenure. The period of bus mastership. For the 603e, there can be separate address bus tenures and data bus tenures. A tenure consists of three phases: arbitration, transfer, termination

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

Underflow. An error condition that occurs during arithmetic operations when the result cannot be represented accurately in the destination register. For example, underflow can happen if two floating-point fractions are multiplied and the result is a single-precision number. The result may require a larger exponent and/or mantissa than the single-precision format makes available. In other words, the result is too small to be represented accurately.

User mode. The unprivileged operating state of the 603e. In user mode, software can only access certain control registers and can only access user memory space. No privileged operations can be performed.

W

Write-through. A memory update policy in which all processor write cycles are written to both the cache and memory.

INDEX

INDEX

INDEX

INDEX

INDEX

INDEX

INDEX

INDEX

INDEX

INDEX

Attention!

This book is a companion to the *PowerPC Microprocessor Family: The Programming Environments*, referred to as *The Programming Environments Manual*. Note that the companion *Programming Environments Manual* exists in two versions. See the Preface for a description of the following two versions:

- *PowerPC Microprocessor Family: The Programming Environments,* Rev 1
  Order #: MPCFPE/AD
- *PowerPC Microprocessor Family: The Programming Environments for 32-Bit Microprocessors, Rev 1*
  Order #: MPCFPE32B/AD

Call the Motorola LDC at 1-800-441-2447 (website: http://ldc.nmd.com) or contact your local sales office to obtain copies.