MPC7450 RISC Microprocessor Family Reference Manual

Supports

MPC7448
MPC7447A
MPC7457
MPC7447
MPC7455
MPC7445
MPC7451
MPC7441
MPC7450

MPC7450UM
Rev. 5
1/2005

freescale™
semiconductor

How to Reach Us:

Home Page:
www.freescale.com

email:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
(800) 521-6274
480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku
Tokyo 153-0064, Japan
0120 191014
+81 2666 8080
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate,
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
(800) 441-2447
303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor
@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. The described product is a PowerPC microprocessor. The PowerPC name is a trademark of IBM Corp. and used under license. All other product or service names are the property of their respective owners.

© Freescale Semiconductor, Inc. 2005. All rights reserved.

MPC7450UM
Rev. 5
1/2005

freescale™
semiconductor

Contents

Paragraph Number **Title** **Page Number**

About This Book

Chapter 1
Overview

Contents

Paragraph Number **Title** **Page Number**

Contents

Paragraph Number | **Title** | **Page Number**

Chapter 2
Programming Model

Contents

Paragraph Number **Title** **Page Number**

Contents

Paragraph Number | **Title** | **Page Number**

Contents

Paragraph Number | **Title** | **Page Number**

Contents

Paragraph Number **Title** **Page Number**

Contents

Paragraph Number | **Title** | **Page Number**

Chapter 3
L1, L2, and L3 Cache Operation

Contents

Paragraph Number **Title** **Page Number**

Contents

Paragraph Number **Title** **Page Number**

Contents

Paragraph Number **Title** **Page Number**

Contents

Paragraph Number **Title** **Page Number**

Chapter 4
Exceptions

Contents

Paragraph Number **Title** **Page Number**

Chapter 5
Memory Management

Contents

Paragraph Number | **Title** | **Page Number**

Chapter 6
Instruction Timing

Contents

Paragraph Number **Title** **Page Number**

Contents

Paragraph Number **Title** **Page Number**

Chapter 7
AltiVec Technology Implementation

Contents

Paragraph Number **Title** **Page Number**

Chapter 8
Signal Descriptions

Contents

Contents

Paragraph Number **Title** **Page Number**

Contents

Paragraph Number **Title** **Page Number**

Contents

Chapter 9
System Interface Operation

Contents

Paragraph Number **Title** **Page Number**

Contents

Paragraph Number **Title** **Page Number**

Chapter 10
Power and Thermal Management

Chapter 11
Performance Monitor

Contents

Paragraph Number **Title** **Page Number**

Appendix A
MPC7450 Instruction Set Listings

Appendix B
Instructions Not Implemented

Appendix C
Special-Purpose Registers

Appendix D
Revision History

Glossary

Index

Contents

Paragraph Number | **Title** | **Page Number**

Figures

Figures

Figure Number | **Title** | **Page Number**

Figures

Figures

Figure Number **Title** **Page Number**

Figures

Figure Number **Title** **Page Number**

Figures

Figure Number **Title** **Page Number**

Tables

Table Number **Title** **Page Number**

Tables

Table Number **Title** **Page Number**

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Table Number | **Title** | **Page Number**

About This Book

The primary objective of this reference manual is to describe the functionality of the MPC7450 for software and hardware developers. The MPC7450 is a PowerPC™ microprocessor. In addition, this manual supports the MPC7441, MPC7445, MPC7451, MPC7455, MPC7457, MPC7447, MPC7447A, and MPC7448. This book is written from the perspective of the MPC7450, and unless otherwise noted, the information applies also to the MPC7441, MPC7445, MPC7451, MPC7455, MPC7457, MPC7447, MPC7447A, and MPC7448. The MPC7450 has the same functionality as the MPC7451 and any differences in data regarding bus timing, signal behavior, and AC, DC, and thermal characteristics are in the hardware specifications. The differences between the various processors are summarized in Section 1.5, "Differences Between MPC7441/MPC7451 and MPC7445/MPC7455," Section 1.6, "Differences Between MPC7441/MPC7451 and MPC7447/MPC7457," Section 1.7, "Differences Between MPC7447 and MPC7447A," and Section 1.8, "Differences Between MPC7447A and MPC7448."

This book is intended as a companion to the *Programming Environments Manual for 32-Bit Implementations of the PowerPC Architecture* (referred to as the *Programming Environments Manual*).

NOTE

About the Companion *Programming Environments Manual*

> This manual, which describes MPC7450 features not defined by the architecture, is to be used with the *Programming Environments Manual*.
>
> Because the PowerPC architecture definition is flexible to support a broad range of processors, the *Programming Environments Manual* describes generally those features common to these processors and indicates which features are optional or may be implemented differently in the design of each processor.
>
> Note that the *Programming Environments Manual* describes only PowerPC architecture features for 32-bit implementations.
>
> Go to www.freescale.com or contact a local sales representative for a copy of the *Programming Environments Manual*.

This manual 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 architecture level to which user-level software should conform. The UISA defines the base user-level instruction set, user-level registers, data types, memory conventions, and the memory and programming models seen by application programmers.
- PowerPC virtual environment architecture (VEA)—The VEA, which is the smallest component of the PowerPC architecture, defines additional user-level functionality that falls outside typical user-level software requirements. The VEA describes the memory model for an environment in

which multiple processors or other devices can access external memory and defines aspects of the cache model and cache control instructions from a user-level perspective. VEA resources are particularly useful for optimizing memory accesses and for managing resources in an environment in which other processors and other devices can access external memory.

Implementations that conform to the VEA also conform to the UISA but may not necessarily adhere to the OEA.

- PowerPC operating environment architecture (OEA)—The OEA defines supervisor-level resources typically required by an operating system. It defines the memory management model, supervisor-level registers, and the exception model.

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

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, but the exception mechanism itself is defined by the OEA.

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

For ease in reference, topics in this book are presented in the same order as the *Programming Environments Manual*. Topics build on one another, beginning with a description and complete summary of the MPC7450 programming model (registers and instructions) and progressing to more specific, architecture-based topics 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.

Additionally, the MPC7450 implements the AltiVec™ technology resources. The following two books describe the AltiVec technology:

- *AltiVec Technology Programming Environments Manual* (AltiVec PEM*)* is a reference guide for programmers. The AltiVec PEM uses a standardized format instruction to describe each instruction, showing syntax, instruction format, register translation language (RTL) code that describes how the instruction works, and a listing of which, if any, registers are affected. At the bottom of each instruction entry is a figure that shows the operations on elements within source operands and where the results of those operations are placed in the destination operand.
- *AltiVec Technology Programming Interface Manual* (AltiVec PIM) describes how programmers can access AltiVec functionality from programming languages such as C and C++. The AltiVec PIM describes the high-level language interface and application binary interface for System V and embedded applications for use with the AltiVec instruction set extension to the PowerPC architecture.

The *PowerPC Architecture: A Specification for a New Family of RISC Processors* defines the architecture from the perspective of the three programming environments and remains the defining document for the PowerPC architecture. For information on ordering Freescale documentation, see “Related Documentation,” on page xlviii.

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.

To locate any published errata or updates for this document, refer to the world-wide web at http://www.freescale.com.

A list of the major differences between revisions of this manual is provided in Appendix D, “Revision History.”

Audience

This manual is intended for system software and hardware developers and applications programmers who want to develop products for the MPC7441, MPC7445, MPC7450, MPC7451, MPC7455, MPC7457, MPC7447, MPC7447A, and MPC7448. It is assumed that the reader understands operating systems, microprocessor system design, basic principles of RISC processing, and details of the PowerPC architecture.

Organization

The following is a summary and a brief description of the chapters in 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 MPC7450. 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,” is useful for software engineers who need to understand the MPC7450-specific registers, operand conventions, and details regarding how PowerPC instructions are implemented on the MPC7450. Instructions are organized by function.
- Chapter 3, “L1, L2, and L3 Cache Operation,” discusses the cache and memory model as implemented on the MPC7450.
- Chapter 4, “Exceptions,” describes the exception model defined in the OEA and the specific exception model implemented on the MPC7450.
- Chapter 5, “Memory Management,” describes the implementation of the memory management unit on the MPC7450 as specified by the OEA.
- Chapter 6, “Instruction Timing,” provides information about latencies, interlocks, special situations, and various conditions that help make programming more efficient. This chapter is of special interest to software engineers and system designers.
- Chapter 7, “AltiVec Technology Implementation,” summarizes the features and functionality provided by the implementation of the AltiVec technology.
- Chapter 8, “Signal Descriptions,” provides descriptions of individual signals of the MPC7450.
- Chapter 9, “System Interface Operation,” describes signal timings for various operations. It also provides information for interfacing to the MPC7450.
- Chapter 10, “Power and Thermal Management,” provides information about power saving and thermal management for the MPC7450.

- Chapter 11, “Performance Monitor,” describes the operation of the performance monitor diagnostic tool incorporated in the MPC7450.
- Appendix A, “MPC7450 Instruction Set Listings,” lists all PowerPC instructions while indicating those instructions that are not implemented by the MPC7450; it also includes the instructions that are specific to the MPC7450. 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 the 32- and 64-bit PowerPC instructions not implemented in the MPC7450.
- Appendix C, “Special-Purpose Registers,” lists all MPC7450 SPRs.
- Appendix D, “Revision History,” lists the major differences between revisions of the *MPC7450 RISC Microprocessor Reference Manual*.
- 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, available through Morgan-Kaufmann Publishers, 340 Pine Street, Sixth Floor, San Francisco, CA, provides useful information about the PowerPC architecture and computer architecture in general:

- The *PowerPC Architecture: A Specification for a New Family of RISC Processors*, Second Edition, by International Business Machines, Inc.

  For updates to the specification, see 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.
- *Computer Architecture: A Quantitative Approach*, Third Edition, by John L. Hennessy and David A. Patterson.
- *Computer Organization and Design: The Hardware/Software Interface*, Second Edition, David A. Patterson and John L. Hennessy.

Related Documentation

Freescale 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:

- *Programming Environments Manual for 32-Bit Implementations of the PowerPC Architecture* (MPCFPE32B/AD)—Describes resources defined by the PowerPC architecture.
- Reference manuals—These books provide details about individual implementations and are intended for use with the *Programming Environments Manual.*

- Addenda/errata to reference manuals—Because some processors have follow-on parts, an addendum is provided that describes additional features and functionality changes. These addenda are intended for use with the corresponding reference manuals.
- Hardware specifications—Hardware specifications provide specific data regarding bus timing, signal behavior, and AC, DC, and thermal characteristics, as well as other design considerations. Separate hardware specifications are provided for each part described in this book (MPC7441, MPC7445, MPC7447, MPC7450, MPC7451, MPC7455, MPC7457, MPC7447A, and MPC7448). Note that when referring to the hardware specifications throughout this book, make sure to refer to the appropriate hardware specifications for the part being used.
- Technical summaries—Each device 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 reference manual.
- The *Programmer's Reference Guide for the PowerPC Architecture* (MPCPRG)—This concise reference includes the register summary, memory control model, exception vectors, and the PowerPC instruction set.
- The *Programmer's Pocket Reference Guide for the PowerPC Architecture* (MPCPRGREF)—This foldout card provides an overview of PowerPC registers, instructions, and exceptions for 32-bit implementations.
- Application notes—These short documents address specific design issues useful to programmers and engineers working with Freescale processors.

Additional literature is published as new processors become available. For a current list of documentation, refer to http://www.freescale.com.

Conventions

This manual uses the following notational conventions:

cleared/set	When a bit takes the value zero, it is said to be cleared; when it takes a value of one, it is said to be set.
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
	Internal signals are set in italics, for example, $\overline{qual\ BG}$
0x0	Prefix to denote hexadecimal number
0b0	Prefix to denote binary number
rA, **r**B	Instruction syntax used to identify a source GPR
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 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 some contexts, such as signal encodings, an unitalicized x indicates a don’t care.
x	An italicized *x* indicates an alphanumeric variable
n	An italicized *n* indicates an numeric variable
¬	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 can be written to as 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
ADB	Allowable disconnect boundary
ALU	Arithmetic logic unit
BAT	Block address translation
BHT	Branch history table
BIST	Built-in self test
BIU	Bus interface unit
BPU	Branch processing unit
BSDL	Boundary-scan description language
BTIC	Branch target instruction cache
CMOS	Complementary metal-oxide semiconductor
COP	Common on-chip processor
CQ	Completion queue
CR	Condition register
CTR	Count register
DABR	Data address breakpoint register
DAR	Data address register
DBAT	Data BAT

Table i. Acronyms and Abbreviated Terms (continued)

Term	Meaning
DCMP	Data TLB compare
DEC	Decrementer register
DLL	Delay-locked loop
DMISS	Data TLB miss address
DMMU	Data MMU
DPM	Dynamic power management
DSISR	Register used for determining the source of a DSI exception
DTLB	Data translation lookaside buffer
EA	Effective address
EAR	External access register
ECC	Error checking and correction
FIFO	First-in-first-out
FIQ	Floating-point issue queue
FPR	Floating-point register
FPSCR	Floating-point status and control register
FPU	Floating-point unit
GIQ	General-purpose register issue queue
GPR	General-purpose register
HID*n*	Hardware implementation-dependent register
IABR	Instruction address breakpoint register
IBAT	Instruction BAT
ICTC	Instruction cache throttling control register
IEEE	Institute for Electrical and Electronics Engineers
IMMU	Instruction MMU
IQ	Instruction queue
ITLB	Instruction translation lookaside buffer
IU	Integer unit
JTAG	Joint Test Action Group
L2	Secondary cache (level 2 cache)
L2CR	L2 cache control register
L3	Level 3 cache
LIFO	Last-in-first-out

Table i. Acronyms and Abbreviated Terms (continued)

Term	Meaning
LR	Link register
LRU	Least recently used
LSB	Least-significant byte
lsb	Least-significant bit
LSQ	Least-significant quad word
lsq	Least-significant quad word
LSU	Load/store unit
MESI	Modified/exclusive/shared/invalid—cache coherency protocol
MMCR*n*	Monitor mode control registers
MMU	Memory management unit
MSB	Most-significant byte
msb	Most-significant bit
MSQ	Most-significant quad word
msq	Most-significant quad word
MSR	Machine state register
NaN	Not a number
No-op	No operation
OEA	Operating environment architecture
PEM	*Programming Environments Manual*
PID	Processor identification tag
PIM	*Programming Interface Manual*
PLL	Phase-locked loop
PLRU	Pseudo least recently used
PMC*n*	Performance monitor counter registers
POR	Power-on reset
POWER	Performance optimized with enhanced RISC architecture
PTE	Page table entry
PTEG	Page table entry group
PVR	Processor version register
RAW	Read-after-write
RISC	Reduced instruction set computing
RTL	Register transfer language

Table i. Acronyms and Abbreviated Terms (continued)

Term	Meaning
RWITM	Read with intent to modify
RWNITM	Read with no intent to modify
SDA	Sampled data address register
SDR1	Register that specifies the page table base address for virtual-to-physical address translation
SIA	Sampled instruction address register
SPR	Special-purpose register
SR*n*	Segment register
SRR0	Machine status save/restore register 0
SRR1	Machine status save/restore register 1
SRU	System register unit
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
UMMCR*n*	User monitor mode control registers
UPMC*n*	User performance monitor counter registers
USIA	User sampled instruction address register
VEA	Virtual environment architecture
VFPU	Vector floating-point unit
VIQ	Vector issue queue
VIU1	Vector instruction unit 1
VIU2	Vector instruction unit 2
VPN	Virtual page number
VPU	Vector permute unit
VSID	Virtual segment identification
VTQ	Vector touch queue
WAR	Write-after-read
WAW	Write-after-write

Table i. Acronyms and Abbreviated Terms (continued)

Term	Meaning
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 and the equivalent terminology used in the PowerPC architecture specification.

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 the MPC7450 microprocessor features and includes a block diagram that shows the major functional components. The MPC7450 is a PowerPC™ microprocessor. This chapter also provides information about how the MPC7450 implementation complies with the PowerPC and AltiVec™ architecture definitions. In addition, this manual supports the MPC7441, MPC7445, MPC7451, MPC7455, MPC7457, MPC7447, MPC7447A, and MPC7448. Any differences between the MPC7450 and the other microprocessors, including the MPC7451, are noted in the reference manual.

1.1 MPC7450 Microprocessor Overview

This section describes the features and general operation of the MPC7450 and provides a block diagram showing the major functional units. The MPC7450 implements the PowerPC architecture and is a reduced instruction set computer (RISC) microprocessor. The MPC7450 consists of a processor core, 32-Kbyte separate L1 instruction and data caches, a 256-Kbyte L2 cache (512-Kbyte for MPC7457 and 1 Mbyte for the MPC7448), and an internal L3 controller with tags that support a glueless backside L3 cache through a dedicated high-bandwidth interface. The MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448 do not support the L3 cache and the L3 interface. The core is a high-performance superscalar design supporting multiple execution units, including four independent units that execute AltiVec instructions.

The MPC7450 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 MPC7450 provides virtual memory support for up to 4 Petabytes (2^{52}) of virtual memory and real memory support for up to 64 Gigabytes (2^{36}) of physical memory.

The MPC7450 also implements the AltiVec instruction set architectural extension. The MPC7450 is a superscalar processor that can dispatch and complete three instructions simultaneously. It incorporates the following execution units:

- 64-bit floating-point unit (FPU)
- Branch processing unit (BPU)
- Load/store unit (LSU)
- Four integer units (IUs):
  - Three shorter latency IUs (IU1a–IU1c)—execute all integer instructions except multiply, divide, and move to/from special-purpose register (SPR) instructions.
  - Longer latency IU (IU2)—executes miscellaneous instructions including condition register (CR) logical operations, integer multiplication and division instructions, and move to/from SPR instructions.

- Four vector units that support AltiVec instructions:
  - — Vector permute unit (VPU)
  - — Vector integer unit 1 (VIU1)—performs shorter latency integer calculations
  - — Vector integer unit 2 (VIU2)—performs longer latency integer calculations
  - — Vector floating-point unit (VFPU)

The ability to execute several instructions in parallel and the use of simple instructions with rapid execution times yield high efficiency and throughput for MPC7450-based systems. Most integer instructions (including VIU1 instructions) have a one-clock cycle execution latency.

Several execution units feature multiple-stage pipelines; that is, the tasks they perform are broken into subtasks executed in successive stages. Typically, instructions follow one another through the stages, so a four-stage unit can work on four instructions when its pipeline is full. So, although an instruction may have to pass through several stages, the execution unit can achieve a throughput of one instruction per clock cycle.

AltiVec computational instructions are executed in four independent, pipelined AltiVec execution units. A maximum of two AltiVec instructions can be issued in order to any combination of AltiVec execution units per clock cycle. Moreover, the VIU2, VFPU, and VPU are pipelined, so they can operate on multiple instructions. The VPU has a two-stage pipeline; the VIU2 and VFPU each have four-stage pipelines. As many as ten AltiVec instructions can be executing concurrently. In the MPC7448, a maximum of two AltiVec instructions can be issued out-of-order to any combination of AltiVec execution units per clock cycle from the bottom two VIQ entries (VIQ1–VIQ0). This means an instruction in VIQ1 destined for VIU1 does not have to wait for an instruction in VIQ0 that is stalled behind an instruction waiting for operand availability.

Note that for the MPC7450, double- and single-precision versions of floating-point instructions have the same latency. For example, a floating-point multiply-add instruction takes 5 cycles to execute, regardless of whether it is single (**fmadds**) or double precision (**fmadd**).

The MPC7450 has independent on-chip, 32-Kbyte, eight-way set-associative, physically addressed L1 (level-one) caches for instructions and data, and independent instruction and data memory management units (MMUs). Each MMU has a 128-entry, two-way set-associative translation lookaside buffer (DTLB and ITLB) that saves recently used page address translations. Block address translation is implemented with the four-entry instruction and data block address translation (IBAT and DBAT) arrays defined by the PowerPC architecture. During block translation, effective addresses are compared simultaneously with all BAT entries, as described in Chapter 5, “Memory Management.” For information about the L1 caches, see Chapter 3, “L1, L2, and L3 Cache Operation.”

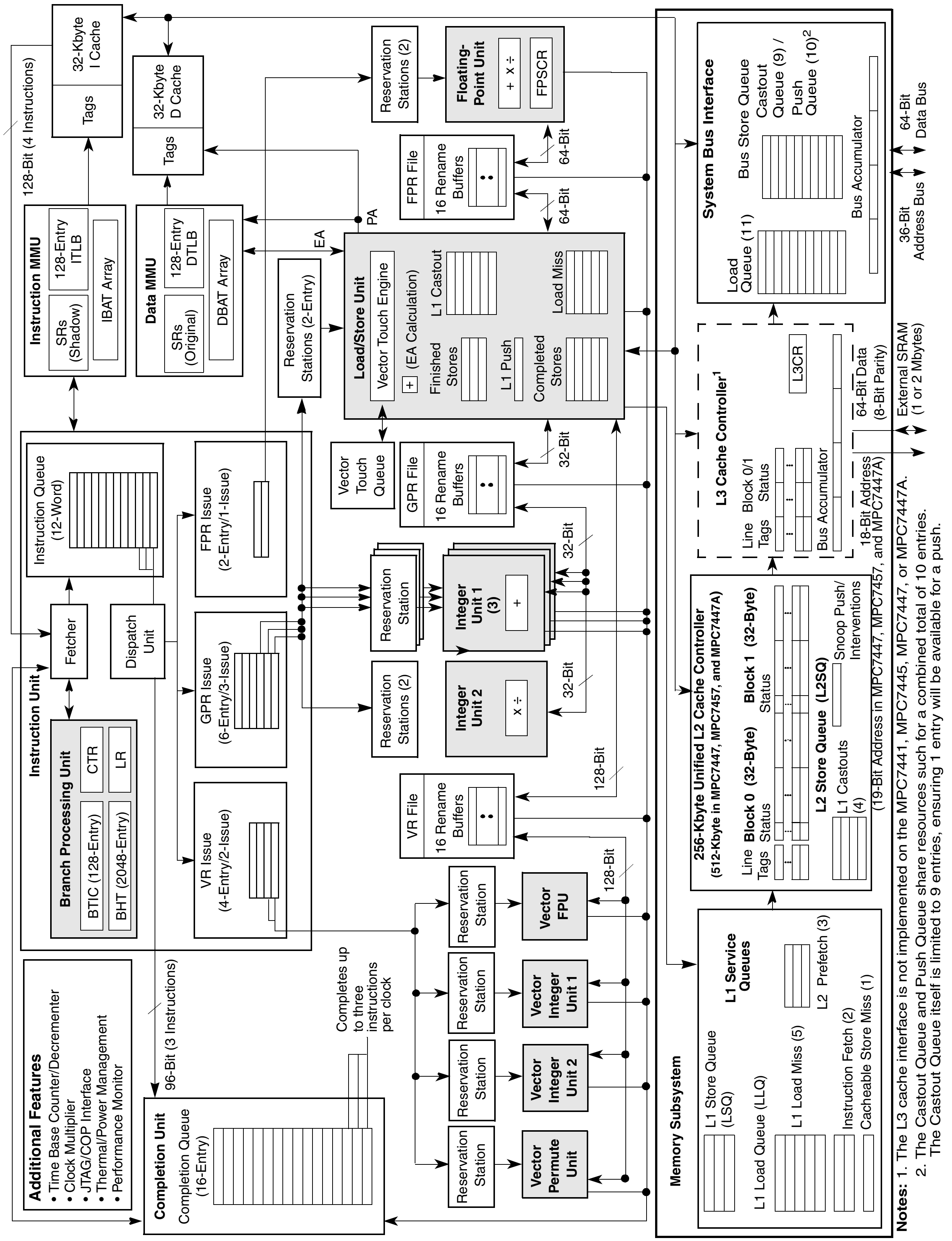
The MPC7450 L2 cache is implemented with an on-chip, 256-Kbyte, eight-way set-associative physically addressed memory available for storing data, instructions, or both. In the MPC7447, MPC7457, and MPC7447A the L2 cache is 512 Kbytes. In the MPC7448, the L2 cache is 1 Mbyte. The L2 cache supports parity generation and checking for both tags and data. It responds with a 9-cycle load latency for an L1 miss that hits in L2. In the MPC7448, the L2 load access time is 11 cycles with ECC disabled and 12 cycles with ECC enabled. The L2 cache is fully pipelined for single-cycle throughput in the MPC7450 (2-cycle throughput in the MPC7448). For information about the L2 cache implementation, see Chapter 3, “L1, L2, and L3 Cache Operation.”

The L3 cache is implemented with an on-chip, eight-way set-associative tag memory, and with external, synchronous SRAMs for storing data, instructions, or both. The external SRAMs are accessed through a dedicated L3 cache port that supports a single bank of 1 or 2 Mbytes of synchronous SRAMs for L3 cache data. The L3 data bus is 64-bits wide and provides multiple SRAM options as well as quick quad-word forwarding to reduce latency. Alternately, the L3 interface can be configured to use half or all of the SRAM area as a direct-mapped, private memory space. For information about the L3 cache implementation, see Chapter 3, “L1, L2, and L3 Cache Operation.” Note that the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448 do not support the L3 cache or L3 cache interface.

The MPC7450 has three power-saving modes, nap, sleep, and deep sleep, which progressively reduce power dissipation. When functional units are idle, a dynamic power management mode causes those units to enter a low-power mode automatically without affecting operational performance, software execution, or external hardware. Section 1.2.10, “Power and Thermal Management,” describes how the power management can be used to reduce power consumption when the processor, or portions of it, are idle. It also describes how the instruction cache throttling mechanism reduces the instruction dispatch rate. The information in these sections are described more fully in Chapter 10, “Power and Thermal Management.”

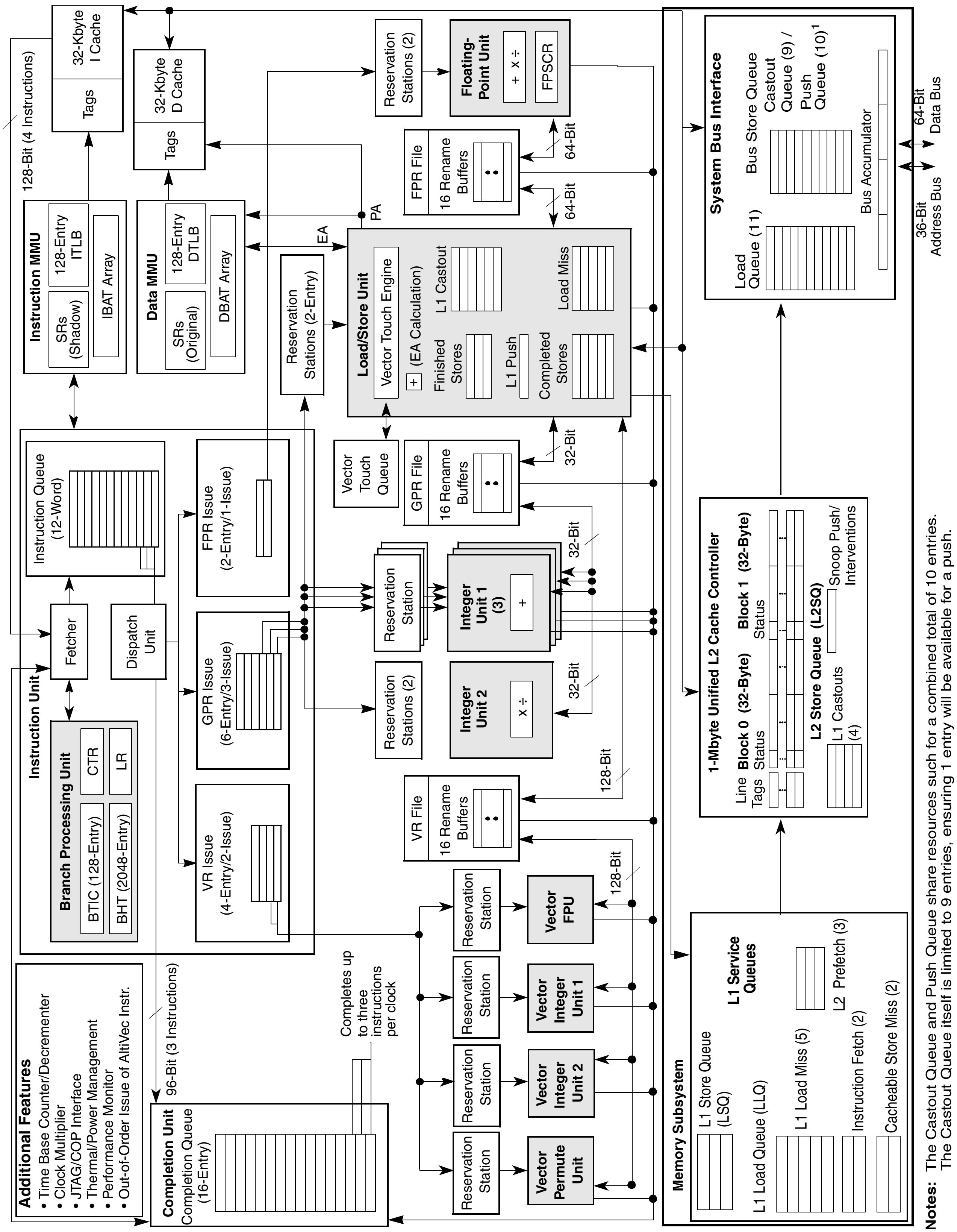
The performance monitor facility provides the ability to monitor and count predefined events such as processor clocks, misses in the instruction cache, data cache, or L2 cache, types of instructions dispatched, mispredicted branches, and other occurrences. The count of such events (which may be an approximation) can be used to trigger the performance monitor exception. Section 1.2.11, “Performance Monitor,” describes the operation of the performance monitor diagnostic tool. This functionality is fully described in Chapter 11, “Performance Monitor.”

Figure 1-1 shows the parallel organization of the execution units (shaded in the diagram) and the instruction unit fetches, dispatches, and predicts branch instructions. Note that this is a conceptual model showing basic features rather than an attempt to show how features are implemented physically. Figure 1-2 shows the organization of the MPC7448 execution units.



Notes: 1. The L3 cache interface is not implemented on the MPC7441, MPC7445, MPC7447, or MPC7447A.
2. The Castout Queue and Push Queue share resources such for a combined total of 10 entries.
The Castout Queue itself is limited to 9 entries, ensuring 1 entry will be available for a push.

Figure 1-1. MPC7450 Microprocessor Block Diagram



Notes: The Castout Queue and Push Queue share resources such for a combined total of 10 entries. The Castout Queue itself is limited to 9 entries, ensuring 1 entry will be available for a push.

Figure 1-2. MPC7448 Microprocessor Block Diagram

1.1.1 MPC7451 Microprocessor Overview

The functionality between the MPC7451 and the MPC7450 is the same. This manual describes the functionality of the MPC7450, and any differences in data regarding bus timing, signal behavior, and AC, DC, and thermal characteristics can be found in the hardware specifications.

1.1.2 MPC7441 Microprocessor Overview

The MPC7441 is a lower-pin-count device that operates identically to the MPC7451, except that it does not support the L3 cache and the L3 cache interface. This manual also describes the functionality of the MPC7441. All information herein applies to the MPC7441, except where otherwise noted (in particular, the L3 cache information does not apply to the MPC7441).

1.1.3 MPC7455 Microprocessor Overview

The MPC7455 operates similarly to the MPC7451. However, the following changes are visible to the programmer or system designer. These changes include:

- Four additional IBAT and four additional DBAT registers
- Additional HID0 bits (HID0[HIGH_BAT_EN] and HID0[XBSEN]
- Four additional SPRG registers

The additional IBATs and DBATs provide mapping for more regions of memory. For more information on new features, see Section 5.3, “Block Address Translation.”

The SPRGs provide additional registers to be used by system software for table software searching. If the SPRGs are not used for software table searches, they can be used by other supervisor programs.

1.1.4 MPC7445 Microprocessor Overview

The MPC7445 is a lower-pin-count device that operates identically to the MPC7455, except that it does not support the L3 cache and the L3 cache interface. This manual also describes the functionality of the MPC7445. All information herein applies to the MPC7445, except where otherwise noted (in particular, the L3 cache information does not apply to the MPC7445).

1.1.5 MPC7457 Microprocessor Overview

The MPC7457 operates similarly to the MPC7455. However, the following changes are visible to the programmer or system designer. These changes include:

- Larger L2 cache (512 Kbytes)
- Additional support for L3 private memory size (4 Mbytes)
- An additional L3_ADDR signal (L3_ADDR[18])
- Modifications to bits in the L3 control register (L3CR)

All information that applies to the MPC7455 also complies to the MPC7457, except where otherwise noted (in particular, the increased L2 cache and the additional L3 cache support is new for the MPC7457).

1.1.6 MPC7447 Microprocessor Overview

The MPC7447 is a lower-pin-count device that operates identically to the MPC7457, except that it does not support the L3 cache and the L3 cache interface. This manual also describes the functionality of the MPC7447. All information herein applies to the MPC7447, except where otherwise noted (in particular, the L3 cache information does not apply to the MPC7447).

1.1.7 MPC7447A Microprocessor Overview

There are no micro-architectural differences between the MPC7447A and the MPC7447. The MPC7447A provides new functionality to reduce the power consumption on the microprocessor. The following features were also added to the MPC7447A:

- Additional bits to the HID1 register for dynamic frequency switching (DFS)
- Temperature diode

Other than the new features, the MPC7447A supports the same functionality as the MPC7447.

1.1.8 MPC7448 Microprocessor Overview

The MPC7448 operates similarly to the MPC7447A. However, the MPC7448 has a number of changes over the core in the MPC7447A. Some of these changes are feature improvements and some are performance changes: improvements or changes necessary for feature improvements. The following changes were added to the MPC7448:

- Larger L2 cache (1 Mbyte)
- L2 data error correction code (ECC)
- Extended L2 pipeline
- Expanded DFS capability (DFS2 and DFS4 mode)
- Out-of-order issue of AltiVec instructions
- Second cacheable store miss
- Additional bits to the HID1 register for dynamic frequency switching (DFS) and PLL configuration
- Signals with new functionality: $\overline{\text{DFS2}}$, $\overline{\text{DFS4}}$, PLL_CFG[5], BVSEL[1], and $\overline{\text{LVRAM}}$

This manual also describes the functionality of the MPC7448. All information herein applies to the MPC7448, except where otherwise noted (in particular, the L3 cache information does not apply to the MPC7448, which does not support the L3 cache or the L3 cache interface).

1.2 MPC7450 Microprocessor Features

This section describes the features of the MPC7450. The interrelationships of these features are shown in Figure 1-1.

1.2.1 Overview of the MPC7450 Microprocessor Features

Major features of the MPC7450 are as follows:

- High-performance, superscalar microprocessor
    - As many as 4 instructions can be fetched from the instruction cache at a time
    - As many as 3 instructions can be dispatched to the issue queues at a time
    - As many as 12 instructions can be in the instruction queue (IQ)
    - As many as 16 instructions can be at some stage of execution simultaneously
    - Single-cycle execution for most instructions
    - One-instruction throughput per clock cycle for most instructions
    - Seven-stage pipeline control
- Eleven independent execution units and three register files
    - Branch processing unit (BPU) features static and dynamic branch prediction
        - 128-entry (32-set, four-way set-associative) branch target instruction cache (BTIC), a cache of branch instructions that have been encountered in branch/loop code sequences. If a target instruction is in the BTIC, it is fetched into the instruction queue a cycle sooner than it can be made available from the instruction cache. Typically, a fetch that hits the BTIC provides the first 4 instructions in the target stream.
        - 2048-entry branch history table (BHT) with 2 bits per entry for four levels of prediction—not-taken, strongly not-taken, taken, strongly taken
        - Up to three outstanding speculative branches
        - Branch instructions that do not update the count register (CTR) or link register (LR) are often removed from the instruction stream.
        - Eight-entry link register stack to predict the target address of Branch Conditional to Link Register (**bclr**) instructions
    - Four integer units (IUs) that share 32 GPRs for integer operands
        - Three identical IUs (IU1a, IU1b, and IU1) can execute all integer instructions except multiply, divide, and move to/from special-purpose register instructions.
        - IU2 executes miscellaneous instructions including the CR logical operations, integer multiplication and division instructions, and move to/from special-purpose register instructions.
    - 64-bit floating-point unit (FPU)
        - Five-stage FPU
        - Fully IEEE 754-1985 compliant FPU for both single- and double-precision operations
        - Supports non-IEEE mode for time-critical operations

    - Hardware support for denormalized numbers
    - Thirty-two 64-bit FPRs for single- or double-precision operands
  - Four vector units and 32-entry vector register file (VRs)
    - Vector permute unit (VPU)
    - Vector integer unit 1 (VIU1) handles short-latency AltiVec integer instructions, such as vector add instructions (for example, **vaddsbs**, **vaddshs**, and **vaddsws**)
    - Vector integer unit 2 (VIU2) handles longer-latency AltiVec integer instructions, such as vector multiply add instructions (for example, **vmhaddshs**, **vmhraddshs**, and **vmladduhm**).
    - Vector floating-point unit (VFPU)
  - Three-stage load/store unit (LSU)
    - Supports integer, floating-point and vector instruction load/store traffic
    - Four-entry vector touch queue (VTQ) supports all four architected AltiVec data stream operations
    - Three-cycle GPR and AltiVec load latency (byte, half word, word, vector) with single-cycle throughput
    - Four-cycle FPR load latency (single, double) with single-cycle throughput
    - No additional delay for misaligned access within double-word boundary
    - Dedicated adder calculates effective addresses (EAs)
    - Supports store gathering
    - Performs alignment, normalization, and precision conversion for floating-point data
    - Executes cache control and TLB instructions
    - Performs alignment, zero padding, and sign extension for integer data
    - Supports hits under misses (multiple outstanding misses)
    - Supports both big- and little-endian modes, including misaligned little-endian accesses
- Three issue queues, FIQ (floating-point issue queue), VIQ (vector issue queue), and GIQ (general purpose issue queue), can accept as many as one, two, and three instructions, respectively, in a cycle. Instruction dispatch requires the following:
  - Instructions can be dispatched only from the three lowest IQ entries—IQ0, IQ1, and IQ2.
  - A maximum of three instructions can be dispatched to the issue queues per clock cycle.
  - Space must be available in the completion queue (CQ) for an instruction to dispatch (this includes instructions that are assigned a space in the CQ but not in an issue queue).
- Rename buffers
  - 16 GPR (general purpose register) rename buffers
  - 16 FPR (floating-point register) rename buffers
  - 16 VR (vector register) rename buffers
- Dispatch unit—The decode/dispatch stage fully decodes each instruction.

- Completion unit
  - The completion unit retires an instruction from the 16-entry CQ when all instructions ahead of it have been completed, the instruction has finished execution, and no exceptions are pending.
  - Guarantees sequential programming model (precise exception model)
  - Monitors all dispatched instructions and retires them in order
  - Tracks unresolved branches and flushes instructions after a mispredicted branch
  - Retires as many as three instructions per clock cycle
- L1 cache has the following characteristics:
  - Two separate 32-Kbyte instruction and data caches (Harvard architecture)
  - Instruction and data caches are eight-way set-associative
  - Instruction and data caches have 32-byte cache blocks. A cache block is the block of memory that a coherency state describes—it corresponds to a cache line for the L1 data cache.
  - Cache directories are physically addressed. The physical (real) address tag is stored in the cache directory.
  - The caches implement a pseudo least-recently-used (PLRU) replacement algorithm within each way.
  - Cache write-back or write-through operation is programmable on a per-page or per-block basis.
  - Instruction cache can provide four instructions per clock cycle; data cache can provide four words per clock cycle
    - Two-cycle latency and single-cycle throughput for instruction or data cache accesses
  - Caches can be disabled in software
  - Caches can be locked in software
  - Supports a four-state modified/exclusive/shared/invalid (MESI) coherency protocol
    - A single coherency status bit for each instruction cache block allows encoding for the following two possible states:

      Invalid (INV)
      Valid (VAL)
    - Two status bits (MESI[0–1]) for each data cache block allow encoding for coherency, as follows:

      00 = invalid (I)
      01 = shared (S)
      10 = exclusive (E)
      11 = modified (M)
  - Separate copy of data cache tags for efficient snooping
  - Both L1 caches support parity generation and checking (enabled through bits in the ICTRL register) as follows:
    - Instruction cache—one parity bit per instruction
    - Data cache—one parity bit per byte of data
  - No snooping of instruction cache except for **icbi** instruction

  - Caches implement a pseudo least-recently-used (PLRU) replacement algorithm within each way
  - Data cache supports AltiVec LRU and transient instructions, as described in Section 1.3.2.2, "AltiVec Instruction Set"
  - Critical double- and/or quad-word forwarding is performed as needed. Critical quad-word forwarding is used for AltiVec loads and instruction fetches. Other accesses use critical double-word forwarding.
- On-chip level 2 (L2) cache has the following features:
  - Integrated 256-Kbyte, eight-way set-associative unified instruction and data cache (512-Kbyte for the MPC7457, MPC7447, and MPC7447A, 1-Mbyte for the MPC7448).
  - Fully pipelined to provide 32 bytes per clock cycle to the L1 caches.
  - Total latency of 9 processor cycles for L1 data cache miss that hits in the L2. In the MPC7448, total latency of 11 processor cycles for L1 data cache miss that hits in the L2 with ECC disabled, 12 cycles when ECC is enabled
  - Uses one of two random replacement algorithms (selectable through L2CR)
  - Cache write-back or write-through operation programmable on a per-page or per-block basis
  - Organized as 32 bytes/block and 2 blocks (sectors)/line (a cache block is the block of memory that a coherency state describes).
  - In the MPC7448, supports error correction and detection using a SECDED (single-error correction, double-error detection) protocol. Every 64 bits of data comes with 8 bits of error detection/correction, which can be programmed as ECC across the 64 bits of data, byte parity, or no error detection/correction.
  - Supports parity generation and checking for both tags and data (enabled through L2CR). In the MPC7448, tag parity is enabled separately in the L2ERRDIS register, and data parity can be enabled through L2CR only when ECC is disabled.
  - In the MPC7448, error injection modes provided for testing
- Level 3 (L3) cache interface (not supported on the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448)
  - Provides critical double-word forwarding to the requesting unit
  - On-chip tags support 1 or 2 Mbytes of external SRAM that is eight-way set-associative
  - Maintains instructions, data, or both instructions and data (selectable through L3CR)
  - Cache write-back or write-through operation programmable on a per-page or per-block basis
  - Organized as 64 bytes/line configured as 2 blocks (sectors) with separate status bits per line for 1-Mbyte configuration.
  - Organized as 128 bytes/line configured as 4 blocks (sectors) with separate status bits per line for 2-Mbyte configuration.
  - 1, 2, or 4 Mbytes (4 Mbytes is only for the MPC7457) of the L3 SRAM can be designated as private memory.
  - Supports same four-state (MESI) coherency protocol as L1 and L2 caches
  - Supports parity generation and checking for both tags and data (enabled through L3CR)

  - Same choice of two random replacement algorithms used by L2 cache (selectable through L3CR)
  - Configurable core-to-L3 frequency divisors
  - 64-bit external L3 data bus sustains 64 bits per L3 clock cycle
  - Supports MSUG2 dual data rate (DDR) synchronous burst SRAMs, PB2 pipelined synchronous burst SRAMs, and pipelined (register-register) late-write synchronous burst SRAMs
- Separate memory management units (MMUs) for instructions and data
  - 52-bit virtual address; 32- or 36-bit physical address
  - Address translation for 4-Kbyte pages, variable-sized blocks, and 256-Mbyte segments
  - Memory programmable as write-back/write-through, caching-inhibited/caching-allowed, and memory coherency enforced/memory coherency not enforced on a page or block basis
  - Separate IBATs and DBATs (four each) also defined as SPRs. Eight IBATs and eight DBATS in the MPC7455, MPC7445, MPC7457, MPC7447, MPC7447A, and MPC7448.
  - Separate instruction and data translation lookaside buffers (TLBs)
    - Both TLBs are 128-entry, two-way set-associative, and use LRU replacement algorithm
    - TLBs are hardware or software reloadable (that is, on a TLB miss a page table search is performed in hardware or by system software)
- Efficient data flow
  - Although the VR/LSU interface is 128 bits, the L1/L2/L3 bus interface allows up to 256 bits.
  - The L1 data cache is fully pipelined to provide 128 bits/cycle to or from the VRs.
  - L2 cache is fully pipelined to provide 32 bytes per processor clock cycle to the L1 cache. In the MPC7448, the L2 cache is pipelined to provide 32 bytes every other clock cycle to the L1 cache.
  - As many as eight outstanding, out-of-order cache misses are allowed between the L1 data cache and L2/L3 bus.
  - As many as 16 out-of-order transactions can be present on the MPX bus.
  - Store merging for multiple store misses to the same line. Only coherency action taken (address-only) for store misses merged to all 32 bytes of a cache block (no data tenure needed)
  - Support for a second cacheable store miss
  - Three-entry finished store queue and five-entry completed store queue between the LSU and the L1 data cache
  - Separate additional queues for efficient buffering of outbound data (such as castouts and write-through stores) from the L1 data cache and L2 cache
- Multiprocessing support features include the following:
  - Hardware-enforced, MESI cache coherency protocols for data cache
  - Load/store with reservation instruction pair for atomic memory references, semaphores, and other multiprocessor operations

- Power and thermal management
  - — The following three power-saving modes are available to the system:
    - – Nap—Instruction fetching is halted. Only those clocks for the time base, decrementer, and JTAG logic remain running. The part goes into the doze state to snoop memory operations on the bus and then back to nap using a $\overline{\text{QREQ}}$/$\overline{\text{QACK}}$ processor-system handshake protocol.
    - – Sleep—Power consumption is further reduced by disabling bus snooping, leaving only the PLL in a locked and running state. All internal functional units are disabled.
    - – Deep sleep—When the part is in the sleep state, the system can disable the PLL. The system can then disable the SYSCLK source for greater system power savings. Power-on reset procedures for restarting and relocking the PLL must be followed upon exiting the deep sleep state.
  - — In the MPC7447A and MPC7448, DFS (dynamic frequency switching) conserves power by lowering processor operating frequency. The MPC7447A has the ability to divide the processor-to-system bus ratio by two during normal functional operation. The MPC7448 has the additional ability to divide by four.
  - — Instruction cache throttling provides control of instruction fetching to limit device temperature.
- Performance monitor can be used to help debug system designs and improve software efficiency
- In-system testability and debugging features through JTAG boundary-scan capability
- Reliability and serviceability
  - — Parity checking on system bus and L3 cache bus
  - — Parity checking on L1, L2, and L3 cache arrays

1.2.2 Instruction Flow

As shown in Figure 1-1, the MPC7450 instruction unit provides centralized control of instruction flow to the execution units. The instruction unit contains a sequential fetcher, 12-entry instruction queue (IQ), dispatch unit, and branch processing unit (BPU). It determines the address of the next instruction to be fetched based on information from the sequential fetcher and from the BPU.

See Chapter 6, “Instruction Timing,” for a detailed discussion of instruction timing.

The sequential fetcher loads instructions from the instruction cache into the instruction queue. The BPU extracts branch instructions from the sequential fetcher. Branch instructions that cannot be resolved immediately are predicted using either the MPC7450-specific dynamic branch prediction or the architecture-defined static branch prediction.

Branch instructions that do not affect the LR or CTR are often removed from the instruction stream. Section 6.4.1.1, “Branch Folding and Removal of Fall-Through Branch Instructions,” describes when a branch can be removed from the instruction stream.

Instructions dispatched beyond a predicted branch do not complete execution until the branch is resolved, preserving the programming model of sequential execution. If branch prediction is incorrect, the instruction unit flushes all predicted path instructions, and instructions are fetched from the correct path.

1.2.2.1 Instruction Queue and Dispatch Unit

The instruction queue (IQ), shown in Figure 1-1, holds as many as 12 instructions and loads as many as 4 instructions from the instruction cache during a single processor clock cycle.

The fetcher attempts to initiate a new fetch every cycle. The two fetch stages are pipelined, so as many as four instructions can arrive to the IQ every cycle. All instructions except branch (**b***x*), Return from Exception (**rfi**), System Call (**sc**), Instruction Synchronize (**isync**), and no-op instructions are dispatched to their respective issue queues from the bottom three positions in the instruction queue (IQ0–IQ2) at a maximum rate of three instructions per clock cycle. Reservation stations are provided for the three IU1s, IU2, FPU, LSU, VPU, VIU2, VIU1, and VFPU. The dispatch unit checks for source and destination register dependencies, determines whether a position is available in the CQ, and inhibits subsequent instruction dispatching as required.

Branch instruction can be detected, decoded, and predicted from entries IQ0–IQ7. See Section 6.3.3, “Dispatch, Issue, and Completion Considerations.”

1.2.2.2 Branch Processing Unit (BPU)

The BPU receives branch instructions from the IQ and executes them early in the pipeline, achieving the effect of a zero-cycle branch in some cases.

Branches with no outstanding dependencies (CR, LR, or CTR unresolved) can be processed and resolved immediately. For branches in which only the direction is unresolved due to a CR or CTR dependency, the branch path is predicted using either architecture-defined static branch prediction or MPC7450-specific dynamic branch prediction. Dynamic branch prediction is enabled if HID0[BHT] is set. For **bclr** branches where the target address is unresolved due to a LR dependency, the branch target can be predicted using the hardware link stack. Link stack prediction is enabled if HID0[LRSTK] is set.

When a prediction is made, instruction fetching, dispatching, and execution continue from the predicted path, but instructions cannot complete and write back results to architected registers until the prediction is determined to be correct (resolved). When a prediction is incorrect, the instructions from the incorrect path are flushed from the processor and processing begins from the correct path.

Dynamic prediction is implemented using a 2048-entry branch history table (BHT), a cache that provides two bits per entry that together indicate four levels of prediction for a branch instruction—not-taken, strongly not-taken, taken, strongly taken. When dynamic branch prediction is disabled, the BPU uses a bit in the instruction encoding to predict the direction of the conditional branch. Therefore, when an unresolved conditional branch instruction is encountered, the MPC7450 executes instructions from the predicted target stream although the results are not committed to architected registers until the conditional branch is resolved. Unresolved branches are held in a three-entry branch queue. When the branch queue is full, no further conditional branches can be processed until one of the conditions in the branch queue is resolved.

When a branch is taken or predicted as taken, instructions from the untaken path must be flushed and the target instruction stream must be fetched into the IQ. The BTIC is a 128-entry, four-way set associative cache that contains the most recently used branch target instructions (up to four instructions per entry) for **b** and **bc** branches. When a taken branch instruction of this type hits in the BTIC, the instructions arrive in the instruction queue 2 clock cycles later, a clock cycle sooner than they would arrive from the instruction cache. Additional instructions arrive from the instruction cache in the next clock cycle. The BTIC reduces the number of missed opportunities to dispatch instructions and gives the processor a 1-cycle head start on processing the target stream.

The BPU contains an adder to compute branch target addresses and three user-accessible registers—the link register (LR), the count register (CTR), and the condition register (CR). The BPU calculates the return pointer for subroutine calls and saves it in the LR for certain types of branch instructions. The LR also contains the branch target address for Branch Conditional to Link Register (**bclr***x*) instructions. The CTR contains the branch target address for Branch Conditional to Count Register (**bcctr***x*) instructions. Because the LR and CTR are SPRs, their contents can be copied to or from any GPR. Also, 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.

1.2.2.3 Completion Unit

The completion unit operates closely with the instruction unit. Instructions are fetched and dispatched in program order. At the point of dispatch, the program order is maintained by assigning each dispatched instruction a successive entry in the 16-entry CQ. The completion unit tracks instructions from dispatch through execution and retires them in program order from the three bottom CQ entries (CQ0–CQ2).

Instructions cannot be dispatched to an execution unit unless there is a CQ vacancy.

Branch instructions that do not update the CTR or LR are often removed from the instruction stream. Those that are removed do not take a CQ entry. Branches that are not removed from the instruction stream follow the same dispatch and completion procedures as non-branch instructions but are not dispatched to an issue queue.

Completing an instruction commits execution results to architected registers (GPRs, FPRs, VRs, LR, and CTR). In-order completion ensures the correct architectural state when the MPC7450 must recover from a mispredicted branch or any exception. An instruction is retired as it is removed from the CQ.

For a more detailed discussion of instruction completion, see Section 6.3.3, “Dispatch, Issue, and Completion Considerations.”

1.2.2.4 Independent Execution Units

In addition to the BPU, the MPC7450 provides the ten execution units described in the following sections.

1.2.2.4.1 AltiVec Vector Permute Unit (VPU)

The VPU executes permutation instructions such as pack, unpack, merge, splat, and permute on vector operands.

1.2.2.4.2 AltiVec Vector Integer Unit 1 (VIU1)

The VIU1 executes simple vector integer computational instructions, such as addition, subtraction, maximum and minimum comparisons, averaging, rotation, shifting, comparisons, and boolean operations.

1.2.2.4.3 AltiVec Vector Integer Unit 2 (VIU2)

The VIU2 executes longer-latency vector integer instructions, such as multiplication, multiplication/addition, and sum-across with saturation.

1.2.2.4.4 AltiVec Vector Floating-Point Unit (VFPU)

The VFPU executes all vector floating-point instructions.

A maximum of two AltiVec instructions can be issued in order to any combination of AltiVec execution units per clock cycle. In the MPC7448, a maximum of two AltiVec instructions can be issued out of order to any combination of AltiVec execution units per clock cycle from the bottom two VIQ entries (VIQ1–VIQ0). An instruction in VIQ1 destined for VIU1 does not have to wait for an instruction in VIQ0 that is stalled behind an instruction waiting for operand availability. Moreover, the VIU2, VFPU, and VPU are pipelined, so they can operate on multiple instructions.

1.2.2.4.5 Integer Units (IUs)

The integer units (three IU1s and IU2) are shown in Figure 1-1. The IU1s execute shorter latency integer instructions, that is, all integer instructions except multiply, divide, and move to/from special-purpose register instructions. IU2 executes integer instructions with latencies of 3 cycles or more.

IU2 has a 32-bit integer multiplier/divider and a unit for executing CR logical operations and move to/from SPR instructions. The multiplier supports early exit for operations that do not require full 32 * 32-bit multiplication.

1.2.2.4.6 Floating-Point Unit (FPU)

The FPU, shown in Figure 1-1, is designed such that double-precision operations require only a single pass, with a latency of 5 cycles. As instructions are dispatched to the FPUs reservation station, source operand data can be accessed from the FPRs or from the FPR rename buffers. Results in turn are written to the rename buffers and are made available to subsequent instructions. Instructions start execution from the bottom reservation station only and execute in program order.

The FPU contains a single-precision multiply-add array and the floating-point status and control register (FPSCR). The multiply-add array allows the MPC7450 to efficiently implement multiply and multiply-add operations. The FPU is pipelined so that one single- or double-precision instruction can be issued per clock cycle.

Note that an execution bubble occurs after four consecutive, independent floating-point arithmetic instructions execute to allow for a normalization special case. Thirty-two 64-bit floating-point registers are provided to support floating-point operations. Stalls due to contention for FPRs are minimized by automatic allocation of the 16 floating-point rename registers. The MPC7450 writes the contents of the rename registers to the appropriate FPR when floating-point instructions are retired by the completion unit.

The MPC7450 supports all IEEE 754 floating-point data types (normalized, denormalized, NaN, zero, and infinity) in hardware, eliminating the latency incurred by software exception routines.

1.2.2.4.7 Load/Store Unit (LSU)

The LSU executes all load and store instructions as well as AltiVec LRU and transient instructions and provides the data transfer interface between the GPRs, FPRs, VRs, and the cache/memory subsystem. The LSU also calculates effective addresses and aligns data.

Load and store instructions are issued and translated in program order; however, some memory accesses can occur out-of-order. Synchronizing instructions can be used to enforce strict ordering. When there are no data dependencies and the guarded bit for the page or block is cleared, a maximum of one out-of-order cacheable load operation can execute per clock cycle from the perspective of the LSU. Loads to FPRs require a 4-cycle total latency. Data returned from the cache is held in a rename register until the completion logic commits the value to a GPR, FPR, or VR. Stores cannot be executed out-of-order and are held in the store queue until the completion logic signals that the store operation is to be completed to memory. The MPC7450 executes store instructions with a maximum throughput of one per clock cycle and a 3-cycle total latency to the data cache. The time required to perform the load or store operation depends on the processor:bus clock ratio and whether the operation involves the on-chip caches, the L3 cache, system memory, or an I/O device.

1.2.3 Memory Management Units (MMUs)

The MPC7450's MMUs support up to 4 Petabytes (2^{52}) of virtual memory and 64 Gigabytes (2^{36}) of physical memory for instructions and data. The MMUs control access privileges for these spaces on block and page granularities. Referenced and changed status is maintained by the processor for each page to support demand-paged virtual memory systems. The memory management units are contained within the load/store unit.

The LSU calculates effective addresses for data loads and stores; the instruction unit calculates effective addresses for instruction fetching. The MMU translates the effective address to determine the correct physical address for the memory access.

The MPC7450 supports the following types of memory translation:

- Real addressing mode—In this mode, translation is disabled by clearing bits in the machine state register (MSR): MSR[IR] for instruction fetching or MSR[DR] for data accesses. When address translation is disabled, the physical address is identical to the effective address. When extended addressing is disabled (HID0[XAEN] = 0) a 32-bit physical address is used, PA[4–35]. For more details, see Section 5.1.3, “Address Translation Mechanisms.”
- Page address translation—translates the page frame address for a 4-Kbyte page size
- Block address translation—translates the base address for blocks: 128 Kbytes to 256 Mbytes (MPC7441, MPC7450, MPC7451) or 4 GBytes (MPC7445, MPC7455, MPC7457, MPC7447, MPC7447A, MPC7448).

If translation is enabled, the appropriate MMU translates the higher-order bits of the effective address into physical address bits. Lower-order address bits are untranslated and so are the same for both logical and physical addresses. These bits are directed to the on-chip caches where they form the index into the eight-way set-associative tag array. After translating the address, the MMU passes the higher-order physical address bits to the cache and the cache lookup completes. For caching-inhibited accesses or accesses that miss in the cache, the untranslated lower-order address bits are concatenated with the translated higher-order address bits; the resulting 32- or 36-bit physical address is used by the memory subsystem and the bus interface unit to access external memory.

The TLBs store page address translations for recent memory accesses. For each access, an effective address is presented for page and block translation simultaneously. If a translation is found in both the TLB and the BAT array, the block address translation in the BAT array is used. Usually the translation is in a TLB and the physical address is readily available to the on-chip cache. When a page address translation is not in a TLB, hardware or system software searches for one in the page table following the model defined by the PowerPC architecture.

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. The MPC7450 instruction and data TLBs are 128-entry, two-way set-associative caches that contain address translations. The MPC7450 can initiate a hardware or system software search of the page tables in memory on a TLB miss.

1.2.4 On-Chip L1 Instruction and Data Caches

The MPC7450 implements separate L1 instruction and data caches. Each cache is 32-Kbyte eight-way set-associative. As defined by the PowerPC architecture, they are physically indexed. Each cache block contains eight contiguous words from memory that are loaded from an eight-word boundary (that is, bits EA[27–31] are zeros); thus, a cache block never crosses a page boundary. An entire cache block can be updated by a four-beat burst load across a 64-bit system bus. Misaligned accesses across a page boundary can incur a performance penalty. The data cache is a nonblocking, write-back cache with hardware support for reloading on cache misses. The critical double word is transferred on the first beat and is forwarded to the requesting unit, minimizing stalls due to load delays. For vector loads, the critical quad word is handled similarly but is transferred on the second beat. The cache being loaded is not blocked to internal accesses while the load completes.

The MPC7450 L1 cache organization is shown in Figure 1-3.

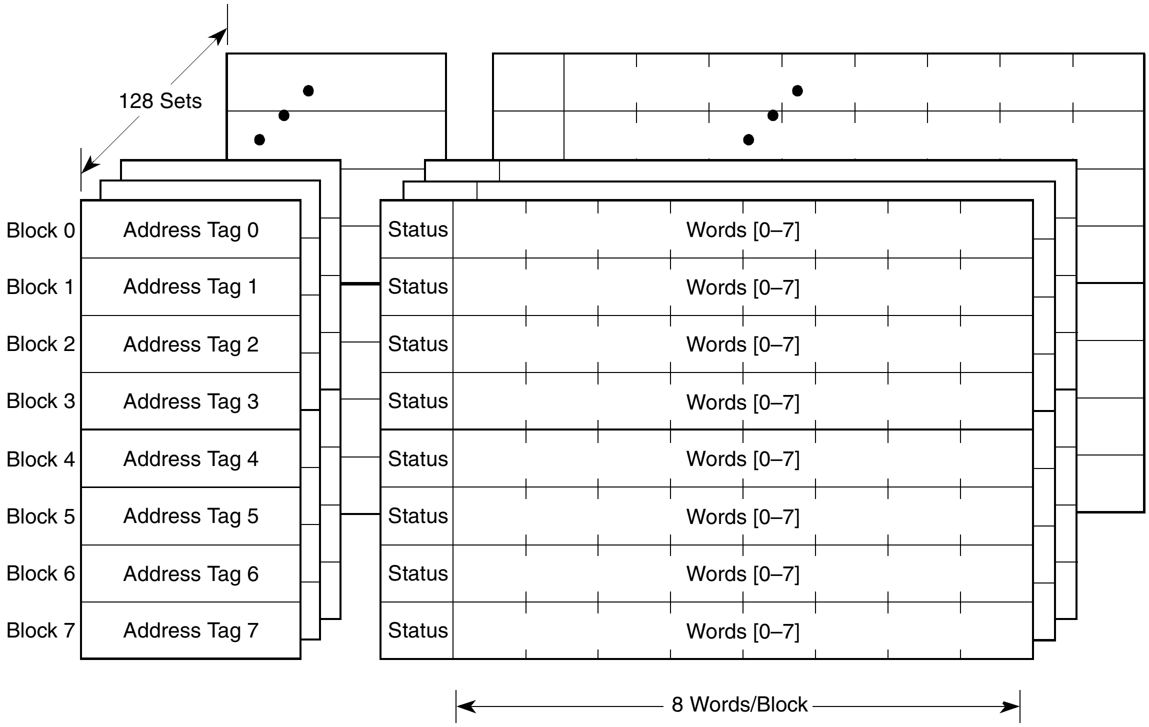


Figure 1-3. L1 Cache Organization

The instruction cache provides up to four instructions per clock cycle to the instruction queue. The instruction cache can be invalidated entirely or on a cache-block basis. It is invalidated and disabled by setting HID0[ICFI] and then clearing HID0[ICE]. The instruction cache can be locked by setting HID0[ILOCK]. The instruction cache supports only the valid/invalid states.

The data cache provides four words per clock cycle to the LSU. Like the instruction cache, the data cache can be invalidated all at once or on a per-cache-block basis. The data cache can be invalidated and disabled by setting HID0[DCFI] and then clearing HID0[DCE]. The data cache can be locked by setting HID0[DLOCK]. The data cache tags are dual-ported, so a load or store can occur simultaneously with a snoop.

The MPC7450 also implements a 128-entry (32-set, four-way set-associative) branch target instruction cache (BTIC). The BTIC is a cache of branch instructions that have been encountered in branch/loop code sequences. If the target instruction is in the BTIC, it is fetched into the instruction queue a cycle sooner than it can be made available from the instruction cache. Typically, the BTIC contains the first four instructions in the target stream.

The BTIC can be disabled and invalidated through software. As with other aspects of MPC7450 instruction timing, BTIC operation is optimized for cache-line alignment. If the first target instruction is one of the first five instructions in the cache block, the BTIC entry holds four instructions. If the first target instruction is the last instruction before the cache block boundary, it is the only instruction in the corresponding BTIC entry. If the next-to-last instruction in a cache block is the target, the BTIC entry holds two valid target instructions, as shown in Figure 1-4.

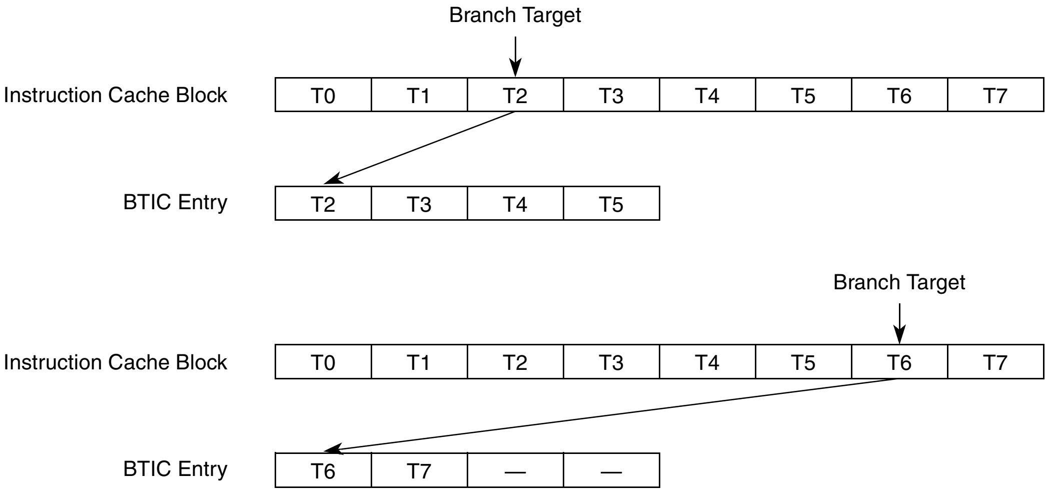


Figure 1-4. Alignment of Target Instructions in the BTIC

BTIC ways are updated using a FIFO algorithm.

For more information and timing examples showing cache hit and cache miss latencies, see Section 6.3.2, “Instruction Fetch Timing.”

1.2.5 L2 Cache Implementation

The L2 cache is a unified cache that receives memory requests from both the L1 instruction and data caches independently. The integrated L2 cache on the MPC7450 is a unified (containing both instructions and data) 256-Kbyte on-chip cache. In the MPC7447, MPC7457, and MPC7447A, the L2 cache has been increased to a 512-Kbyte on-chip cache. In the MPC7448, the L2 cache is 1 Mbyte. It is eight-way set-associative and organized with 32-byte blocks and two blocks/line.

Each line consists of 64 bytes of data organized as two blocks (also called sectors). Although all 16 words in a cache line share the same address tag, each block maintains the three separate status bits for the 8 words of the cache block, the unit of memory at which coherency is maintained. Thus, each cache line can contain 16 contiguous words from memory that are read or written as 8-word operations.

The MPC7450 integrated L2 cache organization is shown in Figure 1-5.

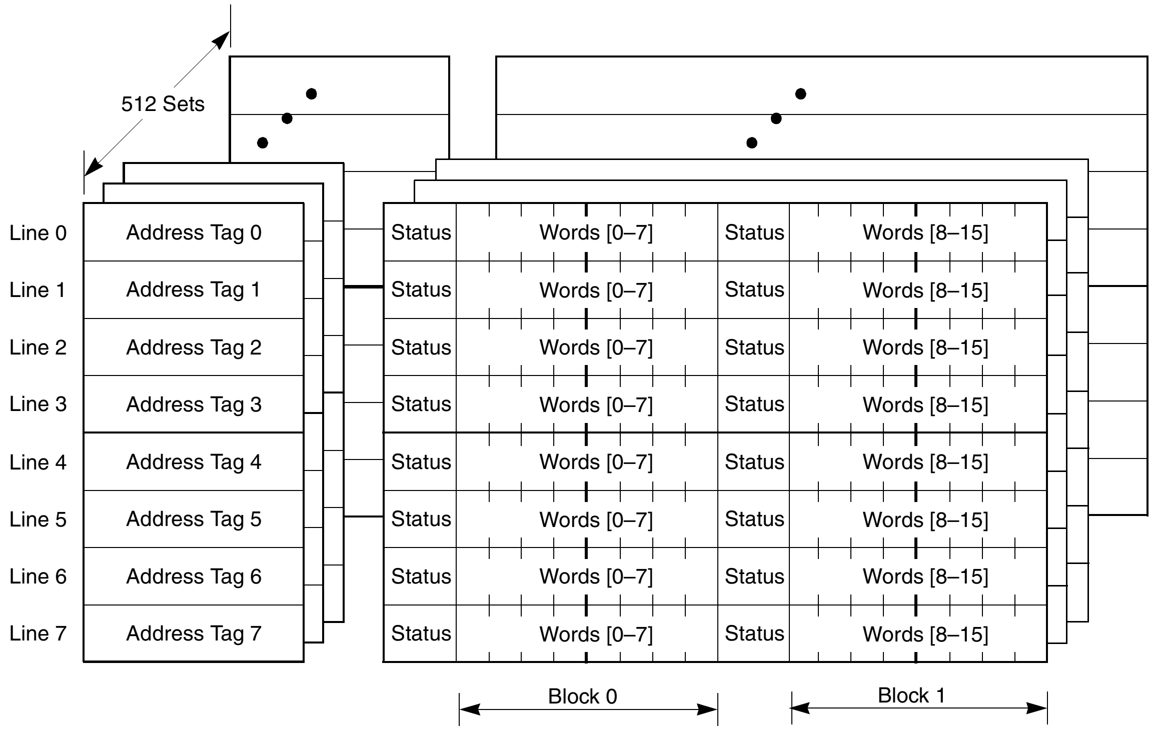


Figure 1-5. L2 Cache Organization for MPC7450

The integrated L2 cache organization of the MPC7457, MPC7447, MPC7447A, and MPC7448 is shown in Figure 1-6.

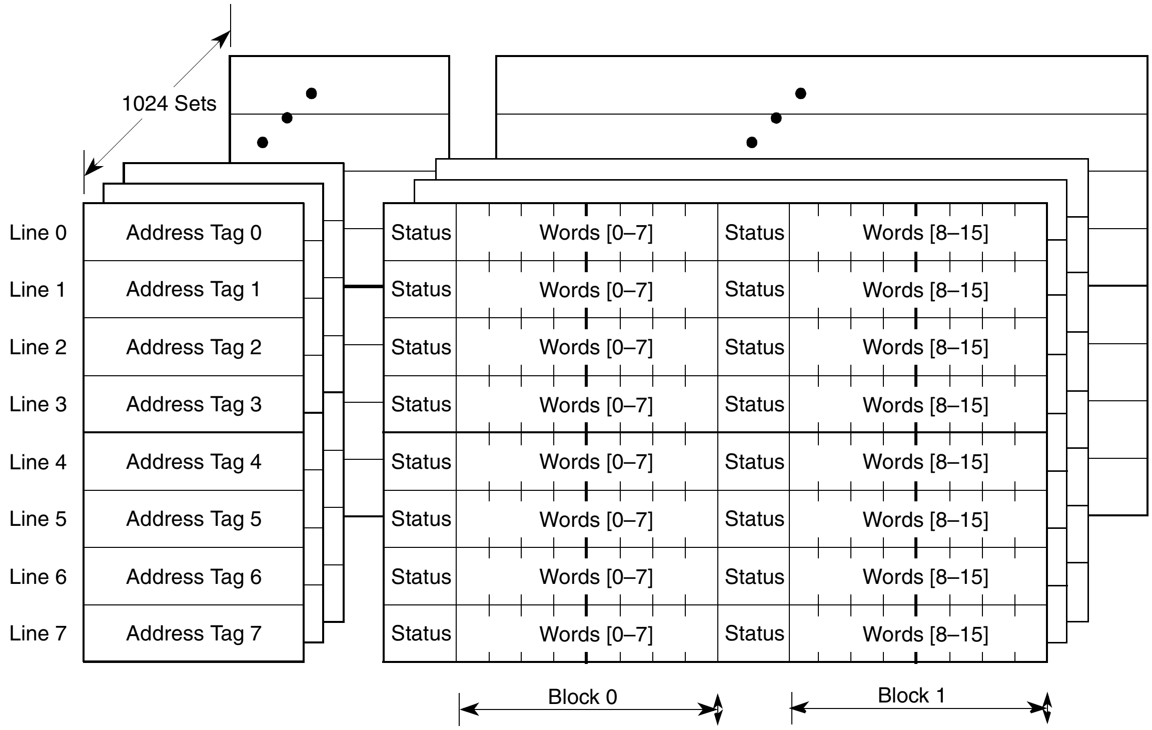


Figure 1-6. L2 Cache Organization for the MPC7457, MPC7447, and MPC7447A

The MPC7448 integrated L2 cache organization is shown in Figure 1-7.

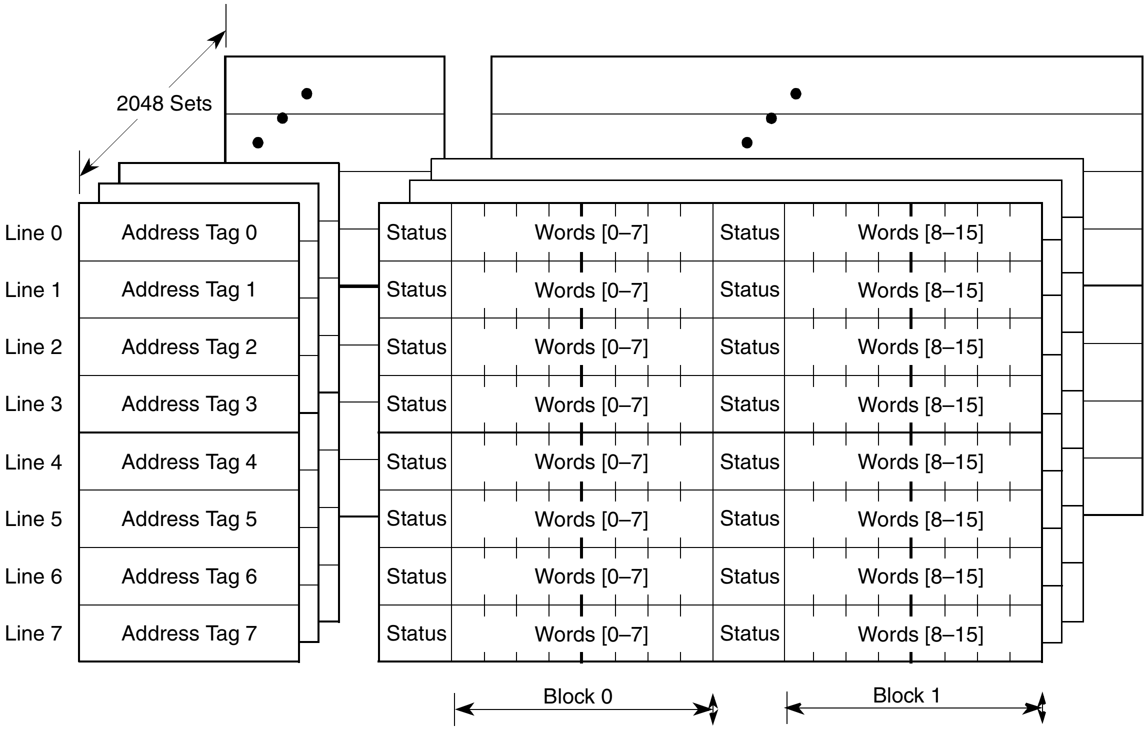


Figure 1-7. L2 Cache Organization for the MPC7448

The L2 cache controller contains the L2 cache control register (L2CR), which:

- Includes bits for enabling parity checking on the L2
- Provides for instruction-only and data-only modes
- Provides hardware flushing for the L2
- Selects between two available replacement algorithms for the L2 cache

The L2 implements the MESI cache coherency protocol using three status bits per sector.

Requests from the L1 cache generally result from instruction misses, data load or store misses, write-through operations, or cache management instructions. Requests from the L1 cache are compared against the L2 tags and serviced by the L2 cache if they hit; if they miss in the L2 cache, they are forwarded to the L3 cache.

The L2 cache tags are fully pipelined and non-blocking for efficient operation. Thus the L2 cache can be accessed internally while a load for a miss is pending (allowing hits under misses). A reload for a cache miss is treated as a normal access and blocks other accesses for only 1 cycle.

For more information, see Chapter 3, “L1, L2, and L3 Cache Operation.”

1.2.6 L3 Cache Implementation

The unified L3 cache receives memory requests from L1 and L2 instruction and data caches independently. The L3 cache interface is implemented with an on-chip, two-way set associative tag memory with 2,048

(2K) tags per way and a dedicated interface with support for up to 2 Mbytes of external synchronous SRAMs. Note that the L3 cache is not supported on the MPC7441, MPC7445, MPC7447, MPC7447A, and the MPC7448

Tags are sectored to support either two or four cache blocks per tag entry, depending on the L2 cache size. Each sector (32-byte cache block) in the L3 cache has three status bits that are used to implement the MESI cache coherency protocol. Accesses to the L3 cache can be designated as write-back or write-through, and the L3 maintains cache coherency through snooping.

The L3 interface can be configured to use 1 or 2 Mbytes of the SRAM area as a private memory space. The MPC7457 in particular can support 1,2, or 4 Mbytes of private memory. Accesses to private memory do not propagate to the system bus. The MPC7450 can also be configured to use 1 Mbyte of SRAM as L3 cache and a second Mbyte as private memory. Also, in this case, private memory accesses do not propagate to the L3 cache or the external system bus.

The private memory space provides a low-latency, high-bandwidth area for critical data or instructions. Accesses to the private memory space do not propagate to the L3 cache nor are they visible to the external system bus. The private memory space is also not snooped, so the coherency of its contents must be maintained by software or not at all. For more information, see Chapter 3, “L1, L2, and L3 Cache Operation.”

The L3 cache control register (L3CR) provides control of L3 cache configuration and interface timing. The L3 private memory control register (L3PM) configures the private memory feature.

The L3 cache interface provides two clock outputs that allow the clock inputs of the SRAMs to be driven at select frequency divisions of the processor core frequency. For the MPC7457, the L3 cache interface provides two sets of two differential clock outputs.

Requests from the L3 cache generally result from instruction misses, data load or store misses, write-through operations, or cache management instructions. Requests from the L1 and L2 cache are compared against the L3 tags and serviced by the L3 cache if they hit; if they miss in the L3 cache, they are forwarded to the bus interface. Note that the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448 do not support the L3 cache and the L3 interface.

1.2.7 System Interface

The MPC7450 supports two interface protocols—MPX bus protocol and a subset of the 60x bus protocol. Note that although this protocol is implemented by the MPC603e, MPC604e, MPC740, and MPC750 processors, it is referred to as the 60x bus interface. The MPX bus protocol is derived from the 60x bus protocol. The MPX bus interface includes several additional features that provide higher memory bandwidth than the 60x bus and more efficient use of the system bus in a multiprocessing environment. Because the MPC7450’s performance is optimized for the MPX bus, use of the MPX bus is recommended over the 60x bus.

The MPC7450 bus interface includes a 64-bit data bus with 8 bits of data parity, a 36-bit address bus with 5 bits of address parity, and additional control signals to allow for unique system level optimizations.

The bus interface protocol is configured using the $\overline{\text{BMODE0}}$ configuration signal at reset. If $\overline{\text{BMODE0}}$ is asserted at the negation of $\overline{\text{HRESET}}$, the MPC7450 uses the MPX bus protocol; if $\overline{\text{BMODE0}}$ is negated during the negation of $\overline{\text{HRESET}}$, the MPC7450 uses a limited subset of the 60x bus protocol. Note that the inverse state of $\overline{\text{BMODE}}$[0:1] at the negation of $\overline{\text{HRESET}}$ is saved in MSSCR0[BMODE].

1.2.8 MPC7450 Bus Operation Features

The MPC7450 has a separate address and data bus, each with its own set of arbitration and control signals. This allows for decoupling the data tenure from the address tenure of a transaction and provides for a wide range of system-bus implementations including:

- Non-pipelined bus operation
- Pipelined bus operation
- Split transaction operation

The MPC7450 supports only the normal memory-mapped address segments defined in the PowerPC architecture. Access to direct store segments results in a DSI exception.

1.2.8.1 MPX Bus Features

The MPX bus has the following features:

- Extended 36-bit address bus plus 5 bits of odd parity (41 bits total)
- 64-bit data bus plus 8 bits of odd parity (72 bits total); a 32-bit data bus mode is not supported
- Support for a four-state (MESI) cache coherence protocol
- On-chip snooping to maintain L1 data cache, L2, and L3 cache coherency for multiprocessing applications and DMA environments
- Support for address-only transfers (useful for a variety of broadcast operations in multiprocessor applications)
- Address pipelining
- Support for up to 16 out-of-order transactions using 4 data transaction index (DTI[0:3]) signals
- Full data streaming
- Support for data intervention in multiprocessor systems

1.2.8.2 60x Bus Features

The following list summarizes the 60x bus interface features:

- Extended 36-bit address bus plus 5 bits of odd parity (41 bits total)
- 64-bit data bus plus 8 bits of odd parity (72 bits total); a 32-bit data bus mode is not supported
- Support for a four-state (MESI) cache coherence protocol
- On-chip snooping to maintain L1 data cache, L2, and L3 cache coherency for multiprocessing applications and DMA environments
- Support for address-only transfers (useful for a variety of broadcast operations in multiprocessor applications)
- Address pipelining
- Support for up to 16 outstanding transactions. No reordering is supported.

1.2.9 Overview of System Interface Accesses

The system interface includes address register queues, prioritization logic, and a bus control unit. The system interface latches snoop addresses for snooping in the L1 data, L2, and L3 caches, the memory hierarchy address register queues, and the reservation controlled by the Load Word and Reserve Indexed (**lwarx**) and Store Word Conditional Indexed (**stwcx.**) instructions. Accesses are prioritized with load operations preceding store operations. Note that the L3 cache interface is not supported on the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448.

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

When the MPC7450 encounters an instruction or data access, it calculates the effective address and uses the lower-order address bits to check for a hit in the on-chip, 32-Kbyte L1 instruction and data caches. During L1 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 (real) address. The physical address bits are then compared with the corresponding cache tag bits to determine if a cache hit occurred in the L1 instruction or data cache. If the access misses in the corresponding cache, the transaction is sent to L1 load miss queue or the L1 store miss queue. L1 load miss queue transactions are sent to the internal 256-Kbyte L2 cache (512-Kbyte for MPC7447, MPC7457, and MPC7447A, 1-Mbyte for the MPC7448) and L3 cache controller simultaneously. Store miss queue transactions are queued up in the L2 cache controller and sent to the L3 cache if necessary. If no match is found in the L2 or L3 cache tags, the physical address is used to access system memory.

In addition to loads, stores, and instruction fetches, the MPC7450 performs hardware table search operations following TLB misses; L1, L2, and L3 cache castout operations; and cache-line snoop push operations when a modified cache line detects a snoop hit from another bus master.

1.2.9.1 System Interface Operation

The primary activity of the MPC7450 system interface is transferring data and instructions between the processor and system memory. There are three types of transfer accesses:

- Single-beat transfers—These memory accesses allow transfer sizes of 1, 2, 3, 4, or 8 bytes in one bus clock cycle. Single-beat transactions are caused by uncacheable read and write operations that access memory directly (that is, when caching is disabled), cache-inhibited accesses, and stores in write-through mode.
- Two-beat burst (16-byte) data transfers—Generated to support caching-inhibited or write-through AltiVec loads and stores (only generated in MPX bus mode) and for caching-inhibited instruction fetches in MPX mode.
- Four-beat burst (32-byte) data transfers—Initiated when an entire cache block is transferred into or out of the internal caches. Because the first-level caches on the MPC7450 are write-back caches, burst-read memory operations are the most common memory accesses, followed by burst-write memory operations, and single-beat (caching-inhibited or write-through) memory read and write operations.

Memory accesses can occur in single-beat (1, 2, 3, 4, and 8 bytes), double-beat (16 bytes), and four-beat (32 bytes) burst data transfers. For memory accesses, the address and data buses are independent to support pipelining and split transactions. The bus interface can pipeline as many as 16 transactions and, in MPX bus mode, supports full out-of-order split-bus transactions. The MPC7450 bursts out of reset in MPX bus mode, fetching eight instructions on the MPX bus at a time.

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 MPC7450 to be integrated into systems that implement various fairness and bus-parking procedures to avoid arbitration overhead.

Typically, memory accesses are weakly ordered to maximize the efficiency of the bus without sacrificing coherency of the data. The MPC7450 allows load operations to bypass store operations (except when a dependency exists). Because the processor can dynamically optimize run-time ordering of load/store traffic, overall performance is improved.

Note that the synchronize (**sync**) and enforce in-order execution of I/O (**eieio**) instructions can be used to enforce strong ordering.

The system interface is synchronous. All MPC7450 inputs are sampled and all outputs are driven on the rising edge of the bus clock cycle. The hardware specifications gives timing information. The system interface is specific for each microprocessor that implements the PowerPC architecture.

1.2.9.2 Signal Groupings

Signals are provided for implementing the bus protocol, clocking, and control of the L3 caches, as well as separate L3 address and data buses. Test and control signals provide diagnostics for selected internal circuits.

The MPC7450 MPX and 60x bus interface protocol signals are grouped as follows:

- Address arbitration—The MPC7450 uses these signals to arbitrate for address bus mastership.
- Address transfer start—These signals indicate that a bus master has begun a transaction on the address bus.
- Address transfer—These signals include the address bus and address parity signals. They are used to transfer the address and to ensure the integrity of the transfer.
- Transfer attribute—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—These signals are used to acknowledge the end of the address phase of the transaction. They also indicate whether a condition exists that requires the address phase to be repeated.
- Data arbitration—The MPC7450 uses these signals to arbitrate for data bus mastership.
- Data transfer—These signals, which consist of the data bus and data parity signals, are used to transfer the data and to ensure the integrity of the transfer.
- Data transfer termination—Data termination signals are required after each data beat in a data transfer. In a single-beat transaction, data termination signals also indicate the end of the tenure. In burst accesses, data termination signals apply to individual beats and indicate the end of the tenure

only after the final data beat. Data termination signals also indicate whether a condition exists that requires the data phase to be repeated.

Many other MPC7450 signals control and affect other aspects of the device, aside from the bus protocol. They are as follows:

- L3 cache address/data—The MPC7450 has separate address and data buses for accessing the L3 cache. Note that the L3 cache interface is not supported by the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448.
- L3 cache clock/control—These signals provide clocking and control for the L3 cache. Note that the L3 cache interface is not supported by the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448.
- Interrupts/resets—These signals include the external interrupt signal, checkstop signals, and both soft reset and hard reset signals. They are used to interrupt and, under various conditions, to reset the processor.
- Processor status and control—These signals enable the time-base facility and are used to select the bus mode and control sleep mode.
- Clock control—These signals determine the system clock frequency. They are also used to synchronize multiprocessor systems.
- Test interface—The JTAG (IEEE 1149.1a-1993) interface and the common on-chip processor (COP) unit provide a serial interface to the system for performing board-level boundary-scan interconnect tests.
- Voltage selection—These signal control the electrical characteristics of the I/O circuitry of the device as appropriate to support various signaling levels.

NOTE

> Active-low signals are shown with overbars. 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[0:4] (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.

1.2.9.3 MPX Bus Mode Functional Groupings

Figure 1-8 illustrates the signal configuration in MPX bus mode for the MPC7450, MPC7451, MPC7441, MPC7455, and MPC7445, showing how the signals are grouped. A pinout diagram and tables showing pin numbers are included in the hardware specifications. Note that the left side of each figure depicts the signals that implement the MPX bus protocol and the right side of each figure shows the remaining signals on the MPC7450 (not part of the bus protocol).

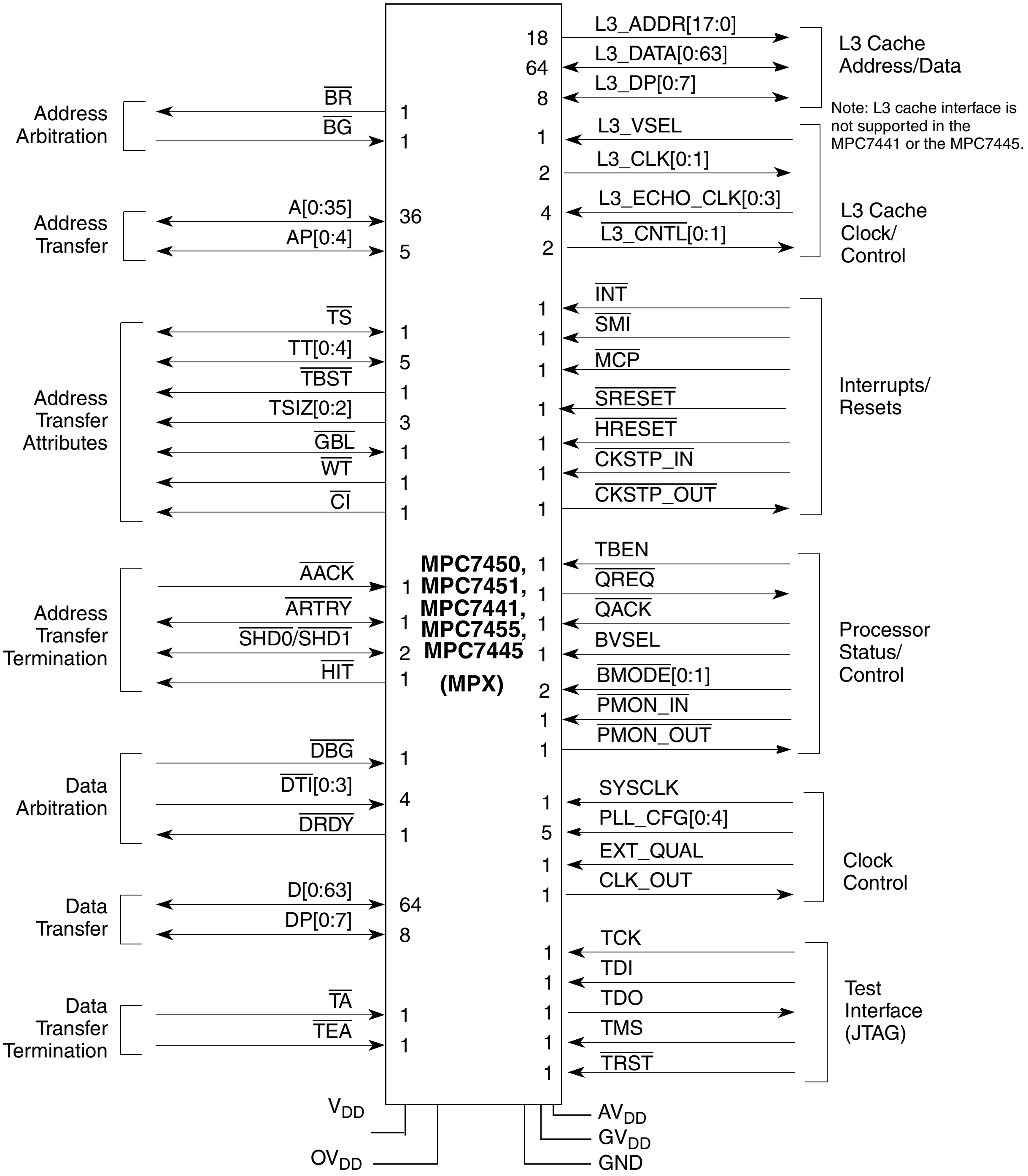
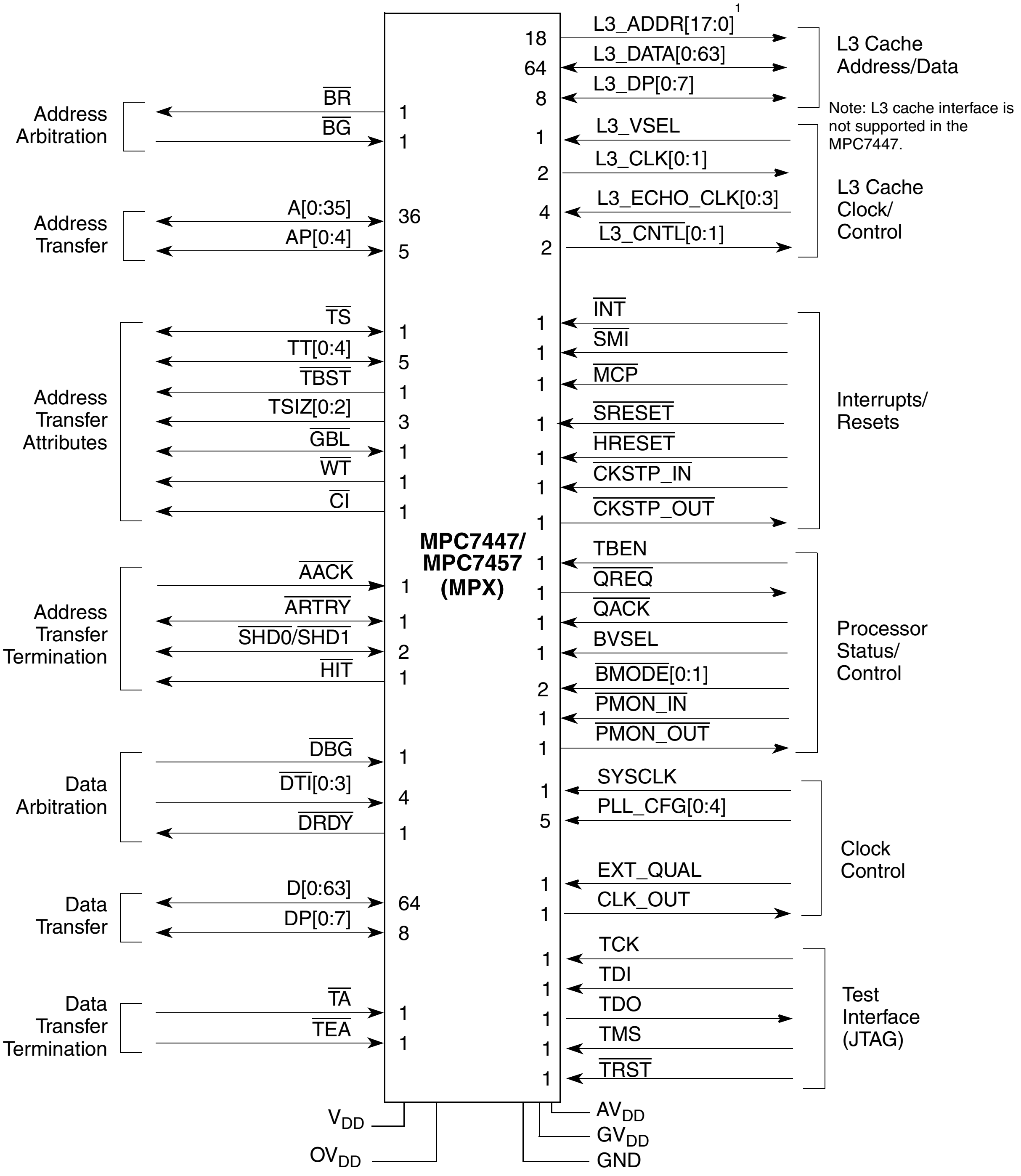


Figure 1-8. MPX Bus Signal Groups in the MPC7450, MPC7451, MPC7441, MPC7455, and MPC7445

Figure 1-9 illustrates the signal configuration in MPX bus mode for the MPC7447 and the MPC7457.



[1] For the MPC7457, there are 19 L3_ADDR signals, (L3_ADDR[0:18]).

Figure 1-9. MPX Bus Signal Groups in the MPC7447 and MPC7457

Figure 1-10 illustrates the signal configuration in MPX bus mode for the MPC7447A.

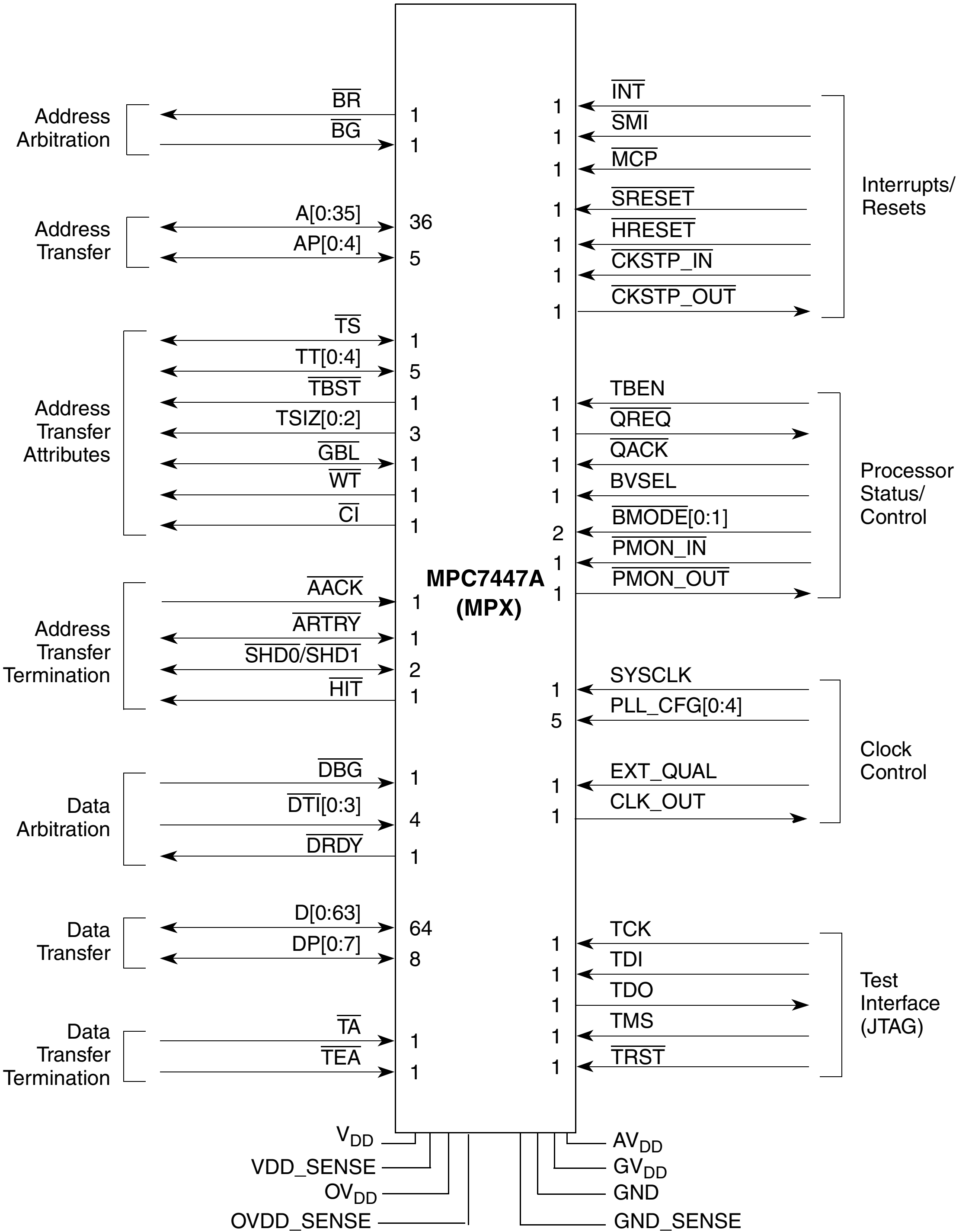


Figure 1-10. MPX Bus Signal Groups in the MPC7447A

Figure 1-11illustrates the signal configuration in MPX bus mode for the MPC7448.

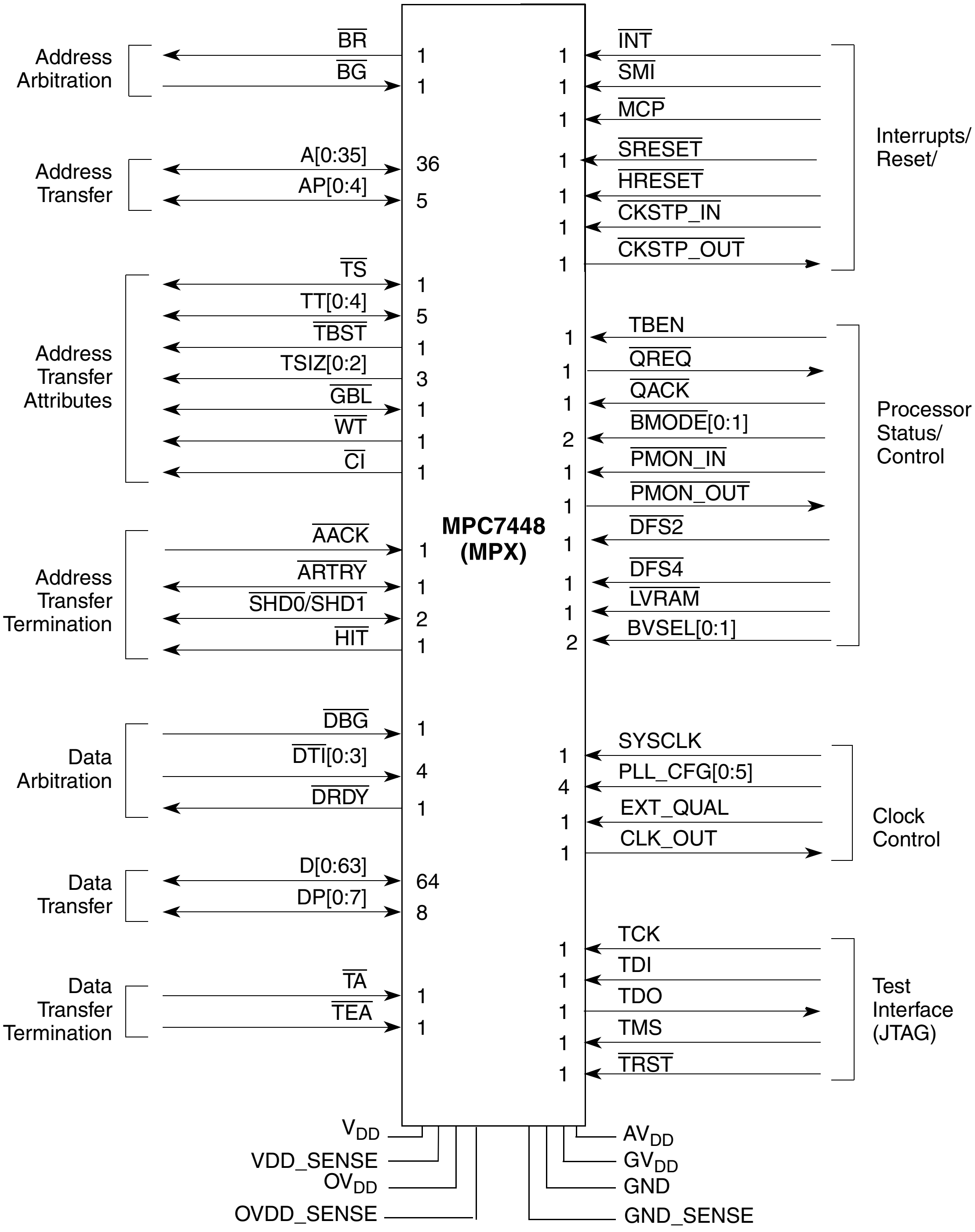


Figure 1-11. MPX Bus Signal Groups in the MPC7448

Signal functionality is described in detail in Chapter 8, “Signal Descriptions,” and Chapter 9, “System Interface Operation.”

1.2.9.3.1 Clocking

For functional operation, the MPC7450 uses a single clock input signal, SYSCLK, from which clocking is derived for the processor core, the L3 interface, and the MPX bus interface. Additionally, internal clock information is made available at the pins to support debug and development.

The MPC7450's clocking structure supports a wide range of processor-to-bus clock ratios. The internal processor core clock is synchronized to SYSCLK with the aid of a VCO-based PLL. The PLL_CFG[0:4] signals (PLL_CFG[0:5] in the MPC7448) are used to program the internal clock rate to a multiple of SYSCLK as defined in the hardware specifications. The bus clock is maintained at the same frequency as SYSCLK. SYSCLK does not need to be a 50% duty-cycle signal.

The MPC7450 generates the clock for the external L3 synchronous data RAMs. The clock frequency for the RAMs is divided down from (and phase-locked to) the MPC7450 core clock frequency using a divisor selected through L3CR[L3CLK]. Note that the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448 do not support the L3 cache or the L3 cache interface.

1.2.10 Power and Thermal Management

The MPC7450 is designed for low-power operation. It provides both automatic and program-controlled power reduction modes. If an MPC7450 functional unit is idle, it automatically goes into a low-power mode. This mode does not affect operational performance. Dynamic power management automatically supplies or withholds power to execution units individually, based upon the contents of the instruction stream. The operation of dynamic power management is transparent to software or any external hardware.

The following three programmable power modes are available to the system:

- Nap—Instruction fetching is halted. Only those clocks for time base, decrementer, and JTAG logic remain running. The MPC7450 goes into the doze state to snoop memory operations on the bus and then back to nap using a $\overline{\text{QREQ}}/\overline{\text{QACK}}$ processor-system handshake protocol.
- Sleep—Power consumption is further reduced by disabling bus snooping, leaving only the PLL in a locked and running state. All internal functional units are disabled.
- Deep sleep—The system can disable the PLL. The system can then disable the SYSCLK source for greater system power savings. Power-on reset procedures for restarting and relocking the PLL must be followed upon exiting deep sleep.

The dynamic frequency switching (DFS) feature in the MPC7447A conserves power by lowering processor operating frequency. The MPC7447A adds the ability to divide the processor-to-system bus ratio by two during normal functional operation. With the introduction of DFS4 mode in the MPC7448, the processor-to-system bus ratio can also be divided by four. Section 10.2.5, "Dynamic Frequency Switching (DFS)," provide information on power saving with DFS in the MPC7447A and the MPC7448.

The MPC7450 also provides an instruction cache throttling mechanism to effectively reduce the instruction execution rate without the complexity and overhead of dynamic clock control. When used with the dynamic power management, instruction cache throttling provides the system designer with a flexible way to control device temperature while allowing the processor to continue operating. For thermal management, the MPC7450 provides a supervisor-level instruction cache throttling control register

(ICTC). Chapter 10, “Power and Thermal Management,” provides information about how to configure the ICTC register for the MPC7450.

1.2.11 Performance Monitor

The MPC7450 incorporates a performance monitor facility that system designers can use to help bring up, debug, and optimize software performance. The performance monitor counts events during execution of instructions related to dispatch, execution, completion, and memory accesses.

The performance monitor incorporates several registers that can be read and written to by supervisor-level software. User-level versions of these registers provide read-only access for user-level applications. These registers are described in Section 1.3.1, “PowerPC Registers and Programming Model.” Performance monitor control registers, MMCR0, MMCR1, and MMCR2 can be used to specify which events are to be counted and the conditions for which a performance monitoring exception is taken. Additionally, the sampled instruction address register, SIAR (USIAR), holds the address of the first instruction to complete after the counter overflowed.

Attempting to write to a user-level read-only performance monitor register causes a program exception, regardless of the MSR[PR] setting.

When a performance monitor exception occurs, program execution continues from vector offset 0x00F00.

Chapter 11, “Performance Monitor,” describes the operation of the performance monitor diagnostic tool incorporated in the MPC7450.

1.3 MPC7450 Microprocessor: Architectural Implementation

The PowerPC architecture consists of three layers. Adherence to the PowerPC architecture can be described in terms of which of the following levels of the architecture is implemented:

- PowerPC user instruction set architecture (UISA)—Defines the base user-level instruction set, user-level registers, data types, floating-point exception model, memory models for a uniprocessor environment, and programming model for a uniprocessor environment
- PowerPC virtual environment architecture (VEA)—Describes the memory model for a multiprocessor environment, defines cache control instructions, and describes other aspects of virtual environments. Implementations that conform to the VEA also adhere to the UISA but 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 MPC7450 implementation supports the three levels of the architecture described above. For more information about the PowerPC architecture, see *PowerPC Microprocessor Family:* the *Programming Environments.* Specific MPC7450 features are listed in Section 1.2, “MPC7450 Microprocessor Features.”

This section describes the PowerPC architecture in general, and specific details about the implementation of the MPC7450 as a low-power, 32-bit device that implements this architecture. The structure of this section follows the reference manual organization; each subsection provides an overview of that chapter.

- Registers and programming model—Section 1.3.1, “PowerPC Registers and Programming Model,” describes the registers for the operating environment architecture common among processors of this family and describes the programming model. It also describes the registers that are unique to the MPC7450.

  Instruction set and addressing modes—Section , “Some registers can be accessed both explicitly and implicitly. In the MPC7450, all SPRs are 32 bits wide. Table 2-1 describes registers implemented by the MPC7450.,” describes the PowerPC instruction set and addressing modes for the PowerPC operating environment architecture, and defines and describes the PowerPC instructions implemented in the MPC7450. The information in this section is described more fully in Chapter 2, “Programming Model.”
- Cache implementation—Section 1.3.3, “On-Chip Cache Implementation,” describes the cache model that is defined generally by the virtual environment architecture. It also provides specific details about the MPC7450 cache implementation. The information in this section is described more fully in Chapter 3, “L1, L2, and L3 Cache Operation.”
- Exception model—Section 1.3.4, “Exception Model,” describes the exception model of the PowerPC operating environment architecture and the differences in the MPC7450 exception model. The information in this section is described more fully in Chapter 4, “Exceptions.”
- Memory management—Section 1.3.5, “Memory Management,” describes generally the conventions for memory management. This section also describes the MPC7450’s implementation of the 32-bit PowerPC memory management specification. The information in this section is described more fully in Chapter 5, “Memory Management.”
- Instruction timing—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 MPC7450. The information in this section is described more fully in Chapter 6, “Instruction Timing.”
- AltiVec implementation—Section 1.3.7, “AltiVec Implementation,” points out that the MPC7450 implements AltiVec registers, instructions, and exceptions as described in the *AltiVec Technology Programming Environments Manual*. Chapter 7, “AltiVec Technology Implementation,” provides complete details.

1.3.1 PowerPC Registers and Programming Model

The PowerPC architecture defines register-to-register operations for most computational instructions. Source operands for these instructions are accessed from the registers or are provided as immediate values embedded in the instruction 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.

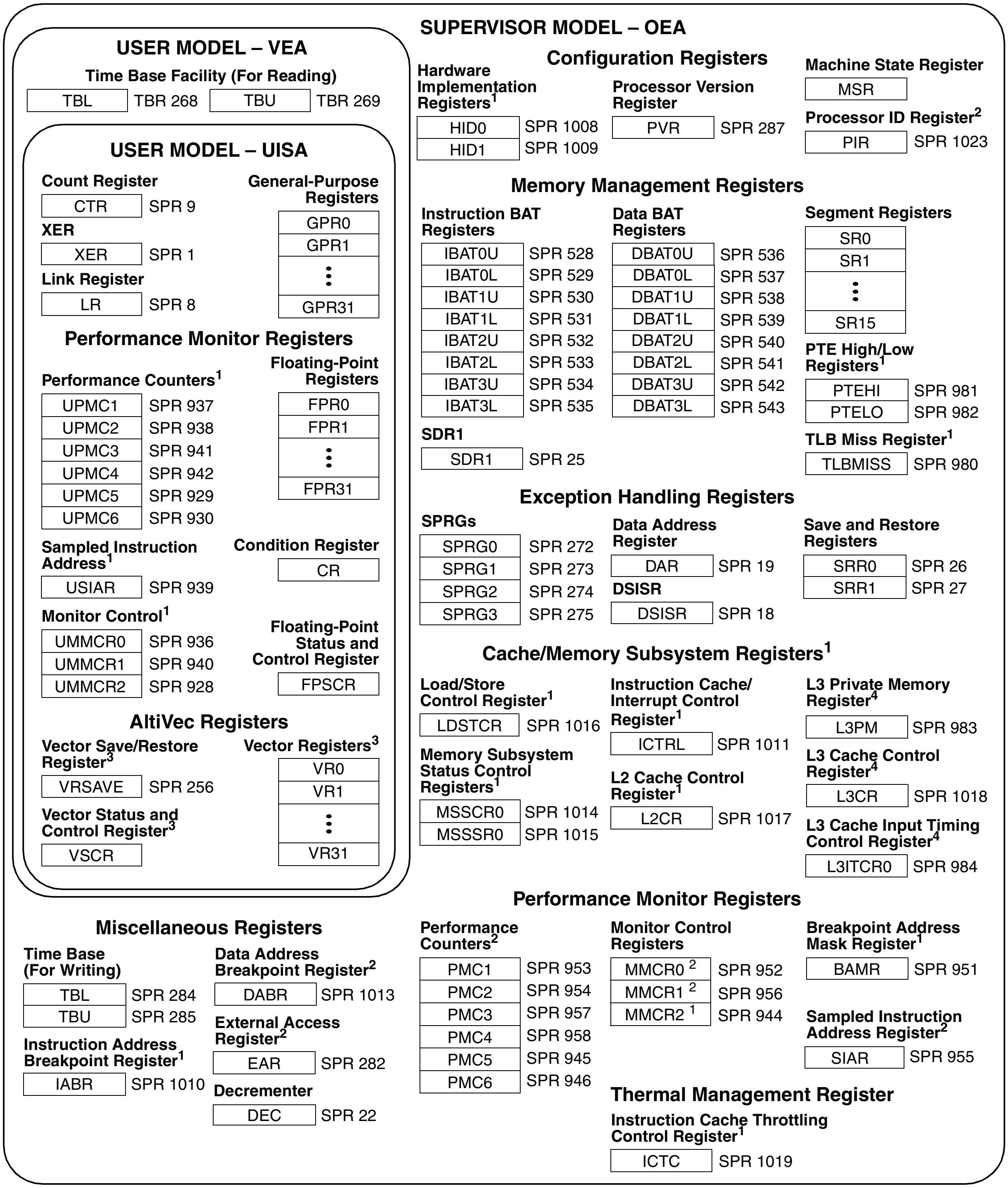
The PowerPC architecture also defines two levels of privilege—supervisor mode of operation (typically used by the operating system) and user mode of operation (used by the application software). The programming models incorporate 32 GPRs, 32 FPRs, SPRs, and several miscellaneous registers. The AltiVec extensions to the PowerPC architecture augment the programming model with 32 VRs, 1 status and control register, and 1 save and restore register. Each processor that implements the PowerPC architecture also has a unique set of implementation-specific registers to support functionality that may not be defined by the PowerPC architecture.

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 1-12 through Figure 1-13 show all the MPC7450 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. For more information, see Chapter 2, “Programming Model.”

The OEA defines numerous SPRs 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 1-12 through Figure 1-13, depending on the program’s access privilege (supervisor or user, determined by the privilege-level bit, MSR[PR]). GPRs, FPRs, and VRs 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.

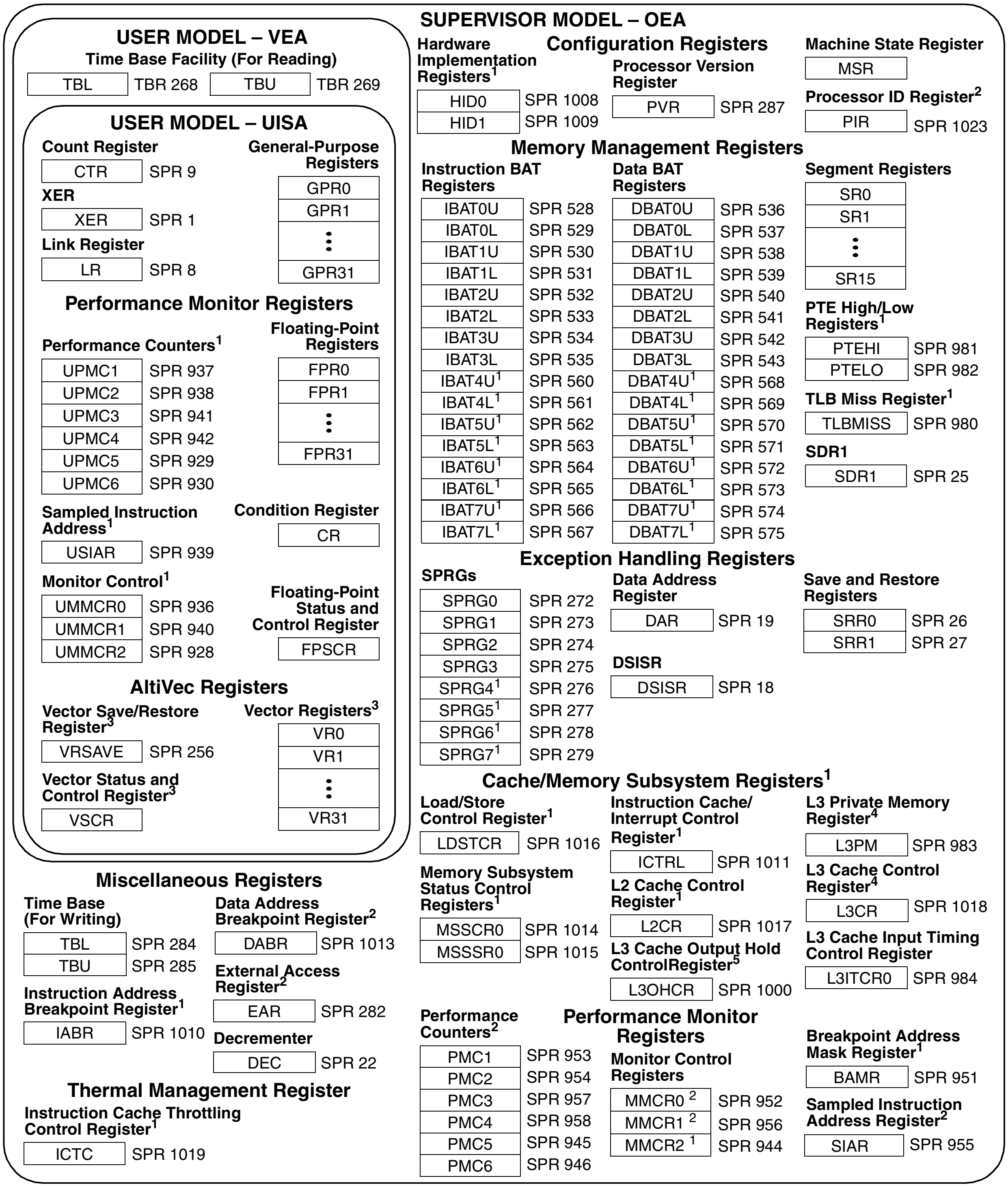
Figure 1-12 shows the MPC7441 and MPC7451 register set.



[1] MPC7441-, MPC7451-specific register may not be supported on other processors that implement the PowerPC architecture.
[2] Register defined as optional in the PowerPC architecture.
[3] Register defined by the AltiVec technology.
[4] MPC7451-specific register.

Figure 1-12. Programming Model—MPC7441/MPC7451 Microprocessor Registers

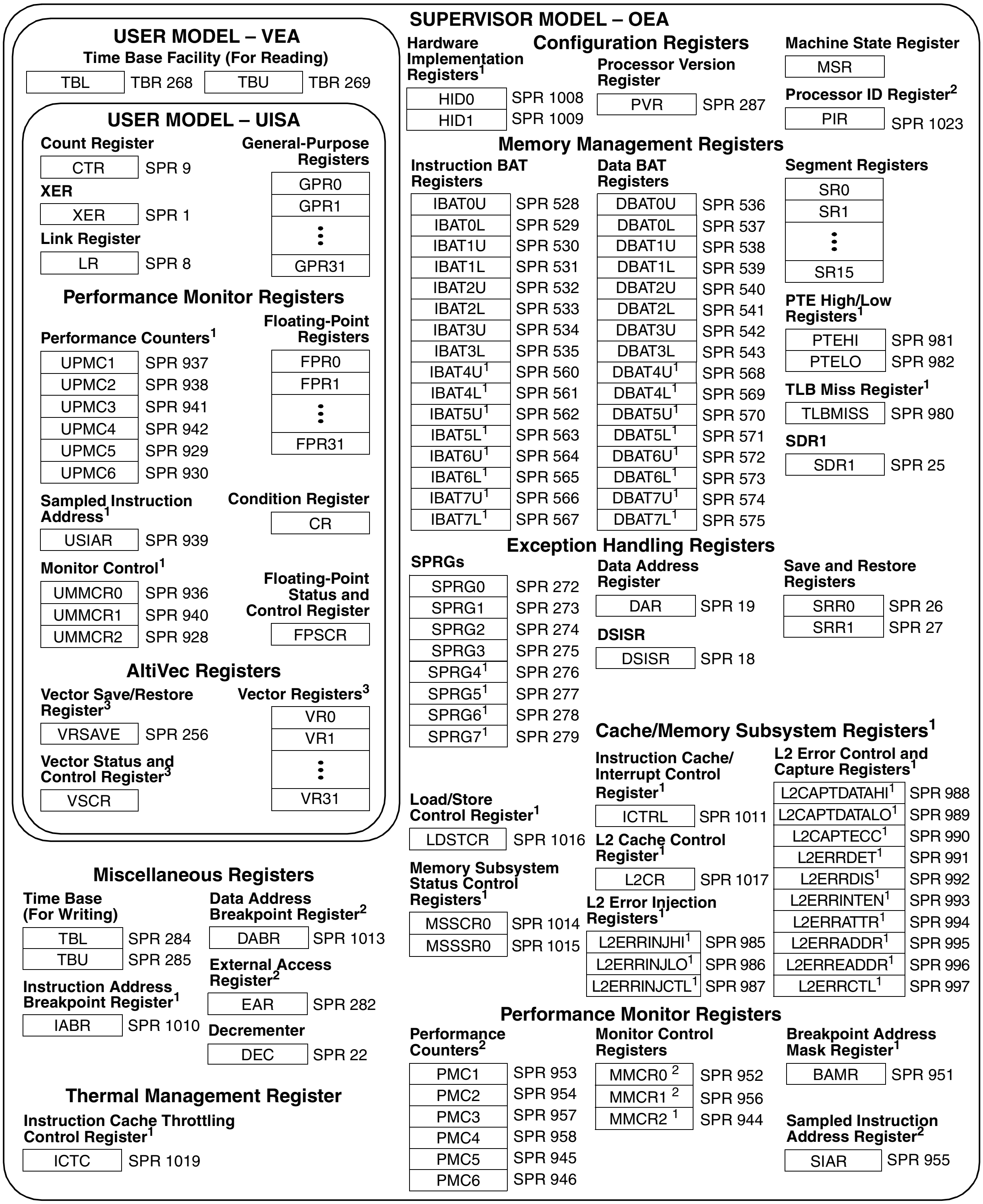
Figure 1-13 shows the MPC7445, MPC7455, MPC7447, MPC7457, and MPC7447A register set.



[1] MPC7445-, MPC7447-, MPC7455-, and MPC7457-specific register may not be supported on other processors that implement the PowerPC architecture.
[2] Register defined as optional in the PowerPC architecture.
[3] Register defined by the AltiVec technology.
[4] MPC7455- and MPC7457-specific register.
[5] MPC7457-specific register.

Figure 1-13. Programming Model—MPC7445, MPC7447, MPC7455, MPC7457, and MPC7447A Microprocessor Registers

Figure 1-14 shows the MPC7448 register set.



[1] MPC7448-specific register may not be supported on other processors that implement the PowerPC architecture.
[2] Register defined as optional in the PowerPC architecture.
[3] Register defined by the AltiVec technology.

Figure 1-14. Programming Model—MPC7448 Microprocessor Registers

Some registers can be accessed both explicitly and implicitly. In the MPC7450, all SPRs are 32 bits wide. Table 2-1 describes registers implemented by the MPC7450.

1.3.2 Instruction Set

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.

For more information, see Chapter 2, “Programming Model.”

1.3.2.1 PowerPC Instruction Set

The PowerPC instructions are divided into the following categories:

- Integer instructions—These include computational and logical instructions.
  - Integer arithmetic instructions
  - Integer compare instructions
  - Integer logical instructions
  - Integer rotate and shift instructions
- Floating-point instructions—These include floating-point computational instructions, as well as instructions that affect the FPSCR.
  - Floating-point arithmetic instructions
  - Floating-point multiply/add instructions
  - Floating-point rounding and conversion instructions
  - Floating-point compare instructions
  - Floating-point status and control instructions
- Load and 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 instructions
  - 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
    - Order loads and stores
- Memory control instructions—These instructions provide control of caches, TLBs, and SRs.
    - Supervisor-level cache management instructions
    - User-level cache instructions
    - Segment register manipulation instructions
    - Translation lookaside buffer management instructions

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

Integer instructions operate on byte, half-word, and word operands. Floating-point instructions operate on single-precision (one word) and double-precision (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.

Processors that implement the PowerPC architecture 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.

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 AltiVec Instruction Set

The AltiVec instructions are divided into the following categories:

- Vector integer arithmetic instructions—These include arithmetic, logical, compare, rotate, and shift instructions.
- Vector floating-point arithmetic instructions—These include floating-point arithmetic instructions, as well as a discussion on floating-point modes.

- Vector load and store instructions—These include load and store instructions for vector registers. The AltiVec technology defines LRU and transient type instructions that can be used to optimize memory accesses.
  - LRU instructions. The AltiVec architecture specifies that the **lvxl** and **stvxl** instructions differ from other AltiVec load and store instructions in that they leave cache entries in a least-recently-used (LRU) state instead of a most-recently-used state.
  - Transient instructions. The AltiVec architecture describes a difference between static and transient memory accesses. A static memory access should have some reasonable degree of locality and be referenced several times or reused over some reasonably long period of time. A transient memory reference has poor locality and is likely to be referenced a very few times or over a very short period of time.

    The following instructions are interpreted to be transient:
    - **dstt** and **dststt** (transient forms of the two data stream touch instructions)
    - **lvxl** and **stvxl**
- Vector permutation and formatting instructions—These include pack, unpack, merge, splat, permute, select, and shift instructions, described in Section 2.6.5, “Vector Permutation and Formatting Instructions.”
- Processor control instructions—These instructions are used to read and write from the AltiVec status and control register, described in Section 2.4.4.6, “Processor Control Instructions.”
- Memory control instructions—These instructions are used for managing of caches (user level and supervisor level), described in Section 2.4.5.3, “Memory Control Instructions.”

1.3.2.3 MPC7450 Microprocessor Instruction Set

The MPC7450 instruction set is defined as follows:

- The MPC7450 provides hardware support for all 32-bit PowerPC instructions.
- The MPC7450 implements the following instructions optional to the PowerPC architecture:
  - External Control In Word Indexed (**eciwx**)
  - External Control Out Word Indexed (**ecowx**)
  - Data Cache Block Allocate (**dcba**)
  - Floating Select (**fsel**)
  - Floating Reciprocal Estimate Single-Precision (**fres**)
  - Floating Reciprocal Square Root Estimate (**frsqrte**)
  - Store Floating-Point as Integer Word (**stfiwx**)
  - Load Data TLB Entry (**tlbld**)
  - Load Instruction TLB Entry (**tlbli)**

1.3.3 On-Chip Cache Implementation

The following subsections describe the PowerPC architecture's treatment of cache in general, and the MPC7450-specific implementation, respectively. A detailed description of the MPC7450 cache implementation is provided in Chapter 3, "L1, L2, and L3 Cache Operation."

1.3.3.1 PowerPC Cache Model

The PowerPC architecture does not define hardware aspects of cache implementations. For example, processors that implement the PowerPC architecture can have unified caches, separate L1 instruction and data caches (Harvard architecture), or no cache at all. These microprocessors control the following memory access modes on a page or block basis:

- Write-back/write-through mode
- Caching-inhibited/caching-allowed mode
- Memory coherency required/memory coherency not required mode

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

The PowerPC architecture defines the term 'cache block' as the cacheable unit. The VEA and OEA define cache management instructions a programmer can use to affect cache contents.

1.3.3.2 MPC7450 Microprocessor Cache Implementation

The MPC7450 cache implementation is described in Section 1.2.4, "On-Chip L1 Instruction and Data Caches," Section 1.2.5, "L2 Cache Implementation," and Section 1.2.6, "L3 Cache Implementation." The BPU also contains a 128-entry BTIC that provides immediate access to cached target instructions. For more information, see Section 1.2.2.2, "Branch Processing Unit (BPU)."

1.3.4 Exception Model

The following sections describe the PowerPC exception model and the MPC7450 implementation. A detailed description of the MPC7450 exception model is provided in Chapter 4, "Exceptions."

1.3.4.1 PowerPC Exception Model

The OEA portion of the PowerPC architecture defines the mechanism by which processors that implement the PowerPC architecture invoke exceptions. Exception conditions may be defined at other levels of the architecture. For example, the UISA defines conditions that may cause floating-point exceptions; the OEA defines the mechanism by which the exception is taken.

The PowerPC exception mechanism allows the processor to change to supervisor state as a result of unusual conditions arising in the execution of instructions and from external signals, bus errors, or various internal conditions. When exceptions occur, information about the state of the processor is saved to certain registers and the processor begins execution at an address (exception vector) predetermined for each exception. Processing of exceptions begins in supervisor mode.

Although multiple exception conditions can map to a single exception vector, often a more specific condition may be determined by examining a register associated with the exception—for example, the DSISR and the floating-point status and control register (FPSCR). Also, software can explicitly enable or disable some exception conditions.

The PowerPC architecture requires that exceptions be taken in program order; therefore, although a particular implementation 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. In addition, if a single instruction encounters multiple exception conditions, those exceptions are taken and handled sequentially. Likewise, exceptions that are asynchronous and precise are recognized when they occur, but are not handled until all instructions currently in the execute stage successfully complete execution and report their results.

To prevent loss of state information, exception handlers must save the information stored in the machine status save/restore registers, SRR0 and SRR1, soon after the exception is taken to prevent this information from being lost due to another exception event. Because exceptions can occur while an exception handler routine is executing, multiple exceptions can become nested. It is the exception handler’s responsibility to save the necessary state information if control is to return to the excepting program.

In many cases, after the exception handler handles an exception, there is an attempt to execute the instruction that caused the exception. Instruction execution continues until the next exception condition is encountered. Recognizing and handling exception conditions sequentially guarantees that the machine state is recoverable and processing can resume without losing instruction results.

The following terms are used to describe the stages of exception processing: recognition, taken, and handling.

- 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 routine begins executing in supervisor mode.
- Handling—Exception handling is performed by the software at the appropriate vector offset. Exception handling is begun in supervisor mode.

The term ‘interrupt’ is used to describe the external interrupt, the system management interrupt, and sometimes the asynchronous exceptions. Note that the PowerPC architecture uses the word ‘exception’ to refer to IEEE-defined floating-point exception conditions that may cause a program exception to be taken; see Section 4.6.7, “Program Exception (0x00700).” The occurrence of these IEEE exceptions may or may not cause an exception to be taken. IEEE-defined exceptions are referred to as IEEE floating-point exceptions or floating-point exceptions.

1.3.4.2 MPC7450 Microprocessor Exceptions

As specified by the PowerPC architecture, exceptions can be either precise or imprecise and either synchronous or asynchronous. Asynchronous exceptions are caused by events external to the processor’s execution; synchronous exceptions are caused by instructions.

The types of exceptions are shown in Table 1-1. Note that all exceptions except for the performance monitor, AltiVec unavailable, instruction address breakpoint, system management, AltiVec assist, and the three software table search exceptions are described in Chapter 6, “Exceptions,” in the *Programming Environments Manual*.

Table 1-1. MPC7450 Microprocessor Exception Classifications

Synchronous/Asynchronous	Precise/Imprecise	Exception Types
Asynchronous, nonmaskable	Imprecise	System reset, machine check
Asynchronous, maskable	Precise	External interrupt, system management interrupt, decrementer exception, performance monitor exception
Synchronous	Precise	Instruction-caused exceptions

The exception classifications are discussed in greater detail in Section 4.2, “MPC7450 Exception Recognition and Priorities.” For a better understanding of how the MPC7450 implements precise exceptions, see Chapter 6, “Instruction Timing.” Table 1-2 lists the exceptions implemented in the MPC7450, and conditions that cause them. Table 1-2 also notes the MPC7450-specific exceptions.

The three software table search exceptions support software page table searching and are enabled by setting HID0[STEN]. See Section 4.6.15, “TLB Miss Exceptions,” and Chapter 5, “Memory Management.”

Table 1-2. Exceptions and Conditions

Exception Type	Vector Offset	Causing Conditions
Reserved	0x00000	—
System reset	0x00100	Assertion of either $\overline{\text{HRESET}}$ or $\overline{\text{SRESET}}$ or at power-on reset
Machine check	0x00200	Assertion of $\overline{\text{TEA}}$ during a data bus transaction, assertion of $\overline{\text{MCP}}$, an address bus parity error on MPX bus, a data bus parity error on MPXbus, an L1 instruction cache error, and L1 data cache error, a memory subsystem detected error including the following: • L2 data parity error • L2 tag parity error • L3 SRAM error • L3 tag parity error • Single-bit and multiple-bit L2 ECC errors MSR[ME] must be set. Note that the L3 cache is not supported on the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448.
DSI	0x00300	As specified in the PowerPC architecture. Also includes the following: • A hardware table search due to a TLB miss on load, store, or cache operations results in a page fault. • Any load or store to a direct-store segment (SR[T] = 1). • A **lwarx** or **stwcx.** instruction to memory with cache-inhibited or write-through memory/cache access attributes.
ISI	0x00400	As specified in the PowerPC architecture
External interrupt	0x00500	MSR[EE] = 1 and $\overline{\text{INT}}$ is asserted

Table 1-2. Exceptions and Conditions (continued)

Exception Type	Vector Offset	Causing Conditions
Alignment	0x00600	• A floating-point load/store, **stmw**, **stwcx.**, **lmw**, **lwarx**, **eciwx**, or **ecowx** instruction operand is not word-aligned. • A multiple/string load/store operation is attempted in little-endian mode • An operand of a **dcbz** instruction is on a page that is write-through or cache-inhibited for a virtual mode access. • An attempt to execute a **dcbz** instruction occurs when the cache is disabled or locked.
Program	0x00700	As specified in the PowerPC architecture
Floating-point unavailable	0x00800	As specified in the PowerPC architecture
Decrementer	0x00900	As defined by the PowerPC architecture, when the msb of the DEC register changes from 0 to 1 and MSR[EE] = 1.
Reserved	0x00A00–00BFF	—
System call	0x00C00	Execution of the System Call (**sc**) instruction
Trace	0x00D00	MSR[SE] =1 or a branch instruction is completing and MSR[BE] =1. The MPC7450 operates as specified in the OEA by taking this exception on an **isync**.
Reserved	0x00E00	The MPC7450 does not generate an exception to this vector. Other processors that implement the PowerPC architecture may use this vector for floating-point assist exceptions.
Reserved	0x00E10–00EFF	—
Performance monitor	0x00F00	The limit specified in PMC*n* is met and MMCR0[ENINT] = 1 (MPC7450-specific)
AltiVec unavailable	0x00F20	Occurs due to an attempt to execute any non-streaming AltiVec instruction when MSR[VEC] = 0. This exception is not taken for data streaming instructions (**dst*x***, **dss,** or **dssall**). (MPC7450-specific)
ITLB miss	0x01000	An instruction translation miss exception is caused when HID0[STEN] = 1 and the effective address for an instruction fetch cannot be translated by the ITLB (MPC7450-specific).
DTLB miss-on-load	0x01100	A data load translation miss exception is caused when HID0[STEN] = 1 and the effective address for a data load operation cannot be translated by the DTLB (MPC7450-specific).
DTLB miss-on-store	0x01200	A data store translation miss exception is caused when HID0[STEN] = 1 and the effective address for a data store operation cannot be translated by the DTLB, or when a DTLB hit occurs, and the changed bit in the PTE must be set due to a data store operation (MPC7450-specific).
Instruction address breakpoint	0x01300	IABR[0–29] matches EA[0–29] of the next instruction to complete and IABR[BE] = 1 (MPC7450-specific).
System management interrupt	0x01400	MSR[EE] = 1 and $\overline{\text{SMI}}$ is asserted (MPC7450-specific).
Reserved	0x01500–015FF	—

Table 1-2. Exceptions and Conditions (continued)

Exception Type	Vector Offset	Causing Conditions
AltiVec assist	0x01600	This MPC7450-specific exception supports denormalization detection in Java mode as specified in the *AltiVec Technology Programming Environments Manual*.
Reserved	0x01700–02FFF	—

1.3.5 Memory Management

The following subsections describe the memory management features of the PowerPC architecture, and the MPC7450 implementation, respectively.

1.3.5.1 PowerPC Memory Management Model

The primary function of the MMU in a processor that implement the PowerPC architecture 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. Note that the MPC7450 does not implement the optional direct-store facility.

Two general types of memory accesses generated by processors that implement the PowerPC architecture require address translation—instruction accesses and data accesses generated by load and store instructions. In addition, the addresses specified by cache instructions and the optional external control instructions also require translation. Generally, the address translation mechanism is defined in terms of the segment descriptors and page tables that the processors use to locate the effective-to-physical address mapping for memory accesses. The segment information translates the effective address to an interim virtual address, and the page table information translates the 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 MPC7450). In addition, two translation lookaside buffers (TLBs) are implemented on the MPC7450 to keep recently used page address translations on-chip. Although the PowerPC OEA describes one MMU (conceptually), the MPC7450 hardware maintains separate TLBs and table search resources for instruction and data accesses that can be performed independently (and simultaneously). Therefore, the MPC7450 is described as having two MMUs, one for instruction accesses (IMMU) and one for data accesses (DMMU).

The block address translation (BAT) mechanism is a software-controlled array that stores the available block address translations on-chip. BAT array entries are implemented as pairs of BAT registers that are accessible as supervisor special-purpose registers (SPRs). There are separate instruction and data BAT mechanisms. In the MPC7450, 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. Section 4.3, “Exception Processing,” describes how the MSR controls critical MMU functionality.

1.3.5.2 MPC7450 Microprocessor Memory Management Implementation

The MPC7450 implements separate MMUs for instructions and data. It maintains a copy of the segment registers in the instruction MMU; however, read and write accesses to the segment registers (**mfsr** and **mtsr**) are handled through the segment registers in the data MMU. The MPC7450 MMU is described in Section 1.2.3, “Memory Management Units (MMUs).”

The MPC7450 implements the memory management specification of the PowerPC OEA for 32-bit implementations but adds capability for supporting 36-bit physical addressing. Thus, it provides 4 Gbytes of physical address space accessible to supervisor and user programs, with a 4-Kbyte page size and 256-Mbyte segment size. In addition, the MPC7450 MMUs use an interim virtual address (52 bits) and hashed page tables in the generation of 32- or 36-bit physical addresses (depending on the setting of HID0[XAEN]). Processors that implement the PowerPC architecture also have a BAT mechanism for mapping large blocks of memory. Block range from 128 Kbytes to 256 Mbytes and are software programmable.

The MPC7450 provides table search operations performed in hardware. The 52-bit virtual address is formed and the MMU attempts to fetch the PTE that contains the physical address from the appropriate TLB on-chip. If the translation is not found in either the BAT array or in a TLB (that is, a TLB miss occurs), the hardware performs a table search operation (using a hashing function) to search for the PTE. Hardware table searching is the default mode for the MPC7450; however, if HID0[STEN] = 1, a software table search is performed.

The MPC7450 also provides support for table search operations performed in software (if HID0[STEN] is set). In this case, the TLBMISS register saves the effective address of the access that requires a software table search. The PTEHI and PTELO registers and the **tlbli** and **tlbld** instructions are used in reloading the TLBs during a software table search operation. The following exceptions support software table searching if HID0[STEN] is set and a TLB miss occurs:

- For an instruction fetch, an ITLB miss exception
- For a data load, a DTLB miss-on-load exception
- For a data store, a DTLB miss-on-store exception

The MPC7450 implements the optional TLB invalidate entry (**tlbie**) and TLB synchronize (**tlbsync**) instructions that can be used to invalidate TLB entries. For more information on the **tlbie** and **tlbsync** instructions, see Section 5.4.4.2, “TLB Invalidation.”

1.3.6 Instruction Timing

This section describes how the MPC7450 microprocessor performs operations defined by instructions and how it reports the results of instruction execution. The MPC7450 design minimizes average instruction execution latency, which is the number of clock cycles it takes to fetch, decode, dispatch, issue, and execute instructions and make results available for subsequent instructions. Some instructions, such as loads and stores, access memory and require additional clock cycles between the execute phase and the write-back phase. Latencies depend on whether an access is to cacheable or noncacheable memory, whether it hits in the L1, L2, or L3 cache, whether a cache access generates a write back to memory, whether the access causes a snoop hit from another device that generates additional activity, and other conditions that affect memory accesses.

To improve throughput, the MPC7450 implements pipelining, superscalar instruction issue, branch folding, removal of fall-through branches, three-level speculative branch handling, and multiple execution units that operate independently and in parallel.

As an instruction passes from stage to stage, the subsequent instruction can follow through the stages as the preceding instruction vacates them, allowing several instructions to be processed simultaneously. Although it may take several cycles for an instruction to pass through all the stages, when the pipeline is full, one instruction can complete its work on every clock cycle. Figure 1-15 represents a generic four-stage pipelined execution unit, which when filled has a throughput of one instruction per clock cycle.

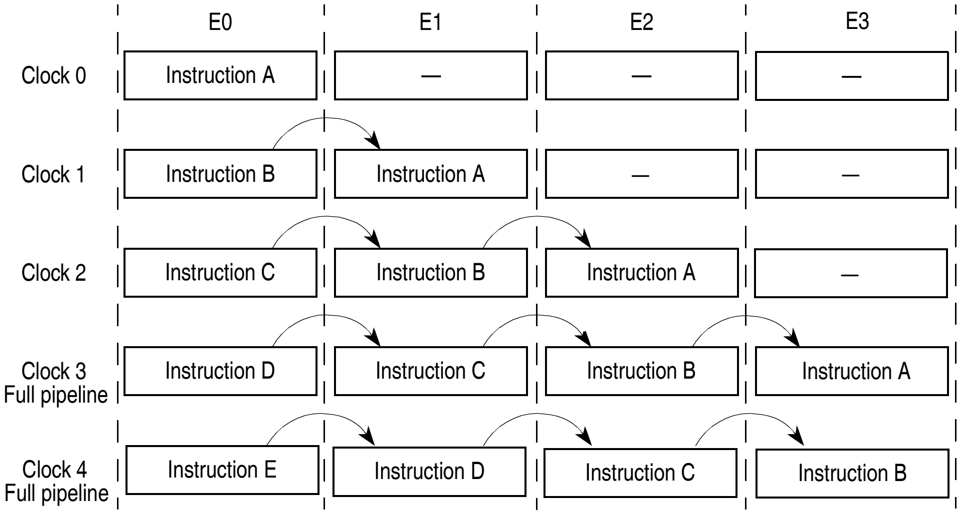


Figure 1-15. Pipelined Execution Unit

Figure 1-16 shows the entire path that instructions take through the fetch1, fetch2, decode/dispatch, execute, issue, complete, and write-back stages, which is considered the MPC7450's master pipeline. The FPU, LSU, IU2, VIU2, VFPU, and VPU are multiple-stage pipelines.

The MPC7450 contains the following execution units:

- Branch processing unit (BPU)
- Three integer unit 1s (IU1a, IU1b, and IU1c)—execute all integer instructions except multiply, divide, and move to/from SPR instructions
- Integer unit 2 (IU2)—executes miscellaneous instructions including the CR logical operations, integer multiplication and division instructions, and move to/from special-purpose register instructions
- 64-bit floating-point unit (FPU)
- Load/store unit (LSU)
- The AltiVec unit contains the following four independent execution units for vector computations; the latencies are shown in Table 7-12.
  - AltiVec permute unit (VPU)
  - AltiVec integer unit 1 (VIU1)
  - Vector integer unit 2 (VIU2)

— Vector floating-point unit (VFPU)

A maximum of two AltiVec instructions can be issued in order to any combination of AltiVec execution units per clock cycle. In the MPC7448, a maximum of two AltiVec instructions can be issued out-of-order to any combination of AltiVec execution units per clock cycle from the bottom two VIQ entries (VIQ1–VIQ0). An instruction in VIQ1 destined for VIU1 does not have to wait for an instruction in VIQ0 that is stalled behind an instruction waiting for operand availability. Moreover, the VIU2, VFPU, and VPU are pipelined, so they can operate on multiple instructions.

The MPC7450 can complete as many as three instructions on each clock cycle. In general, the MPC7450 processes instructions in seven stages—fetch1, fetch2, decode/dispatch, issue, execute, complete, and writeback as shown in Figure 1-16. Note that the pipeline example in Figure 6-1 is similar to the four-stage VFPU pipeline in Figure 1-16.

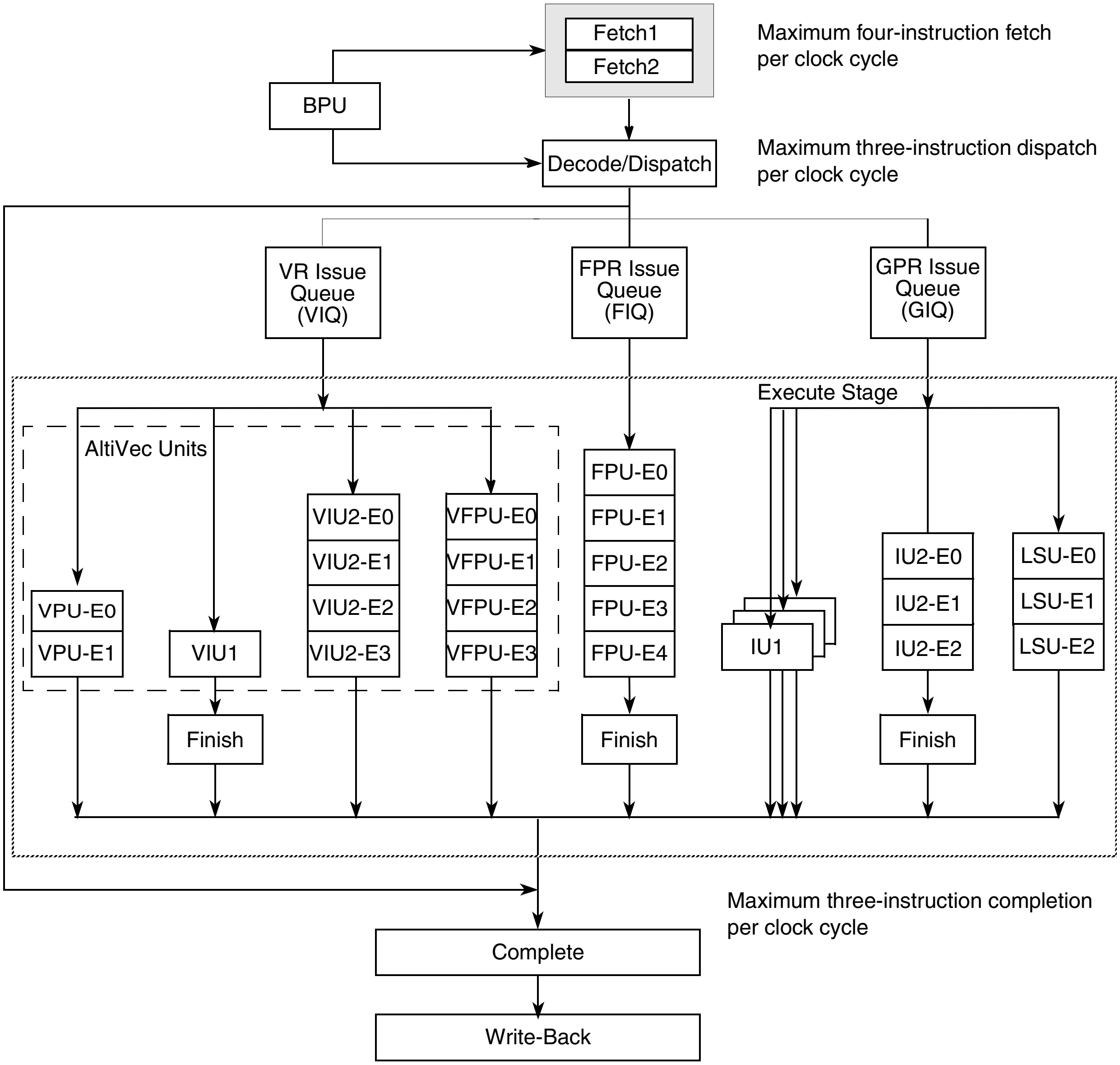


Figure 1-16. Superscalar/Pipeline Diagram

The instruction pipeline stages are described as follows:

- Instruction fetch—Includes the clock cycles necessary to request an instruction and the time the memory system takes to respond to the request. Instructions retrieved are latched into the instruction queue (IQ) for subsequent consideration by the dispatcher.

  Instruction fetch timing depends on many variables, such as whether an instruction is in the branch target instruction cache (BTIC), the on-chip instruction cache, or the L2 or L3 cache. Those factors increase when it is necessary to fetch instructions from system memory and include the processor-to-bus clock ratio, the amount of bus traffic, and whether any cache coherency operations are required.

- The decode/dispatch stage fully decodes each instruction; most instructions are dispatched to the issue queues (branch, **isync**, **rfi**, and **sc** instructions do not go to issue queues).
- The three issue queues, FIQ, VIQ, and GIQ, can accept as many as one, two, and three instructions, respectively, in a cycle. Instruction dispatch requires the following:
  - — Instructions are dispatched only from the three lowest IQ entries—IQ0, IQ1, and IQ2.
  - — A maximum of three instructions can be dispatched to the issue queues per clock cycle.
  - — Space must be available in the CQ for an instruction to dispatch (this includes instructions that are assigned a space in the CQ but not an issue queue).
- The issue stage reads source operands from rename registers and register files and determines when instructions are latched into the execution unit reservation stations. The GIQ, FIQ, and VIQ (AltiVec) issue queues have the following similarities:
  - – Operand lookup in the GPRs, FPRs, and VRs, and their rename registers.
  - – Issue queues issue instructions to the proper execution units.
  - – Each issue queue holds twice as many instructions as can be dispatched to it in one cycle; the GIQ has six entries, the VIQ has four, and the FIQ has two.

  The three issue queues are described as follows:

  - — The GIQ accepts as many as three instructions from the dispatch unit each cycle. IU1, IU2, and all LSU instructions (including floating-point and AltiVec loads and stores) are dispatched to the GIQ.
  - — Instructions can be issued out-of-order from the bottom three GIQ entries (GIQ2–GIQ0). An instruction in GIQ1 destined for an IU1 does not have to wait for an instruction in GIQ0 that is stalled behind a long-latency integer divide instruction in the IU2.
  - — The VIQ accepts as many as two instructions from the dispatch unit each cycle. All AltiVec instructions (other than load, store, and vector touch instructions) are dispatched to the VIQ. As many as two instructions can be issued to the four AltiVec execution units, but unlike the GIQ, instructions in the VIQ cannot be issued out-of-order.
  - — The FIQ can accept one instruction from the dispatch unit per clock cycle. It looks at the first instruction in its queue and determines if the instruction can be issued to the FPU in this cycle.
- The execute stage accepts instructions from its issue queue when the appropriate reservation stations are not busy. In this stage, the operands assigned to the execution stage from the issue stage are latched.

  The execution unit executes the instruction (perhaps over multiple cycles), writes results on its result bus, and notifies the CQ when the instruction finishes. The execution unit reports any exceptions to the completion stage. Instruction-generated exceptions are not taken until the excepting instruction is next to retire.

  Most integer instructions have a 1-cycle latency, so results of these instructions are available 1 clock cycle after an instruction enters the execution unit. The FPU, LSU, IU2, VIU2, VFPU, and VPU units are pipelined, as shown in Figure 7-3.

  Note that AltiVec computational instructions are executed in the four independent, pipelined AltiVec execution units. The VPU has a two-stage pipeline, the VIU1 has a one-stage pipeline, and the VIU2 and VFPU have four-stage pipelines. As many as 10 AltiVec instructions can be executing concurrently.

- The complete and write-back stages maintain the correct architectural machine state and commit results to the architected registers in the proper order. If completion logic detects an instruction containing an exception status, all following instructions are cancelled, their execution results in rename buffers are discarded, and the correct instruction stream is fetched.

  The complete stage ends when the instruction is retired. Three instructions can be retired per clock cycle. If no dependencies exist, as many as three instructions are retired in program order. Section 6.7.4, "Completion Unit Resource Requirements," describes completion dependencies.

The write-back stage occurs in the clock cycle after the instruction is retired.

1.3.7 AltiVec Implementation

The MPC7450 implements the AltiVec registers and instruction set as they are described by the *AltiVec Technology Programming Environments Manual.* Two additional implementation specific exceptions have been added; they are as follows:

- The AltiVec assist exception, which is used in handling denormalized numbers in Java mode
- An alignment exception for cache-inhibited AltiVec loads and stores and write-through stores that execute when in 60x bus mode

Both exceptions are described fully in Chapter 4, "Exceptions." Also, the default setting for VSCR[NJ] bit has changed from being non-Java compliant (VSCR[NJ] = 1) in the MPC7400/7410 to having a default setting of Java–compliant (VSCR[NJ] = 0) in the MPC7450. The AltiVec implementation is described fully in Chapter 7, "AltiVec Technology Implementation."

1.4 Differences Between MPC7450 and MPC7400/MPC7410

Table 1-3 compares the key features of the MPC7450 with the earlier MPC7400/MPC7410. To achieve a higher frequency, the number of logic levels per clock cycle is reduced. In addition, the pipeline of the MPC7450 is extended (compared to the MPC7400), while maintaining the same level of performance (in terms of number of instructions executed per clock cycle. Table 1-3 shows these differences.

Table 1-3. MPC7450 and MPC7400/MPC7410 Feature Comparison

Microarchitectural Feature	MPC7450	MPC7400/MPC7410
Basic Pipeline Functions		
Logic inversions per cycle	18	28
Pipeline stages up to execute	5	3
Total pipeline stages (minimum)	7	4
Pipeline maximum instruction throughput	3 + branch	2 + branch
Pipeline Resources		
Instruction queue size	12	6

Table 1-3. MPC7450 and MPC7400/MPC7410 Feature Comparison (continued)

Microarchitectural Feature	MPC7450	MPC7400/MPC7410
Completion queue size	16	8
Renames (GPR, FPR, VR)	16, 16, 16	6, 6, 6
Maximum Execution Throughput		
Short-latency integer units (IU1s)	3	2
Vector units	2 (any 2 of 4 units)	2 (permute/integer)
Floating-point unit	1	
Out-of-Order Window Size in Execution Queues		
Short-latency integer units	1 entry * 3 queues	1 entry * 2 queues
Vector units	In order, 4 queues	In order, 2 queues
Floating-point unit	In order	
Branch Processing Resources		
Prediction structures	BTIC, BHT, link stack	BTIC, BHT
BTIC size, associativity	128-entry, 4-way	64-entry, 4-way
BHT size	2K-entry	512-entry
Link stack depth	8	None
Unresolved branches supported	3	2
Branch taken penalty (BTIC hit)	1	0
Minimum misprediction penalty	6	4
Execution Unit Timings (Latency-Throughput)		
Aligned load (integer, float, vector)	3-1, 4-1, 3-1	2-1, 2-1, 2-1
Misaligned load (integer, float, vector)	4-2, 5-2, 4-2	3-2, 3-2, 3-2
L1 miss, L2 hit latency	9—data access 13—instruction access	9 (11)[1]
IU1s (adds, subs, shifts, rotates, compares, logicals)	1-1	1-1
Integer multiply (32 * 8, 32 * 16, 32 * 32)	3-1, 3-1, 4-2	2-1, 3-2, 5-4
Scalar floating-point	5-1	3-1
VIU1 (vector integer unit 1—shorter latency vector integer)	1-1	1-1
VIU2 (vector integer unit 2—longer latency vector integer)	4-1	3-1
VFPU (vector floating-point)	4-1	4-1
VPU (vector permute)	2-1	1-1

Table 1-3. MPC7450 and MPC7400/MPC7410 Feature Comparison (continued)

Microarchitectural Feature	MPC7450	MPC7400/MPC7410
MMUs		
MMUs (instruction and data)	128-entry, 2-way	128-entry, 2-way
Table search mechanism	Hardware and software	Hardware
L1 Instruction Cache/Date Cache Features		
Size	32K/32K	
Associativity	8-way	
Locking granularity/style	4-Kbyte/way	Full cache
Parity on instruction cache	Word	None
Parity on data cache	Byte	None
Number of data cache misses (load/store)	5/1	8 (any combination)
Data stream touch engines	4 streams	
On-Chip L2 Cache Features		
Cache level	L2	Tags and controller only (see off-chip cache support below)
Size/associativity	256-Kbytes/8-way	
Access width	256 bits	
Number of 32-byte sectors/line	2	
Parity	Byte	
Off-Chip Cache Support		
Cache level	L3	L2
On-chip tag logical size	1 Mbyte, 2 Mbytes	512 Kbytes, 1 Mbyte, 2 Mbytes
Associativity	8-way	2-way
Number of 32-byte sectors/line	2, 4	1, 2, 4
Off-chip data SRAM support	MSUG2 DDR, LW, PB2	LW, PB2, PB3
Data path width	64	
Private memory SRAM sizes	1 Mbyte, 2 Mbytes	512 Kbyte, 1 Mbyte, 2 Mbytes
Parity	Byte	

---

[1] Numbers in parentheses are for 2:1 SRAM.

1.5 Differences Between MPC7441/MPC7451 and MPC7445/MPC7455

Table 1-4 compares the key differences between the MPC7451 and the MPC7455. The table provides the section number where the details of the differences are discussed. Differences between the two processors are defined through-out the manual. Table 1-4 provides a high-level overview to the differences. Table 1-4 shows these differences.

Table 1-4. MPC7451 and MPC7455 Differences

Microarchitectural Feature	MPC7441/MPC7451	MPC7445/MPC7455	Section
MMU			
Block address translation (BAT) registers —Maps regions of memory	16 BAT registers	32 BATs[1] —8 additional instruction and 8 data BAT registers IBAT4U IBAT4L IBAT5U IBAT5L IBAT6U IBAT6L IBAT7U IBAT7L DBAT4U DBAT4L DBAT5U DBAT5L DBAT6U DBAT6L DBAT7U DBAT7L	1.1.3 5.3.1
SPRGs —Used by system software for software table searches	4 SPRs	8 SPRs —4 additional SPRs registers SPRG4–SPRG7	5.5.5.1.3
Additional HID0 bits		HID0[HIGH_BAT_EN] = 1, enables additional BATs	5.3.1
	Block size range = 128 Kbytes to 256 Mbytes	HID0[XBSEN] = 1, increases block size, Block size range = 128 Kbytes to 4 Gbytes	5.3.2.1

[1] The 32 BAT registers count the upper BAT as one, and the lower BAT as one.

1.6 Differences Between MPC7441/MPC7451 and MPC7447/MPC7457

Table 1-5 compares the key differences between the MPC7451 and the MPC7457. The table provides the section number where the details of the differences are discussed. Differences between the two processors are defined throughout the manual. Table 1-5 provides a high-level overview of the differences.

Table 1-5. MPC7451 and MPC7457 Differences

Microarchitectural Feature	MPC7441/MPC7451	MPC7447/MPC7457	Section
L2 Cache			
Cache level	L2		3.6
Size/associativity	256-Kbyte/8-way	512-Kbyte/8-way	3.6.1
Access width	256 bits		3.6
Number of 32-byte sectors/ line	2		3.6
Parity	Byte		3.6.3.1.2
Off-Chip Cache Support[1]			
Cache level	L3		3.7
On-chip tag logical size	1 Mbyte, 2 Mbytes	1 Mbyte, 2 Mbytes, 4 Mbytes	3.7.3.2
Associativity	8-way		3.7
Number of 32 byte sectors/line	2		3.7
Off-chip data SRAM support	MSUG2 DDR, LW, PB2		3.7.3.9
Data path width	64 bits		
Private memory SRAM sizes	1 Mbyte, 2 Mbyte	1 Mbyte, 2 Mbytes, 4 Mbytes	3.7.3.2
Parity	Byte		3.7.3.5
L3 bus ratios	2:1, 2.5:1, 3:1, 3.5:1, 4:1, 5:1, 6:1	2:1, 2.5:1, 3:1, 3.5:1, 4:1, 5:1, 6:1, 6.5:1, 7:1, 7.5:1, 8:1	2.2.5.5.15
Signals			
L3 address signals	L3_ADDR[0:17]	L3_ADDR[0:18]	8.4.1.1
PLL configuration signals	PLL_CFG[0:4]		2.2.5.2

Table 1-5. MPC7451 and MPC7457 Differences (continued)

Microarchitectural Feature	MPC7441/MPC7451	MPC7447/MPC7457	Section
System Interface			
System bus multipliers	2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6, 6.5, 7, 7.5, 8	2, 5, 5.5, 6, 6.5, 7, 7.5, 8, 8.5, 9, 9.5, 10, 10.5, 11, 11.5, 12, 12.5, 13, 13.5, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 28, 32	2.2.5.2

[1] L3 cache interface is not supported on the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448.

1.7 Differences Between MPC7447 and MPC7447A

Table 1-6 compares the key features of the MPC7447A with the key features of the earlier MPC7445 and MPC7447. All are based on the MPC7450 RISC microprocessor and are very similar architecturally. The MPC7447A is identical to the MPC7447 but includes the DFS and temperature diode features.

Table 1-6. Microarchitecture Comparison

Microarchitectural Specs	MPC7447A	MPC7447	Section
Basic Pipeline Functions			6.2
Logic inversions per cycle	18		
Pipeline stages up to execute	5		
Total pipeline stages (minimum)	7		
Pipeline maximum instruction throughput	3 + branch		
Pipeline Resources			6.3
Instruction buffer size	12		
Completion buffer size	16		
Renames (integer, float, vector)	16, 16, 16		
Maximum Execution Throughput			6.4
SFX	3		
Vector	2 (any 2 of 4 units)		
Scalar floating-point	1		
Out-of-Order Window Size in Execution Queues			6.4
SFX integer units	1 entry × 3 queues		
Vector units	In order, 4 queues		
Scalar floating-point unit	In order		

Table 1-6. Microarchitecture Comparison (continued)

Microarchitectural Specs	MPC7447A	MPC7447	Section
Branch Processing Resources			6.6
Prediction structures	BTIC, BHT, link stack		
BTIC size, associativity	128-entry, 4-way		
BHT size	2K-entry		
Link stack depth	8		
Unresolved branches supported	3		
Branch taken penalty (BTIC hit)	1		
Minimum misprediction penalty	6		
Execution Unit Timings (Latency-Throughput)			6.4
Aligned load (integer, float, vector)	3-1, 4-1, 3-1		
Misaligned load (integer, float, vector)	4-2, 5-2, 4-2		
L1 miss, L2 hit latency	9 data/13 instruction		
SFX (aDd Sub, Shift, Rot, Cmp, logicals)	1-1		
Integer multiply (32 × 8, 32 × 16, 32 × 32)	3-1, 3-1, 4-2		
Scalar float	5-1		
VSFX (vector simple)	1-1		
VCFX (vector complex)	4-1		
VFPU (vector float)	4-1		
VPER (vector permute)	2-1		
MMUs			
TLBs (instruction and data)	128-entry, 2-way		5.1
Tablewalk mechanism	Hardware + software		5.5.2
Instruction BATs/Data BATs	8/8		5.3
L1 I Cache/D Cache Features			3.2
Size	32K/32K		
Associativity	8-way		
Locking granularity	Way		
Parity on I cache	Word		
Parity on D cache	Byte		
Number of D cache misses (load/store)	5/1		
Data stream touch engines	4 streams		
On-Chip Cache Features			

Table 1-6. Microarchitecture Comparison (continued)

Microarchitectural Specs	MPC7447A	MPC7447	Section
Cache level	L2		3.6
Size/associativity	512-Kbyte/8-way		
Access width	256 bits		
Number of 32-byte sectors/line	2		
Parity	Byte		
Thermal Control			
Dynamic frequency switching (DFS)	Yes	No	10.2.5
Thermal diode	Yes	No	10.4

1.8 Differences Between MPC7447A and MPC7448

The MPC7448 has a number of changes over the core in the MPC7447A. Some of these changes are feature improvements (larger 1-Mbyte L2 cache, expanded DFS capability, L2 data ECC). Some are performance changes: improvements (second store miss) or changes necessary for feature improvements (extended L2 pipeline). Table 1-7 describes the differences between the MPC7447A and the MPC7448.

Table 1-7. Microarchitecture Comparison

Microarchitectural Specs	MPC7447A	MPC7448	Section
Basic Pipeline Functions			6.2
Logic inversions per cycle	18		
Pipeline stages up to execute	5		
Total pipeline stages (minimum)	7		
Pipeline maximum instruction throughput	3 + branch		
Pipeline Resources			6.3
Instruction buffer size	12		
Completion buffer size	16		
Renames (integer, float, vector)	16, 16, 16		
Maximum Execution Throughput			6.4
SFX	3		
Vector	2 (any 2 of 4 units)		
Scalar floating-point	1		

Table 1-7. Microarchitecture Comparison (continued)

<table>
<tr><th>Microarchitectural Specs</th><th>MPC7447A</th><th>MPC7448</th><th>Section</th></tr>
<tr><td colspan="3">Out-of-Order Window Size in Execution Queues</td><td rowspan="4">6.4</td></tr>
<tr><td>SFX integer units</td><td colspan="2">1 entry × 3 queues</td></tr>
<tr><td>Vector units</td><td colspan="2">In order, 4 queues</td></tr>
<tr><td>Scalar floating-point unit</td><td colspan="2">In order</td></tr>
<tr><td colspan="3">Branch Processing Resources</td><td rowspan="8">6.6</td></tr>
<tr><td>Prediction structures</td><td colspan="2">BTIC, BHT, link stack</td></tr>
<tr><td>BTIC size, associativity</td><td colspan="2">128-entry, 4-way</td></tr>
<tr><td>BHT size</td><td colspan="2">2K-entry</td></tr>
<tr><td>Link stack depth</td><td colspan="2">8</td></tr>
<tr><td>Unresolved branches supported</td><td colspan="2">3</td></tr>
<tr><td>Branch taken penalty (BTIC hit)</td><td colspan="2">1</td></tr>
<tr><td>Minimum misprediction penalty</td><td colspan="2">6</td></tr>
<tr><td colspan="3">Execution Unit Timings (Latency-Throughput)</td><td rowspan="12">6.4</td></tr>
<tr><td>Aligned load (integer, float, vector)</td><td colspan="2">3-1, 4-1, 3-1</td></tr>
<tr><td>Misaligned load (integer, float, vector)</td><td colspan="2">4-2, 5-2, 4-2</td></tr>
<tr><td>L1 miss, L2 hit latency</td><td>9 data/13 instruction</td><td>11 data[1], 15/16 instruction</td></tr>
<tr><td>SFX (aDd Sub, Shift, Rot, Cmp, logicals)</td><td colspan="2">1-1</td></tr>
<tr><td>Integer multiply (32 × 8, 32 × 16, 32 × 32)</td><td colspan="2">3-1, 3-1, 4-2</td></tr>
<tr><td>Scalar float</td><td colspan="2">5-1</td></tr>
<tr><td>VSFX (vector simple)</td><td colspan="2">1-1</td></tr>
<tr><td>VCFX (vector complex)</td><td colspan="2">4-1</td></tr>
<tr><td>VFPU (vector float)</td><td colspan="2">4-1</td></tr>
<tr><td>VPER (vector permute)</td><td colspan="2">2-1</td></tr>
<tr><td colspan="3">MMUs</td><td></td></tr>
<tr><td>TLBs (instruction and data)</td><td colspan="2">128-entry, 2-way</td><td>5.1</td></tr>
<tr><td>Tablewalk mechanism</td><td colspan="2">Hardware + software</td><td>5.5.2</td></tr>
<tr><td>Instruction BATs/Data BATs</td><td colspan="2">8/8</td><td>5.3</td></tr>
</table>

Table 1-7. Microarchitecture Comparison (continued)

Microarchitectural Specs	MPC7447A	MPC7448	Section
L1 I Cache/D Cache Features			3.2
Size	32K/32K		
Associativity	8-way		
Locking granularity	Way		
Parity on I cache	Word		
Parity on D cache	Byte		
Number of D cache misses (load/store)	5/1	5/2	
Data stream touch engines	4 streams		
On-Chip Cache Features			
Cache level	L2		3.6
Size/associativity	512-Kbyte/ 8-way	1-Mbyte/ 8-way	
Access width	32 bytes	16 bytes[2]	
Number of 32-byte sectors/line	2		
Parity	Byte		
ECC	No	Yes	
Thermal Control			
Dynamic frequency switching (DFS)	Yes		10.2.5
Thermal diode	Yes		10.4

[1] 12 cycles with ECC enabled.

[2] See Section 3.1.3.2, “L2 Cache Block,” for more information.

1.9 Revision History

A list of the major differences between revisions of the *MPC7450 RISC Microprocessor Family Reference Manual*, is provided in Appendix D, “Revision History.”

Chapter 2
Programming Model

This chapter describes the MPC7450 programming model, emphasizing those features specific to the MPC7450 processor and summarizing those that are common to processors that implement the PowerPC architecture. It consists of three major sections, which describe the following:

- Registers implemented in the MPC7450
- Operand conventions
- The MPC7450 instruction set

For detailed information about architecture-defined features, see the *Programming Environments Manual* and the *AltiVec Technology Programming Environments Manual*.

2.1 AltiVec Technology and the Programming Model

AltiVec programming model features are described as follows:

- Thirty-four additional registers—32 VRs, VRSAVE, and VSCR. See Section 7.1, “AltiVec Technology and the Programming Model.”

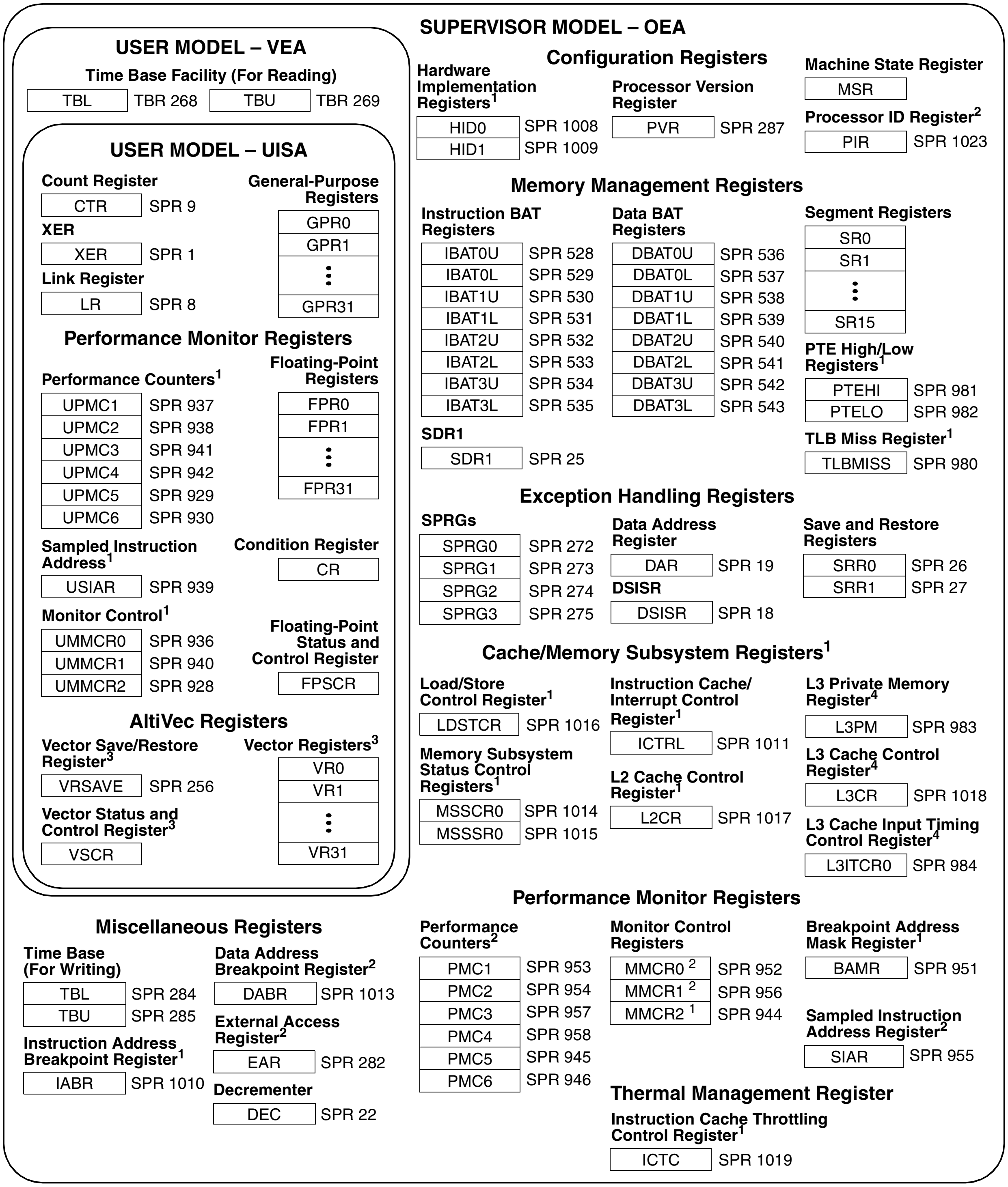
2.2 MPC7450 Processor Register Set

This section describes the registers implemented in the MPC7450. It includes an overview of registers defined by the PowerPC architecture and the AltiVec technology, highlighting differences in how these registers are implemented in the MPC7450, and a detailed description of MPC7450-specific registers. Full descriptions of the architecture-defined register set are provided in Chapter 2, “Register Set,” in the *Programming Environments Manual* and Chapter 2, “AltiVec Register Set,” in the *AltiVec Technology Programming Environments Manual* (PEM).

Registers are defined at all three levels of the PowerPC architecture—user instruction set architecture (UISA), virtual environment architecture (VEA), and operating environment architecture (OEA). The PowerPC architecture defines register-to-register operations for all computational instructions. Source data for these instructions is accessed from the on-chip registers or provided as immediate values 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.

2.2.1 Register Set Overview

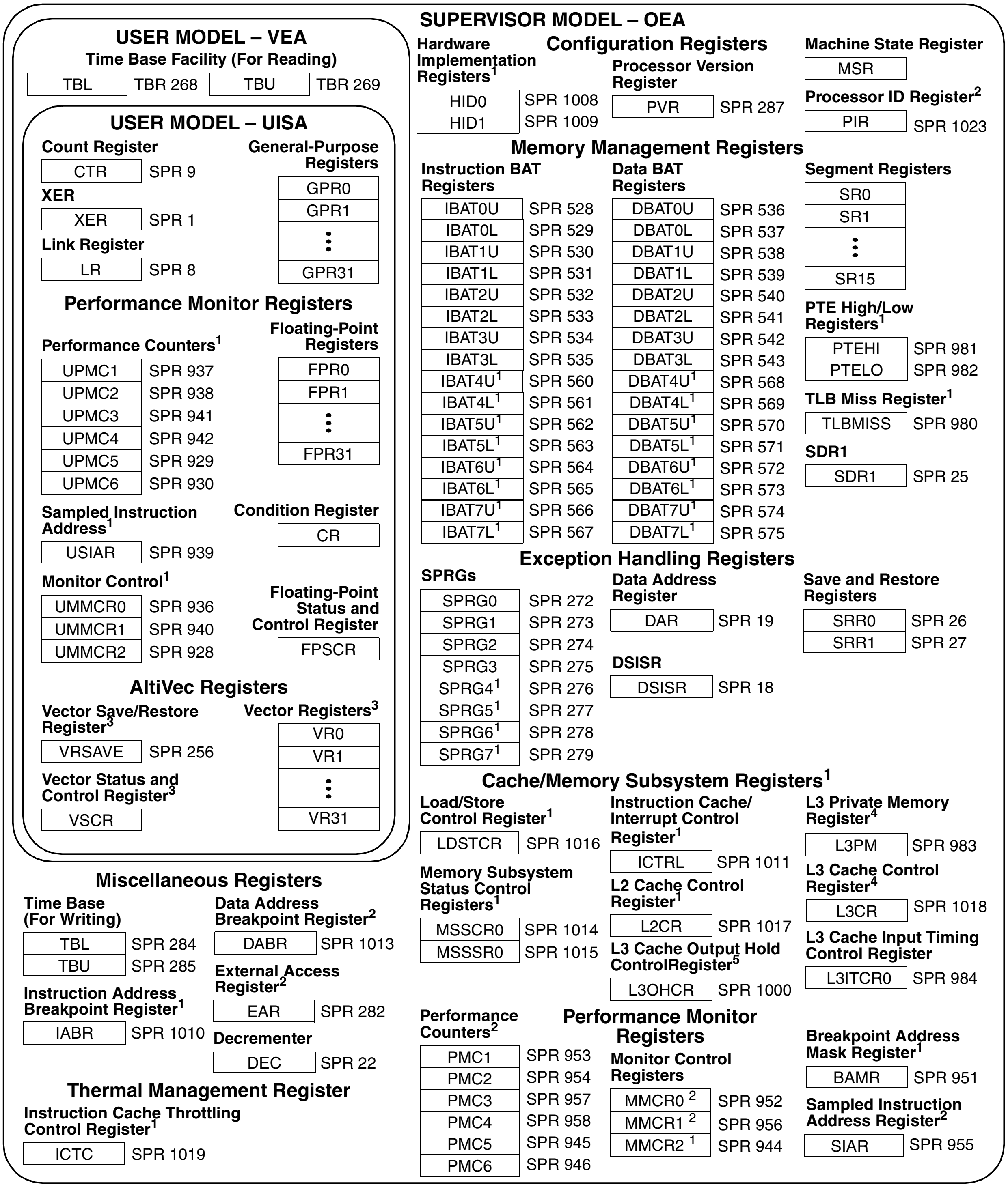
Figure 2-1 shows the MPC7441 and MPC7451 register set.



[1] MPC7441-, MPC7451-specific register may not be supported on other processors that implement the PowerPC architecture.
[2] Register defined as optional in the PowerPC architecture.
[3] Register defined by the AltiVec technology.
[4] MPC7451-specific register.

Figure 2-1. Programming Model—MPC7441/MPC7451 Microprocessor Registers

Figure 2-2 shows the MPC7445, MPC7447, MPC7455, and MPC7457 register set.



[1] MPC7445-, MPC7447-, MPC7455-, and MPC7457-specific register may not be supported on other processors that implement the PowerPC architecture.

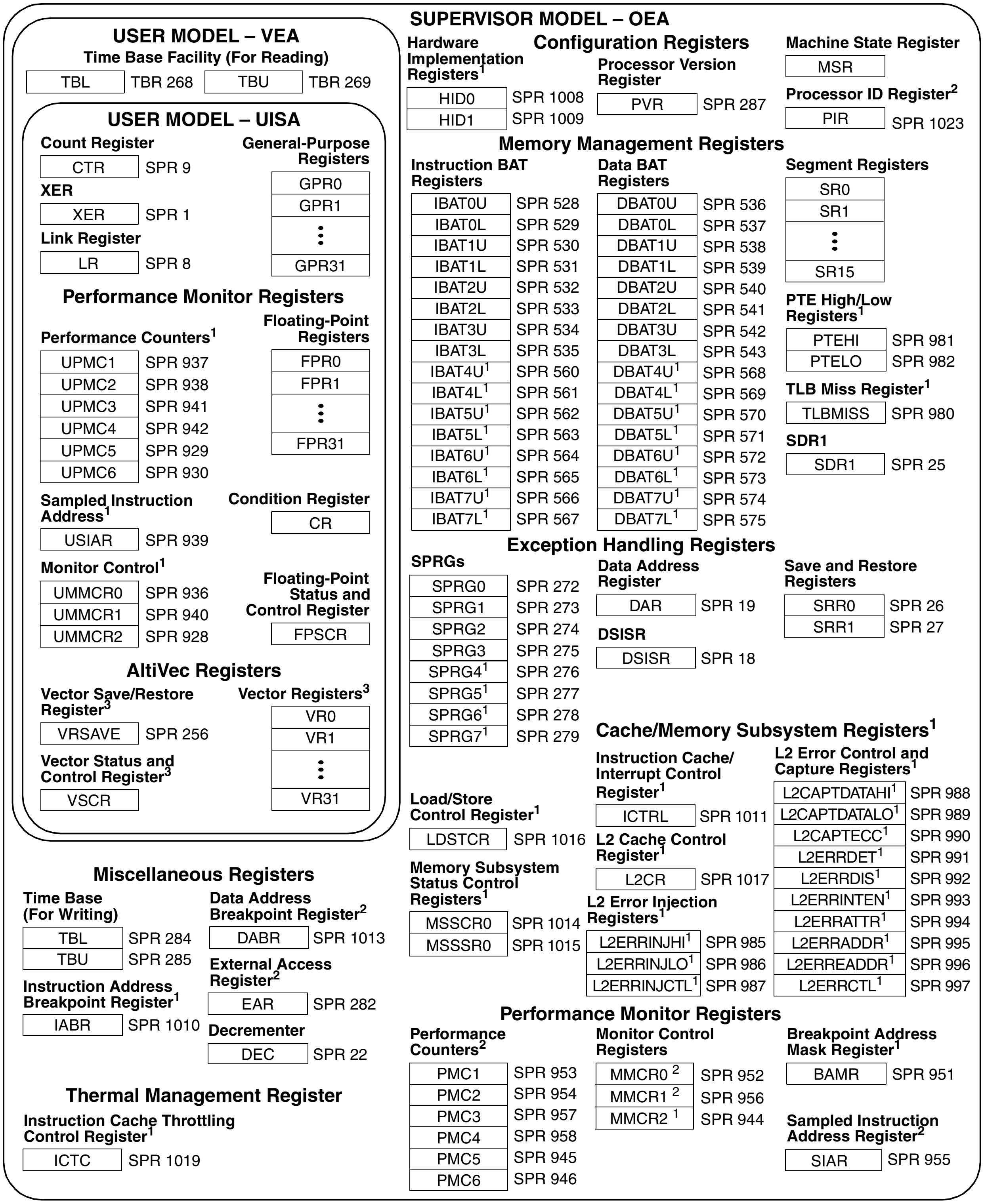
[2] Register defined as optional in the PowerPC architecture.

[3] Register defined by the AltiVec technology.

[4] MPC7455- and MPC7457-specific register.

Figure 2-2. Programming Model—MPC7445, MPC7447, MPC7455, MPC7457, and MPC7447A Microprocessor Registers

Figure 2-3 shows the MPC7448 register set.



[1] MPC7448-specific register may not be supported on other processors that implement the PowerPC architecture.
[2] Register defined as optional in the PowerPC architecture.
[3] Register defined by the AltiVec technology.

Figure 2-3. Programming Model—MPC7448 Microprocessor Registers

The number to the right of the special-purpose registers (SPRs) is the number used in the syntax of the instruction operands to access the register (for example, the number used to access the XER register is SPR 1). These registers can be accessed using **mtspr** and **mfspr**. Note that not all registers in Figure 2-1 are SPRs, for example VSCR and VRs are AltiVec registers and do not have an SPR number.

2.2.2 MPC7450 Register Set

Table 2-1 summarizes the registers implemented in the MPC7450.

Table 2-1. Register Summary for the MPC7450

Name	SPR	Description	Reference / Section
UISA Registers			
CR	—	Condition register. The 32-bit CR consists of eight 4-bit fields, CR0–CR7, that reflect results of certain arithmetic operations and provide a mechanism for testing and branching.	PEM
CTR	9	Count register. Holds a loop count that can be decremented during execution of appropriately coded branch instructions. The CTR can also provide the branch target address for the Branch Conditional to Count Register (**bcctr***x*) instruction.	PEM
FPR0–FPR31	—	Floating-point registers (FPR*n*). The 32 FPRs serve as the data source or destination for all floating-point instructions.	PEM
FPSCR	—	Floating-point status and control register. Contains floating-point exception signal bits, exception summary bits, exception enable bits, and rounding control bits for compliance with the IEEE 754 standard.	PEM
GPR0–GPR31	—	General-purpose registers (GPR*n*). The thirty-two GPRs serve as data source or destination registers for integer instructions and provide data for generating addresses.	PEM
LR	8	Link register. Provides the branch target address for the Branch Conditional to Link Register (**bclr***x*) instruction, and can be used to hold the logical address of the instruction that follows a branch and link instruction, typically used for linking to subroutines.	PEM
UMMCR0 [1] UMMCR1 [1] UMMCR2 [1]	936 940 928	User monitor mode control registers (UMMCR*n*). Used to enable various performance monitor exception functions. UMMCRs provide user-level read access to MMCR registers.	2.2.5.9.2 & 11.3.2.1, 2.2.5.9.4 & 11.3.3.1, 2.2.5.9.6 & 11.3.4.1
UPMC1–UPMC6 [1]	937, 938 941, 942 929, 930	User performance monitor counter registers (UPMC*n*). Used to record the number of times a certain event has occurred. UPMCs provide user-level read access to PMC registers.	2.2.5.9.9, 11.3.6.1
USIAR [1]	939	User sampled instruction address register. Contains the effective address of an instruction executing at or around the time that the processor signals the performance monitor exception condition. USIAR provides user-level read access to the SIAR.	2.2.5.9.11, 11.3.7.1

Table 2-1. Register Summary for the MPC7450 (continued)

Name	SPR	Description	Reference / Section
VR0–VR31 [2]	—	Vector registers (VR*n*). Data source and destination registers for all AltiVec instructions.	7.1.1.4
VRSAVE [2]	256	Vector save/restore register. Defined by the AltiVec technology to assist application and operating system software in saving and restoring the architectural state across process context-switched events. The register is maintained only by software to track live or dead information on each AltiVec register.	7.1.1.5
VSCR [2]	—	Vector status and control register. A 32-bit vector register that is read and written in a manner similar to the FPSCR.	7.1.1.4
XER	1	Indicates overflows and carries for integer operations. **Implementation Note**—To emulate the POWER architecture **lscbx** instruction, XER[16–23] are be read with **mfspr**[XER] and written with **mtspr**[XER].	PEM
VEA			
TBL, TBU (For reading)	TBR 268 TBR 269	Time base facility. Consists of two 32-bit registers, time base lower and upper registers (TBL/TBU). TBL (TBR 268) and TBU (TBR 269) can only be read from and not written to.TBU and TBL can be read with the move from time base register (**mftb**) instruction. **Implementation Note**—Reading from SPR 284 or 285 using the **mftb** instruction causes an illegal instruction exception.	PEM 2.2.4.1 2.4.5.1
OEA			
BAMR [1]	951	Breakpoint address mask register. Used in conjunction with the events that monitor IABR hits. **Note:** See Table 2-46 for specific synchronization requirements on this register.	2.2.5.9.7, 11.3.5
DABR [3]	1013	Data address breakpoint register. Optional register implemented in the MPC7450 and is used to cause a breakpoint exception if a specified data address is encountered. See Table 2-46 for specific synchronization requirements on this register.	PEM
DAR	19	Data address register. After a DSI or alignment exception, DAR is set to the effective address (EA) generated by the faulting instruction.	PEM
DEC	22	Decrementer register. A 32-bit decrementer counter used with the decrementer exception. **Implementation Note**—In the MPC7450, DEC is decremented and the time base increments at 1/4 of the system bus clock frequency.	PEM
DSISR	18	DSI source register. Defines the cause of DSI and alignment exceptions.	PEM

Table 2-1. Register Summary for the MPC7450 (continued)

Name	SPR	Description	Reference / Section
EAR	282	External access register. Used with **eciwx** and **ecowx**. Note that the EAR and the **eciwx** and **ecowx** instructions are optional in the PowerPC architecture. See Table 2-46 for specific synchronization requirements on this register.	PEM
HID0 [1] HID1 [1]	1008, 1009	Hardware implementation-dependent registers. Control various functions, such as the power management features, and locking, enabling, and invalidating the instruction and data caches. The HID1 includes bits that reflects the state of PLL_CFG[0:4] (PLL_CFG[0:5] in the MPC7448) clock signals and control other bus-related functions. See Table 2-46 for specific synchronization requirements on this register.	2.2.5.1, 2.2.5.2
IABR [1]	1010	Instruction address breakpoint register. Used to cause a breakpoint exception if a specified instruction address is encountered. See Table 2-46 for specific synchronization requirements on this register.	2.2.5.6
IBAT0U/L IBAT1U/L IBAT2U/L IBAT3U/L IBAT4U/L [4] IBAT5U/L [4] IBAT6U/L [4] IBAT7U/L [4] DBAT0U/L DBAT1U/L DBAT2U/L DBAT3U/L DBAT4U/L [4] DBAT5U/L [4] DBAT6U/L [4] DBAT7U/L [4]	528, 529 530, 531 532, 533 534, 535 560, 561 562, 563 564, 565 566, 567 536, 537 538, 539 540, 541 542, 543 568, 569 570, 571 572, 573 574, 575	Block-address translation (BAT) registers. The PowerPC OEA includes an array of block address translation registers that can be used to specify four blocks of instruction space and four blocks of data space. The BAT registers are implemented in pairs: four pairs of instruction BATs (IBAT0U–IBAT3U and IBAT0L–IBAT3L) and four pairs of data BATs (DBAT0U–DBAT3U and DBAT0L–DBAT3L). Sixteen additional BAT registers have been added for the MPC7455. These registers are enabled by setting HID0[HIGH_BAT_EN]. When HID0[HIGH_BAT_EN] = 1, the 16 additional BAT registers, organized as four pairs of instruction BAT registers(IBAT4U–IBAT7U paired with IBAT4L–IBAT7L) and four pairs of data BAT registers (DBAT4U–DBAT7U paired with DBAT4L–DBAT7L) are available. Thus, the MPC7455 can define a total of 16 blocks implemented as 32 BAT registers. Because BAT upper and lower words are loaded separately, software must ensure that BAT translations are correct during the time that both BAT entries are being loaded. The MPC7450 implements IBAT[G]; however, attempting to execute code from an IBAT area with G = 1 causes an ISI exception. See Table 2-46 for specific synchronization requirements on this register.	PEM, 5.1.3
ICTC [1]	1019	Instruction cache throttling control register. Has bits for enabling instruction cache throttling and for controlling the interval at which instructions are fetched. This controls overall junction temperature.	2.2.5.8.1, 10.3

Table 2-1. Register Summary for the MPC7450 (continued)

Name	SPR	Description	Reference / Section
ICTRL [1]	1011	Instruction cache and interrupt control register. Used in configuring interrupts and error reporting for the instruction and data caches. See Table 2-46 for specific synchronization requirements on this register.	2.2.5.5.21
L2CR [1]	1017	L2 cache control register. Includes bits for enabling parity checking, setting the L2 cache size, and flushing and invalidating the L2 cache.	2.2.5.5.1
L2ERRINJHI [5] L2ERRINJLO [5] L2ERRINJCTL [5] L2CAPTDATAHI [5] L2CAPTDATALO [5] L2CAPTDATAECC [5] L2ERRDET [5] L2ERRDIS [5] L2ERRINTEN [5] L2ERRATTR [5] L2ERRADDR [5] L2ERREADDR [5] L2ERRCTL [5]	985 986 987 988 989 990 991 992 993 994 995 996 997	L2 error registers. The L2 cache supports error injection into the L2 data, data ECC or tag, which can be used to test error recovery software by deterministically creating error scenarios. L2ERRINJHI, L2ERRINJLO, and L2ERRINJCTL are error injection registers. The error control and capture registers control the detection and reporting of tag parity and ECC errors.	2.2.5.5.2 2.2.5.5.3 2.2.5.5.4 2.2.5.5.5 2.2.5.5.6 2.2.5.5.7 2.2.5.5.8 2.2.5.5.9 2.2.5.5.10 2.2.5.5.11 2.2.5.5.12 2.2.5.5.13 2.2.5.5.14
L3CR [6]	1018	L3 cache control register. Includes bits for enabling parity checking, setting the L3-to-processor clock ratio, and identifying the type of RAM used for the L3 cache implementation.	2.2.5.5.15
L3ITCR0 [6] L3ITCR1 [7] L3ITCR2 [7] L3ITCR3 [7]	984 1001 1002 1003	L3 cache input timing control register. Includes bits for controlling the input AC timing of the L3 cache interface.	2.2.5.5.17 2.2.5.5.18 2.2.5.5.19 2.2.5.5.20
L3OHCR [7]	1000	L3 cache output hold control register. Includes bits for controlling the output AC timing of the L3 cache interface of the MPC7457.	2.2.5.5.16
L3PM [6]	983	The L3 private memory register. Configures the base address of the range of addresses that the L3 uses as private memory (not cache). See Table 2-46 for specific synchronization requirements on this register.	2.2.5.5.23
LDSTCR [1]	1016	Load/store control register. Controls data L1 cache way-locking. See Table 2-46 for specific synchronization requirements on this register.	2.2.5.5.22
MMCR0 [3] MMCR1 [3] MMCR2 [1]	952 956 944	Monitor mode control registers (MMCR*n*). Enable various performance monitor exception functions. UMMCR0–UMMCR2 provide user-level read access to these registers.	2.2.5.9.1, 11.3.2 2.2.5.9.3, 11.3.3 2.2.5.9.5, 11.3.4

Table 2-1. Register Summary for the MPC7450 (continued)

<table>
<tr><th>Name</th><th>SPR</th><th>Description</th><th>Reference / Section</th></tr>
<tr><td>MSR</td><td>—</td><td>Machine state register. Defines the processor state. The MSR can be modified by the mtmsr, sc, and rfi instructions. It can be read by the mfmsr instruction. When an exception is taken, MSR contents are saved to SRR1. See Section 4.3, “Exception Processing.” The following bits are optional in the PowerPC architecture.
Note that setting MSR[EE] masks decrementer and external interrupt exceptions and MPC7450-specific system management, and performance monitor exceptions.
See Table 2-46 for specific synchronization requirements on this register.
<table>
<tr><th>Bit</th><th>Name</th><th>Description</th></tr>
<tr><td>6</td><td>VEC</td><td>AltiVec available. MPC7450 and AltiVec technology specific; optional to the PowerPC architecture.
0 AltiVec technology is disabled.
1 AltiVec technology is enabled.
Note: When a non-stream AltiVec instruction accesses VRs or the VSCR when VEC = 0 an AltiVec unavailable exception is generated. This does not occur for data streaming instructions (dst(t), dstst(t), and dss); the VRs and the VSCR are available to data streaming instructions even if VEC = 0. VRSAVE can be accessed even if VEC = 0.</td></tr>
<tr><td>13</td><td>POW</td><td>Power management enable. MPC7450-specific and optional to the PowerPC architecture.
0 Power management is disabled.
1 Power management is enabled. The processor can enter a power-saving mode determined by HID0[NAP,SLEEP] when additional conditions are met. See Table 2-7.</td></tr>
<tr><td>29</td><td>PMM</td><td>Performance monitor marked mode. MPC7450-specific and optional to the PowerPC architecture. See Chapter 11, “Performance Monitor.”
0 Process is not a marked process.
1 Process is a marked process.</td></tr>
</table>
</td><td>PEM,
2.2.3.4,
4.3</td></tr>
<tr><td>MSSCR0 [1]</td><td>1014</td><td>Memory subsystem control register. Used to configure and operate many aspects of the memory subsystem.</td><td>2.2.5.3</td></tr>
<tr><td>MSSSR0 [1]</td><td>1015</td><td>Memory subsystem status register. Used to configure and operate the parity functions in the L2 and L3 caches for the MPC7450.
See Table 2-46 for specific synchronization requirements on this register.</td><td>2.2.5.4</td></tr>
</table>

Table 2-1. Register Summary for the MPC7450 (continued)

Name	SPR	Description	Reference / Section
PIR	1023	Processor identification register. Provided for system use. All 32 bits of the PIR can be written to with the **mtspr** instruction.	PEM 2.2.3.3
PMC1– PMC6 [3]	953, 954 957, 958 945, 946	Performance monitor counter registers (PMC*n*). Used to record the number of times a certain event has occurred. UPMCs provide user-level read access to these registers.	2.2.5.9.8, 11.3.6
PTEHI, PTELO	981, 982	The PTEHI and PTELO registers are used by the **tlbld** and **tlbli** instructions to create a TLB entry. When software table searching is enabled (HID0[STEN] = 1), and a TLB miss exception occurs, the bits of the page table entry (PTE) for this access are located by software and saved in the PTE registers.	2.2.5.7.2, 5.5.5.1.2
PVR	287	Processor version register. Read-only register that identifies the version (model) and revision level of the processor.	PEM, 2.2.3.1
SDAR, USDAR	—	Sampled data address register. The MPC7450 does not implement the optional registers (SDAR or the user-level, read-only USDAR register) defined by the PowerPC architecture. Note that in previous processors the SDA and USDA registers could be written to by boot code without causing an exception, this is not the case in the MPC7450. A **mtspr** or **mfspr** SDAR or USDAR instruction causes a program exception**.**	2.2.5.9.12
SDR1	25	Sample data register. Specifies the base address of the page table entry group (PTEG) address used in virtual-to-physical address translation. **Implementation Note**—The SDR1 register has been modified (with the SDR1[HTABEXT] and SDR1[HTMEXT] fields) for the MPC7450 to support the extended 36-bit physical address (when HID0[XAEN] = 1]). See Table 2-46 for specific synchronization requirements on this register.	PEM, 2.2.3.6, 5.5.1
SIAR [3]	955	Sampled instruction address register. *C*ontains the effective address of an instruction executing at or around the time that the processor signals the performance monitor exception condition. USIAR provides user-level read access to the SIAR.	2.2.5.9.10 11.3.7
SPRG0– SPRG3 SPRG4– SPRG7 [4]	272–275 276–279	SPRG*n*. Provided for operating system use. The SPRG4–7 provide additional registers to be used by system software for software table searching.	PEM, 5.5.5.1.3
SR0– SR15	—	Segment registers (SR*n*). Note that the MPC7450 implements separate instruction and data MMUs. It associates architecture-defined SRs with the data MMU. It reflects SRs values in separate, shadow SRs in the instruction MMU. See Table 2-46 for specific synchronization requirements on this register.	PEM

Table 2-1. Register Summary for the MPC7450 (continued)

Name	SPR	Description	Reference / Section
SRR0 SRR1	26 27	Machine status save/restore registers (SRR*n*). Used to save the address of the instruction at which execution continues when **rfi** executes at the end of an exception handler routine. SRR1 is used to save machine status on exceptions and to restore machine status when **rfi** executes. **Implementation Note**—When a machine check exception occurs, the MPC7450 sets one or more error bits in SRR1. Refer to the individual exceptions for individual SRR1 bit settings.	PEM, 2.2.3.5, 4.3
SVR [5]	286	System version register. Read-only register provided for future product compatibility.	2.2.3.2
TBL TBU (For writing)	284 285	Time base. A 64-bit structure (two 32-bit registers) that maintains the time of day and operating interval timers. The TB consists of two registers—time base upper (TBU) and time base lower (TBL). The time base registers can be written to only by supervisor-level software. TBL (SPR 284) and TBU (SPR 285) can only be written to and not read from. TBL and TBU can be written to, with the move to special purpose register (**mtspr**) instruction. **Implementation Note**—Reading from SPR 284 or 285 causes an illegal instruction exception.	PEM 2.2.4.1 2.4.5.1
TLBMISS [1]	980	The TLBMISS register is automatically loaded when software searching is enabled (HID0[STEN] = 1) and a TLB miss exception occurs. Its contents are used by the TLB miss exception handlers (the software table search routines) to start the search process.	2.2.5.7.1 5.5.5.1.1

[1] MPC7441-, MPC7445-, MPC7447- MPC7447A-, MPC7448-, MPC7451-, MPC7455-, and MPC7457-specific register that may not be supported on other processors that implement the PowerPC architecture.

[2] Register is defined by the AltiVec technology.

[3] Defined as optional register in the PowerPC architecture.

[4] MPC7445-, MPC7447-, MPC7447A-, MPC7448-, MPC7455-, and MPC7457-specific register.

[5] MPC7448-specific register.

[6] MPC7451-, MPC7455-, and MPC7457-specific register

[7] MPC7457-specific register.

The PowerPC UISA registers are user-level. General-purpose registers (GPRs), floating-point registers (FPRs) and vector registers (VRs) are accessed through instruction operands. Access to registers can be explicit (by using instructions for that purpose such as Move to Special-Purpose Register (**mtspr**) and Move from Special-Purpose Register (**mfspr**) instructions) or implicit as part of the execution of an instruction. Some registers are accessed both explicitly and implicitly.

> **Implementation Note**—The MPC7450 fully decodes the SPR field of the instruction. If the SPR specified is undefined, an illegal instruction program exception occurs.

2.2.3 PowerPC Supervisor-Level Registers (OEA)

The OEA defines the registers an operating system uses for memory management, configuration, exception handling, and other operating system functions and they are summarized in Table 2-1. The following supervisor-level register defined by the PowerPC architecture contains additional implementation-specific information for the MPC7450.

2.2.3.1 Processor Version Register (PVR)

For more information, see “Processor Version Register (PVR),” in Chapter 2, “PowerPC Register Set,” of the *Programming Environments Manual*. Table 2-2 shows the PVR settings for each device. The revision level is updated for each silicon revision.

Table 2-2. PVR Settings

Part No.	Processor Version No.	Starting Processor Revision Level
MPC7451/MPC7441	0x8000	0200
MPC7455/MPC7445	0x8001	0100
MPC7457/MPC7447	0x8002	0100
MPC7447A	0x8003	0100
MPC7448	0x8004	0100

Table 2-3 describes the MPC7450 PVR bits that are not required by the PowerPC architecture.

Table 2-3. Additional PVR Bits

Bits	Name	Description
0–15	Type	Processor type
16–19	Tech	Processor technology
20–23	Major	Major revision number
24–31	Minor	Minor revision number

2.2.3.2 System Version Register (SVR)—MPC7448 Specific

SVR is a read-only register provided on the MPC7448. It is provided for future product compatibility. On the MPC7448, SVR always reads as 0.

2.2.3.3 Processor Identification Register (PIR)

For more information, see “Processor Identification Register (PIR),” in Chapter 2, “PowerPC Register Set,” of the *Programming Environments Manual*.

> **Implementation Note**—The MPC7450 provides write access to the PIR with **mtspr** using SPR 1023.

2.2.3.4 Machine State Register (MSR)

The MSR defines the state of the processor. When an exception occurs, MSR bits, as described in Table 2-4 are altered as determined by the exceptions. The MSR can also be modified by the **mtmsr**, **sc**, and **rfi** instructions. It can be read by the **mfmsr** instruction.

The MPC7450 MSR is shown in Figure 2-4.

▨ Reserved

0000_0	VEC	00_0000	POW	0	ILE	EE	PR	FP	ME	FE0	SE	BE	FE1	0	IP	IR	DR	0	PMM	RI	LE
0 5	6	7 12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Figure 2-4. Machine State Register (MSR)

The MSR bits are defined in Table 2-4.

Table 2-4. MSR Bit Settings

Bit(s)	Name	Description
0–5	—	Reserved
6	VEC[1, 2]	AltiVec vector unit available 0 The processor prevents dispatch of AltiVec instructions (excluding the data streaming instructions—**dst**, **dstt**, **dstst**, **dststt**, **dss**, and **dssall**). The processor also prevents access to the vector register file (VRF) and the vector status and control register (VSCR). Any attempt to execute an AltiVec instruction that accesses the VRF or VSCR, excluding the data streaming instructions generates the AltiVec unavailable exception. The data streaming instructions are not affected by this bit; the VRF and VSCR registers are available to the data streaming instructions even when the MSR[VEC] is cleared. 1 The processor can execute AltiVec instructions and the VRF and VSCR registers are accessible to all AltiVec instructions. Note that the VRSAVE register is not protected by MSR[VEC].
7–12	—	Reserved
13	POW [1, 3]	Power management enable 0 Power management disabled (normal operation mode) 1 Power management enabled (reduced power mode) Power management functions are implementation-dependent. See Chapter 10, “Power and Thermal Management.”
14	—	Reserved. Implementation-specific
15	ILE	Exception little-endian mode. When an exception occurs, this bit is copied into MSR[LE] to select the endian mode for the context established by the exception.
16	EE	External interrupt enable 0 The processor delays recognition of external interrupts and decrementer exception conditions. 1 The processor is enabled to take an external interrupt or the decrementer exception.
17	PR [4]	Privilege level 0 The processor can execute both user- and supervisor-level instructions. 1 The processor can only execute user-level instructions.

Table 2-4. MSR Bit Settings (continued)

Bit(s)	Name	Description
18	FP [2]	Floating-point available 0 The processor prevents dispatch of floating-point instructions, including floating-point loads, stores, and moves. 1 The processor can execute floating-point instructions and can take floating-point enabled program exceptions.
19	ME	Machine check enable 0 Machine check exceptions are disabled. 1 Machine check exceptions are enabled.
20	FE0 [2]	IEEE floating-point exception mode 0 (see Table 2-5)
21	SE	Single-step trace enable 0 The processor executes instructions normally. 1 The processor generates a single-step trace exception upon the successful execution of every instruction except **rfi** and **sc**. Successful execution means that the instruction caused no other exception.
22	BE	Branch trace enable 0 The processor executes branch instructions normally. 1 The processor generates a branch type trace exception when a branch instruction executes successfully.
23	FE1 [2]	IEEE floating-point exception mode 1 (see Table 2-5)
24	—	Reserved. This bit corresponds to the AL bit of the POWER architecture.
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. 0 Exceptions are vectored to the physical address 0x000*n_nnnn*. 1 Exceptions are vectored to the physical address 0xFFF*n_nnnn*.
26	IR [5]	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 [4]	Data address translation 0 Data address translation is disabled. 1 Data address translation is enabled. For more information see Chapter 5, “Memory Management.”
28	—	Reserved

Table 2-4. MSR Bit Settings (continued)

Bit(s)	Name	Description
29	PMM [1]	Performance monitor marked mode 0 Process is not a marked process. 1 Process is a marked process. This bit can be set when statistics need to be gathered on a specific (marked) process. The statistics will only be gathered when the marked process is executing. MPC7451-specific; defined as optional by the PowerPC architecture. For more information about the performance monitor marked mode bit, see Section 11.4, "Event Counting."
30	RI	Indicates whether system reset or machine check exception is recoverable. 0 Exception is not recoverable. 1 Exception is recoverable. The RI bit indicates whether from the perspective of the processor, it is safe to continue (that is, processor state data such as that saved to SRR0 is valid), but it does not guarantee that the interrupted process is recoverable.
31	LE [6]	Little-endian mode enable 0 The processor runs in big-endian mode. 1 The processor runs in little-endian mode.

[1] Optional to the PowerPC architecture.

[2] A context synchronizing instruction must follow a mtmsr instruction.

[3] A dssall and sync must precede a mtmsr instruction and then a context synchronizing instruction must follow.

[4] A dssall and sync must precede a mtmsr and then a sync and context synchronizing instruction must follow. Note that if a user is not using the AltiVec data streaming instructions, then a dssall is not necessary prior to accessing the MSR[DR] or MSR[PR] bit.

[5] A context synchronizing instruction must follow a mtmsr. When changing the MSR[IR] bit the context synchronizing instruction must reside at both the untranslated and the translated address following the mtmsr.

[6] A dssall and sync must precede an rfi to guarantee a solid context boundary. Note that if a user is not using the AltiVec data streaming instructions, then a dssall is not necessary prior to accessing the MSR[LE] bit.

Note that setting MSR[EE] masks not only the architecture-defined external interrupt and decrementer exceptions but also the MPC7450-specific system management, and performance monitor exceptions.

The IEEE floating-point exception mode bits (FE0 and FE1) together define whether floating-point exceptions are handled precisely, imprecisely, or whether they are taken at all. As shown in Table 2-5, if either FE0 or FE1 are set, the MPC7450 treats exceptions as precise. MSR bits are guaranteed to be written to SRR1 when the first instruction of the exception handler is encountered. For further details, see Chapter 2, "Register Set" and Chapter 6, "Exceptions," of the *Programming Environments Manual*.

Table 2-5. IEEE Floating-Point Exception Mode Bits

FE0	FE1	Mode
0	0	Floating-point exceptions disabled
0	1	Imprecise nonrecoverable. For this setting, the MPC7450 operates in floating-point precise mode.
1	0	Imprecise recoverable. For this setting, the MPC7450 operates in floating-point precise mode.
1	1	Floating-point precise mode

2.2.3.5 Machine Status Save/Restore Registers (SRR0, SRR1)

When an exception is taken, the processor uses SRR0 and SRR1 to save the contents of the MSR for the current context and to identify where instruction execution should resume after the exception is handled.

When an exception occurs, the address saved in SRR0 helps determine where instruction processing should resume when the exception handler returns control to the interrupted process. Depending on the exception, this may be the address in SRR0 or at the next address in the program flow. All instructions in the program flow preceding this one will have completed execution and no subsequent instruction will have begun execution. This may be the address of the instruction that caused the exception or the next one (as in the case of a system call or trace exception). The SRR0 register is shown in Figure 2-5.

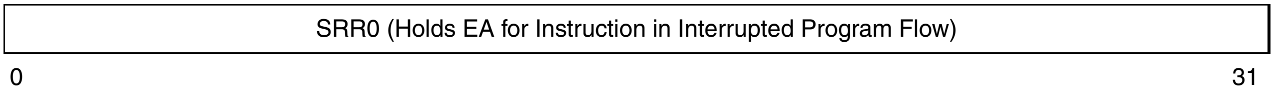


Figure 2-5. Machine Status Save/Restore Register 0 (SRR0)

SRR1 is used to save machine status (selected MSR bits and possibly other status bits) on exceptions and to restore those values when an **rfi** instruction is executed. SRR1 is shown in Figure 2-6.

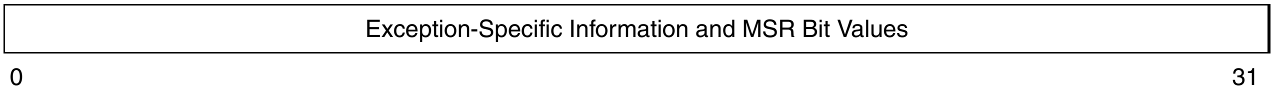


Figure 2-6. Machine Status Save/Restore Register 1 (SRR1)

Typically, when an exception occurs, SRR1[0–15] are loaded with exception-specific information and MSR[16–31] are placed into the corresponding bit positions of SRR1. For most exceptions, SRR1[0–5] and SRR1[7–15] are cleared, and MSR[6, 16–31] are placed into the corresponding bit positions of SRR1. Table 2-4 provides a summary of the SRR1 bit settings when a machine check exception occurs. For a specific exception's SRR1 bit settings, see Section 4.6, "Exception Definitions."

2.2.3.6 SDR1 Register

The SDR1 register specifies the page table entry group (PTEG) address used in virtual-to-physical address translation. See "SDR1," in Chapter 2, "PowerPC Register Set," of the *Programming Environments Manual* for the description with a 32-bit physical address. The SDR1 register has been modified for the MPC7450 to support the extended 36-bit physical address (when HID0[XAEN] = 1]). See Section 5.5.1, "SDR1 Register Definition—Extended Addressing," for details on how SDR1 is modified to support a 36-bit physical address.

> **Implementation Note**—SDR1[HTABEXT] and SDR1[HTMEXT] fields have been added to support extended addressing. Section 5.5.1, "SDR1 Register Definition—Extended Addressing," describes in detail the differences when generating a 36-bit PTEG address. Figure 2-7 shows the format of the modified SDR1.

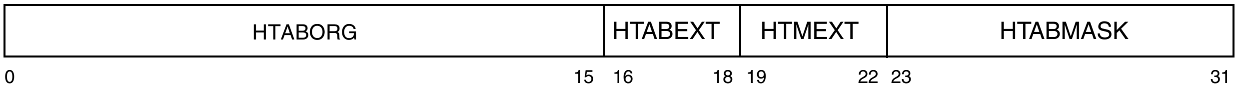


Figure 2-7. SDR1 Register Format—Extended Addressing

Bit settings for the SDR1 register are described in Table 2-6.

Table 2-6. SDR1 Register Bit Settings—Extended Addressing

Bits	Name	Description
0–15	HTABORG	Physical base address of page table If HID0[XAEN] = 1, field contains physical address [4–19] If HID0[XAEN] = 0, field contains physical address [0–15]
16–18	HTABEXT	Extension bits for physical base address of page table If HID0[XAEN] = 1, field contains physical address [1–3] If HID0[XAEN] = 0, field is reserved
19–22	HTMEXT	Hash table mask extension bits If HID0[XAEN] = 1, field contains hash table mask [0–3] If HID0[XAEN] = 0, field is reserved
23–31	HTABMASK	Mask for page table address If HID0[XAEN] = 1, field contains hash table mask [4–12] If HID0[XAEN] = 0, field contains hash table mask [0–7]

SDR1 can be accessed with **mtspr** and **mfspr** using SPR 25. For synchronization requirements on the register see Section 2.4.2.4, “Synchronization.”

2.2.4 PowerPC User-Level Registers (VEA)

The PowerPC VEA defines the time base facility (TB), which consists of two 32-bit registers—time base upper (TBU) and time base lower (TBL).

2.2.4.1 Time Base Registers (TBL, TBU)

The time base registers can be written only by supervisor-level instructions but can be read by both user- and supervisor-level software. The time base registers have two different addresses. TBU and TBL can be read from the TBR 268 and 269, respectively, with the move from time base register (**mftb**) instruction. TBU and TBL can be written to TBR 284 and 285, respectively, with the move to special purpose register (**mtspr**) instruction. Reading from SPR 284 or 285 causes an illegal instruction exception. For more information, see “PowerPC VEA Register Set—Time Base,” in Chapter 2, “PowerPC Register Set,” of the *Programming Environments Manual.*

2.2.5 MPC7450-Specific Register Descriptions

The PowerPC architecture allows for implementation-specific SPRs. This section describes registers that are defined for the MPC7450 but are not included in the PowerPC architecture. Note that in the MPC7450, these registers are all supervisor-level registers. All of the registers described in the *AltiVec Technology*

Programming Environments Manual are implemented in MPC7450, see Chapter 2, "AltiVec Register Set," for details about these registers.

Note that while it is not guaranteed that the implementation of MPC7450-specific registers is consistent among processors that implement the PowerPC architecture, other processors can implement similar or identical registers.

The registers in the following subsections are presented in the order of the chapters in this book. First, the processor control registers are described followed by the cache control registers. Next, the implementation-specific registers for exception processing and memory management are presented, followed by the thermal management register. The performance monitor registers are presented last.

2.2.5.1 Hardware Implementation-Dependent Register 0 (HID0)

The hardware implementation-dependent register 0 (HID0) controls the state of several functions within the MPC7450. The HID0 register for the MPC7441 and the MPC7451 is shown in Figure 2-8.

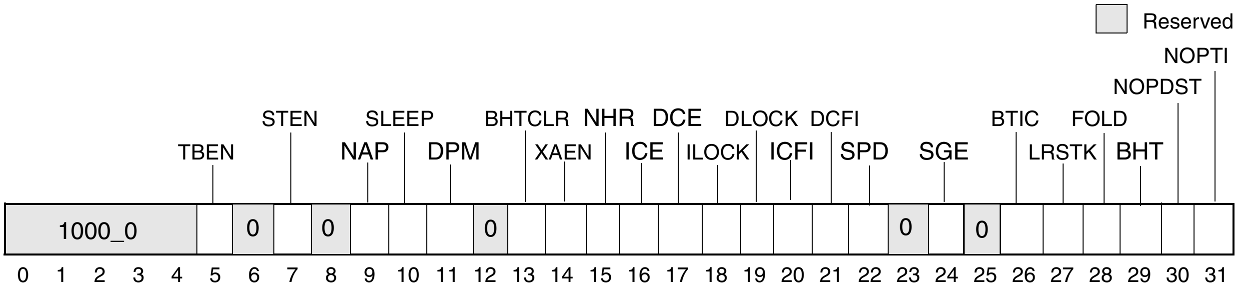


Figure 2-8. Hardware Implementation-Dependent Register 0 (HID0) for MPC7441 and MPC7451

The HID0 register for the MPC7445, MPC7455, MPC7457, MPC7447, MPC7447A, and MPC7448 is shown in Figure 2-9.

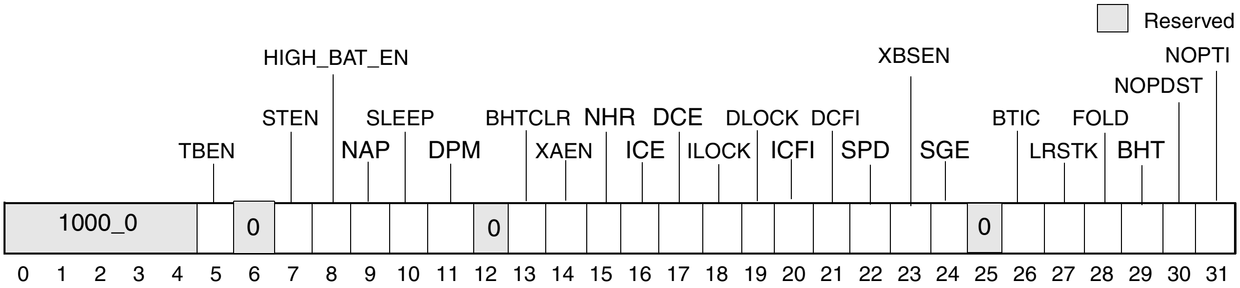


Figure 2-9. Hardware Implementation-Dependent Register 0 (HID0) for MPC7445, MPC7455, MPC7457, MPC7447, MPC7447A, and MPC7448

The HID0 bits are described in Table 2-7.

Table 2-7. HID0 Field Descriptions

Bits	Name	Description
0–4	—	Reserved. Defined as HID0[0]: EMCP, HID0[2]: EBA, HID0[3]: EBD, HID0[4]: BCLK on some earlier processors. Read as 0b1000_0.
5	TBEN [1]	Time base enable. Note that this bit must be set and the TBEN signal must be asserted to enable the time base and decrementer.
6	—	Reserved. Defined as ECLK on some earlier processors.
7	STEN[2]	Software table search enable. When a TLB miss occurs, the MPC7450 takes one of three TLB miss exceptions so that software can search the page tables for the desired PTE. See Section 4.6.15, "TLB Miss Exceptions," for details on the MPC7450 facilities for software table searching. 0 Hardware table search enabled 1 Software tables search enabled
8	—	Reserved for the MPC7441 and the MPC7451. Defined as DOZE on some earlier processors. The MPC7451 does not require a HID0 bit for DOZE mode, but rather is supported through a $\overline{\text{QREQ}}$/$\overline{\text{QACK}}$ processor-system handshake protocol. Refer to Section 10.2, "Programmable Power Mode," for further details.
	HIGH_BAT_EN[3]	Additional BATs enabled for the MPC7445, MPC7447, MPC7448, MPC7455, and the MPC7457. 0 Additional 4 IBATs (4–7) and 4 DBATs (4–7) disabled 1 Additional 4 IBATs (4–7) and 4 DBATs (4–7) enabled The additional BATs provide for more mapping of memory with the block address translation method.
9	NAP [1]	Nap mode enable. Operates in conjunction with MSR[POW]. 0 Nap mode disabled. 1 Nap mode enabled. Nap mode is invoked by setting MSR[POW] while this bit is set. In nap mode, the PLL and the time base remain active. Note that if both NAP and SLEEP are set, the MPC7450 ignores the SLEEP bit.
10	SLEEP [1]	Sleep mode enable. Operates in conjunction with MSR[POW]. 0 Sleep mode disabled. 1 Sleep mode enabled. Sleep mode is invoked by setting MSR[POW] while this bit is set. $\overline{\text{QREQ}}$ is asserted to indicate that the processor is ready to enter sleep mode. If the system logic determines that the processor can enter sleep mode, the quiesce acknowledge signal, $\overline{\text{QACK}}$, is asserted back to the processor. When the $\overline{\text{QACK}}$ signal assertion is detected, the processor enters sleep mode after several processor clocks. At this point, the system logic can turn off the PLL by first configuring PLL_CFG[0:4] to PLL bypass mode, and then disabling SYSCLK.
11	DPM [1]	Dynamic power management enable 0 Dynamic power management is disabled. 1 Functional units enter a low-power mode automatically if the unit is idle. This does not affect operational performance and is transparent to software or any external hardware.
12	—	Reserved. For test use; software should not set this bit.
13	BHTCLR [4]	Clear branch history table 0 The MPC7450 clears this bit one cycle after it is set. 1 Setting BHTCLR bit initializes all entries in BHT to weakly, not taken whether or not the BHT is enabled by HID0[BHT]. However, for correct results, the BHT should be disabled (HID0[BHT] = 0) before setting BHTCLR. Setting BHTCLR causes the branch unit to be busy for 64 cycles while the initialization process is completed.

Table 2-7. HID0 Field Descriptions (continued)

Bits	Name	Description
14	XAEN [5]	Extended addressing enabled 0 Extended addressing is disabled; the 4 most significant bits of the 36-bit physical address are cleared and a 32-bit physical address is used. 1 Extended addressing is enabled;, the 32-bit effective address is translated to a 36-bit physical address. If HID0[XAEN] is changed (cleared or set), the BATs and TLBs must be invalidated first.
15	NHR [1]	Not hard reset (software-use only). Helps software distinguish a hard reset from a soft reset. 0 A hard reset occurred if software had previously set this bit. 1 A hard reset has not occurred. If software sets this bit after a hard reset, when a reset occurs and this bit remains set, software knows it was a soft reset. The MPC7450 never writes this bit unless executing an **mtspr**(HID0).
16	ICE [6]	Instruction cache enable 0 The instruction cache is neither accessed nor updated. All pages are accessed as if they were marked cache-inhibited (WIM = x1x). Potential cache accesses from the bus (snoop and cache operations) are ignored. In the disabled state for the L1 caches, the cache tag state bits are ignored and all accesses are propagated to the bus as burst transactions. For those transactions, $\overline{\text{CI}}$ is asserted regardless of address translation. ICE is zero at power-up. 1 The instruction cache is enabled. Note that HID0[ICFI] must be set at the same time that this bit is set.
17	DCE [2]	Data cache enable 0 The data cache is neither accessed nor updated. All pages are accessed as if they were marked cache-inhibited (WIM = x1x). Potential cache accesses from the bus (snoop and cache operations) are ignored. In the disabled state for the L1 caches, the cache tag state bits are ignored and all accesses are propagated to the L2 cache, L3 cache, or bus as cache-inhibited. For those transactions, $\overline{\text{CI}}$ is asserted regardless of address translation.DCE is zero at power-up. 1 The data cache is enabled.Note that HID0[DCFI] must be set at the same time that this bit is set.
18	ILOCK [7]	Instruction cache lock 0 Normal operation 1 All of the ways of the instruction cache are locked. A locked cache supplies data normally on a read hit. On a miss, the access is treated the same as if the instruction cache was disabled.Thus, the bus request is a 32-byte burst read, but the cache is not loaded with data. The data is reloaded into the L2 and L3, unless the L2CR[L2DO] and L3CR[L3DO] bits are set, respectively. Note that setting this bit has the same effect as setting ICTRL[ICWL] to all ones. However, when this bit is set, ICTRL[ICWL] is ignored. Chapter 3, "L1, L2, and L3 Cache Operation," gives further details.
19	DLOCK [2]	Data cache lock 0 Normal operation 1 All the ways of the data cache are locked. A locked cache supplies data normally on a read hit but is treated as a cache-inhibited transaction on a miss. On a miss, a load transaction still reads a full cache line from the L2, L3, or bus but does not reload that line into the L1. Any store miss is treated like a write-through store and the transaction occurs on the bus with the $\overline{\text{WT}}$ signal asserted. A snoop hit to a locked L1 data cache operates as if the cache were not locked. A cache block invalidated by a snoop remains invalid until the cache is unlocked. Note that setting this bit has the same effect as setting LDSTCR[DCWL] to all ones. However, when this bit is set, LDSTCR[DCWL] is ignored. Refer to Chapter 3, "L1, L2, and L3 Cache Operation," for further details. To prevent locking during a cache access, a **sync** instruction must precede the setting of DLOCK and a **sync** must follow.

Table 2-7. HID0 Field Descriptions (continued)

Bits	Name	Description
20	ICFI [6]	Instruction cache flash invalidate 0 The instruction cache is not invalidated. The bit is cleared when the invalidation operation begins (the next cycle after the write operation to the register). The instruction cache must be enabled for the invalidation to occur. 1 An invalidate operation is issued that marks the state of each instruction cache block as invalid. Cache access is blocked during this time. Setting ICFI clears all the valid bits of the blocks and sets the PLRU bits to point to way L0 of each set. When the L1 flash invalidate bits are set through an **mtspr** operation, the hardware automatically clears these bits in the next cycle (provided that the corresponding cache enable bits are set in HID0). Note, in the MPC603 and MPC603e processors, the proper use of the ICFI and DCFI bits was to set and clear them in two consecutive **mtspr** operations. Software that already has this sequence of operations does not need to be changed to run on the MPC7450.
21	DCFI [2]	Data cache flash invalidate 0 The data cache is not invalidated. The bit is cleared when the invalidation operation begins (the next cycle after the write operation to the register). 1 An invalidate operation is issued that marks the state of each data cache block as invalid without writing back modified cache blocks to memory. Cache access is blocked during this time. Bus accesses to the cache are signaled as a miss during invalidate-all operations. Setting DCFI clears all the valid bits of the blocks and the PLRU bits to point to way L0 of each set. When the L1 flash invalidate bits are set through an **mtspr** operation, the hardware automatically clears these bits in the next cycle. Note that setting DCFI invalidates the data cache regardless of whether it is enabled. Note, in the MPC603e processors, the proper use of the ICFI and DCFI bits was to set them and clear them in two consecutive **mtspr** operations. Software that already has this sequence of operations does not need to be changed to run on the MPC7450.
22	SPD [1]	Speculative data cache and instruction cache access disable 0 Speculative bus accesses to nonguarded space (G = 0) from both the instruction and data caches is enabled. 1 Speculative bus accesses to nonguarded space in both caches is disabled. Thus, setting this bit prevents L1 data cache misses from going to the memory subsystem until the instruction that caused the miss is next to complete. The HID0[SPD] bit also prevents instruction cache misses from going to the memory subsystem until there are no unresolved branches. For more information on this bit and its effect on re-ordering of loads and stores, see Section 3.3.3.5, “Enforcing Store Ordering with Respect to Loads.”
23	—[8]	Reserved on the MPC7455. Defined as IFTT or IFEM on some earlier processors.
	XBSEN[9]	Extended BAT block size enable. (MPC7455 0 Disables IBAT*n*U[XBL] & DBAT*n*U[XBL] bits and clears these bits to zero. 1 Enables IBAT*n*U[XBL] & DBAT*n*U[XBL] bits BATnU[1518] become the 4 MSBs of the extended 15 bit BL field (BATnU[15–29]). This allows for extended BAT block sizes of 512MB, 1 GB, 2GB, and 4 GB. If HID0[XBBSEN] is set at startup and then cleared after startup, the XBL bits will not clear but stay the same as they were set at startup. HID0[XBSEN] should be set once at startup and once set should not be cleared. WhenHID0[XBSEN] is set at startup, and then HID0[XBSEN] is cleared, the IBAT*n*U[XBL] & DBAT*n*U[XBL] bits are not cleared but stay the same as what was set at startup. If backwards compatibility with previous processors is a concern, then HID0[XBSEN] should stay cleared so that the XBL bits are treated as 0’s. This allows the BAT translation to have a maximum block length of 256MB.

Table 2-7. HID0 Field Descriptions (continued)

Bits	Name	Description
24	SGE [10]	Store gathering enable 0 Store gathering is disabled. 1 Integer store gathering is performed as described in 3.1.2.3, "Store Gathering/Merging," and Section 6.4.4.2, "Store Gathering."
25	—	Reserved. Defined as DCFA on some earlier processors.
26	BTIC [1]	Branch target instruction cache enable. Used to enable use of the 128-entry branch instruction cache. 0 The BTIC contents are invalidated and the BTIC behaves as if it were empty. New entries cannot be added until the BTIC is enabled. 1 The BTIC is enabled and new entries can be added. The BTIC is flushed by context synchronization, which is required after a move to HID0. Thus if the synchronization rules are followed, modifying this BTIC bit implicitly flushes the BTIC. See Chapter 6, "Instruction Timing," for further details.
27	LRSTK [1]	Link register stack enable 0 Link register prediction is disabled. 1 Allows **bclr** and **bclrl** instructions to predict the branch target address using the link register stack which can accelerate returns from subroutines. See Chapter 6, "Instruction Timing," for further details.
28	FOLD [1]	Branch folding enable 0 Branch folding is disabled. All branches are dispatched to the completion buffer. 1 Branch folding is enabled, allowing branches to be folded out of the instruction prefetch stream before dispatch. The MPC7450 attempts to fold branches that do not modify the link and or count register. Note that if a branch is one of the three instruction buffers that are candidates for dispatch the cycle after it is processed, it cannot be folded it was not taken. See Chapter 6, "Instruction Timing," for further details.
29	BHT [1]	Branch history table enable 0 BHT disabled. The MPC7450 uses static branch prediction as defined by the PowerPC architecture (UISA) for those branch instructions the BHT would have otherwise used to predict (that is, those that use the CR or CTR mechanism to determine direction). For more information on static branch prediction, see "Conditional Branch Control," in Chapter 4 of the *Programming Environments Manual*. 1 Allows the use of the dynamic prediction 2048-entry branch history table (BHT). The BHT is disabled at power-on reset. All entries are set to weakly, not-taken.
30	NOPDST [2]	No-op **dst**, **dstt**, **dstst**, and **dststt** instructions 0 The **dst, dstt, dstst,** and **dststt** instructions are enabled. 1 The **dst, dstt, dstst,** and **dststt** instructions are no-oped globally, and all previously executed **dst** streams are cancelled.
31	NOPTI [10]	No-op the data cache touch instructions 0 The **dcbt** and **dcbtst** instructions are enabled. 1 The **dcbt** and **dcbtst** instructions are no-oped globally.

[1] A context synchronizing instruction must follow the mtspr.

[2] A dssall and sync must precede a mtspr and then a sync and context synchronizing instruction must follow. Note that if a user is not using the AltiVec data streaming instructions, then a dssall is not necessary prior to accessing the HID0{DCE] or HID0[DCFI] bit.

[3] MPC7445- and MPC7455-specific bit.

[4] A context synchronizing instruction must precede a mtspr and a branch instruction should follow. The branch instruction may be either conditional or unconditional. It ensures that all subsequent branch instructions see the newly initialized BHT values. For correct results, the BHT should be disabled (HID0[BHT] = 0) before setting BHTCLR.

[5] A dssall and sync must precede a mtspr and then a sync and a context-synchronizing instruction must follow. Alteration of HID0[XAEN] must be done with caches and translation disabled. The caches and TLBs must be flushed before they are re-enabled after the XAEN bit is altered. Note that if a user is not using the AltiVec data streaming instructions, then a dssall is not necessary prior to accessing the HID0[XAEN] bit.

[6] A context synchronizing instruction must immediately follow a mtspr. A mtspr instruction for HID0 should not modify either of these bits at the same time it modifies another bit that requires additional synchronization.

[7] A context synchronizing instruction must precede and follow a mtspr.

[8] MPC7441-, MPC7445-, MPC7450-, MPC7451-specific bit.

[9] MPC7455-, MPC7447-, MPC7457-, MPC7447A-, MPC7448- specific bit.

[10] A mtspr must follow a sync and a context synchronizing instruction.

HID0 can be accessed with **mtspr** and **mfspr** using SPR 1008. All **mtspr** instructions should be followed by a context synchronization instruction such as **isync**, for specific details see Section 2.4.2.4, “Synchronization.”

2.2.5.2 Hardware Implementation-Dependent Register 1 (HID1)

The hardware implementation-dependent register 1 (HID1) reflects the state of the PLL_CFG[0:4] signals and controls other functions. For the MPC7451, MPC7455, MPC7457, and MPC7447, PC bits 0–4 reflect the values on the PLL_CFG[0:4] pins. In the MPC7447A and MPC7448, the values of these bits are updated by the processor when DFS is enabled. See the *MPC7447A RISC Microprocessor Hardware Specifications* and the *MPC7448 RISC Microprocessor Hardware Specifications* for details on the HID1 configuration for these processors. The HID1 bits of the MPC7450 are shown in Figure 2-10.

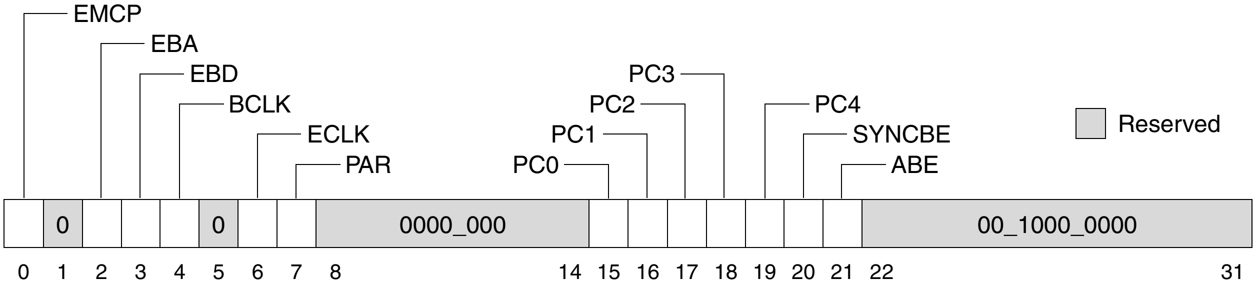


Figure 2-10. Hardware Implementation-Dependent Register 1 (HID1) for the MPC7450

The HID1 bits of the MPC7447A are shown in Figure 2-11.

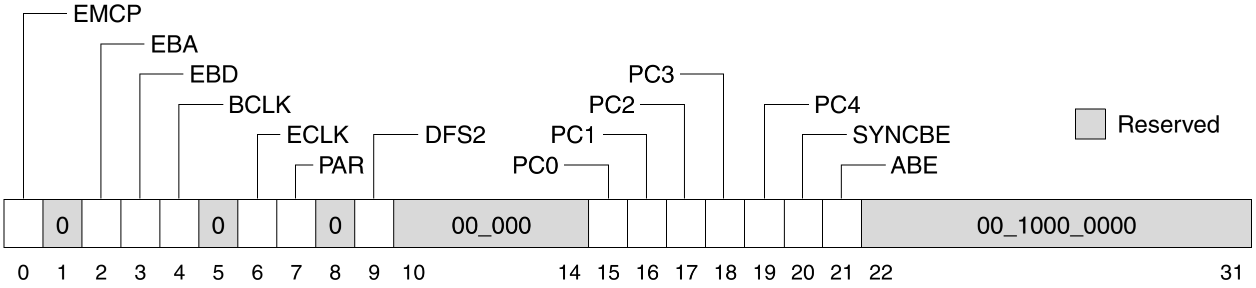
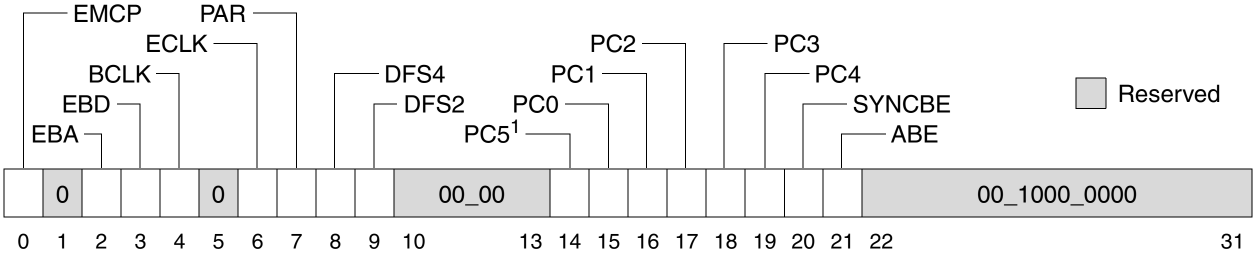


Figure 2-11. Hardware Implementation-Dependent Register 1 (HID1) for the MPC7447A

The HID1 bits of the MPC7448 are shown in Figure 2-12. See the *MPC7448 RISC Microprocessor Hardware Specifications* for details on HID1 configuration in the MPC7448.



[1] MPC7448-specific bit.

Figure 2-12. MPC7448 Hardware Implementation-Dependent Register 1 (HID1) for the MPC7448

The HID1 bits are described in Table 2-8.

Table 2-8. HID1 Field Descriptions

Bits[1]	Name	Description
0	EMCP	Machine check signal enable 0 Machine check is disabled. 1 Machine check input signal ($\overline{\text{MCP}}$) is enabled to cause machine check errors or checkstops.
1	—	Reserved
2	EBA	Enable/disable 60x/MPX bus address bus parity checking. 0 Address bus parity checking is disabled. 1 Allows an address bus parity error to cause a checkstop if MSR[ME] = 0 or a machine check exception if MSR[ME] = 1. Clearing EBA and EBD allows the processor to operate with memory subsystems that do not generate parity. The MPC7450 always generates parity regardless of whether checking is enable or disabled.
3	EBD	Enable/disable MPX/60x bus data parity checking. 0 Data parity checking is disabled. 1 Allows a data bus parity error to cause a checkstop if MSR[ME] = 0 or a machine check exception if MSR[ME] = 1. Clearing EBA and EBD allows the processor to operate with memory subsystems that do not generate parity.The MPC7450 always generates parity regardless of whether checking is enable or disabled.
4	BCLK	CLK_OUT output enable and clock type selection. Used in conjunction with HID1[ECLK] and the $\overline{\text{HRESET}}$ signal to configure CLK_OUT. See Table 2-9.
5	—	Reserved
6	ECLK	CLK_OUT output enable and clock type selection. Used in conjunction with HID1[BCLK] and the $\overline{\text{HRESET}}$ signal to configure CLK_OUT. See Table 2-9.
7	PAR	Disable precharge for $\overline{\text{ARTRY}}$, $\overline{\text{SHD0}}$, and $\overline{\text{SHD1}}$ pins. 0 $\overline{\text{ARTRY}}$, $\overline{\text{SHD0}}$, and $\overline{\text{SHD1}}$ signals are driven high when negated. 1 $\overline{\text{ARTRY}}$, $\overline{\text{SHD0}}$, and $\overline{\text{SHD1}}$ signals are not driven high when negated. Thus, the system must restore these signals to the high state on negation.

Table 2-8. HID1 Field Descriptions (continued)

Bits[1]	Name	Description
8	—	Reserved.
	DFS4 [3]	Dynamic frequency switching (DFS) divide-by-four mode. 0 DFS divide-by-four mode is disabled. 1 DFS divide-by-four mode is enabled. When both DFS2 and DFS4 bits are set, divide-by-four mode is selected.
9	DFS2 [2, 3]	Dynamic frequency switching (DFS) divide-by-two mode. 0 DFS divide-by-two mode is disabled. 1 DFS divide-by-two mode is enabled. Note that the divisors are only applicable to the processor-to-system ratio chosen at reset by the external PLL_CFG pins. If the HID1 settings select a ratio that is not supported (see the *MPC7447A RISC Microprocessor Hardware Specifications* or *MPC7448 RISC Microprocessor Hardware Specifications* for supported ratios), the setting of the HID1[DFS2] bit is ignored.
10–13	—	Reserved
14	—	Reserved
	PC5 [3]	PLL configuration bit 5 (read-only). Reflects the state of the PLL multiplier.
15	PC0	PLL configuration bit 0 (read-only). Reflects the state of the PLL multiplier.
16	PC1	PLL configuration bit 1 (read-only). Reflects the state of the PLL multiplier.
17	PC2	PLL configuration bit 2 (read-only). Reflects the state of the PLL multiplier.
18	PC3	PLL configuration bit 3 (read-only). Reflects the state of the PLL multiplier.
19	PC4	PLL configuration bit 4 (read-only). Reflects the state of the PLL multiplier.
20	SYNCBE	Address broadcast enable for **sync**, **eieio** 0 Address broadcasting of **sync**, and **eieio** is disabled. 1 Address broadcasting of **sync**, and **eieio** is enabled. Note this bit must be set in MP systems and systems that reorder stores.
21	ABE	Address broadcast enable for **dcbf**, **dcbst**, **dcbi**, **icbi**, **tlbie**, and **tlbsync**. 0 Address broadcasting of **dcbf**, **dcbst**, **dcbi**, **icbi**, **tlbie**, and **tlbsync** is disabled. Note that when HID1[ABE] is cleared this does not exclude all cache operations from the bus, just **icbi**, **tlbie**, and **tlbsync**. 1 Address broadcasting for cache control operations (**dcbf**, **dcbst**, **dcbi**, **icbi**) and TLB control operations (**tlbie** and **tlbsync**) is enabled. Note that whether the broadcast occurs depends on the setting of the M bit of WIMG and whether the access causes a hit to modified memory. See Section 3.8.2, “Bus Operations Caused by Cache Control Instructions,” for more information on broadcast operations. The ABE bit must be set for MP systems.
22–31	—	Reserved. Read as 0b00_1000_0000.

[1] A sync and context synchronizing instruction must follow a mtspr.

[2] MPC7447A-specific bit, reserved on MPC7450.

[3] MPC7448-specific bit.

NOTE

The required software sequence for setting or clearing the HID1[DFS2] bit is as follows:

```
sync
mtspr HID1
sync
isync
```

Table 2-9 shows how HID1[BCLK], HID1[ECLK], and $\overline{\text{HRESET}}$ are used to configure CLK_OUT. See Section 8.4.6.3, “JTAG Test Data Output (TDO)—Output,” for more information.

Table 2-9. HID1[BCLK] and HID1[ECLK] CLK_OUT Configuration

$\overline{\text{HRESET}}$	HID1[ECLK]	HID1[BCLK]	CLK_OUT
Asserted	x	x	High impedance
Negated	0	0	Zero
Negated	0	1	Bus/2
Negated	1	0	Core
Negated	1	1	Core/2

HID1 can be accessed with **mtspr** and **mfspr** using SPR 1009. All **mtspr** instructions should be followed by a **sync** and context synchronization instruction for specific details see Section 2.4.2.4, “Synchronization.”

2.2.5.2.1 MPC7447A-Specific HID1 PLL Configuration Field

The PLL configuration field (HID1[15–19] bits) will dynamically update upon the selection of a DFS divisor mode to reflect the new ratio. The ratios:

- 2.5:1,
- 3.5:1, and
- 4.5:1

that are not selectable on the MPC7447A at hard reset through the PLL_CFG pins, share PLL configuration field encodings with

- 8.5:1
- 13.5:1 and
- 9.5:1, respectively

These settings can be correctly decoded by including the HID1[DFS2] bit in the decode. See the *MPC7447A RISC Microprocessor Hardware Specifications* for details on decoding the HID1[15–19] and HID1 DFS settings. The DFS2 bit was originally named DFS1. For details on decoding the HID1[14–19] and HID1 DFS settings in the MPC7448, see the *MPC7448 RISC Microprocessor Hardware Specifications*.

2.2.5.3 Memory Subsystem Control Register (MSSCR0)

The memory subsystem control register (MSSCR0), shown in Figure 2-13, is used to configure and operate the memory subsystem for the MPC7450. It is accessed as SPR 1014. The MSSCR0 is initialized to all zeros except for the read-only bits.

Because MSSCR0 alters how the MPC7450 responds to snoop requests, it is important that changes to the values of the fields in MSSCR0 are handled correctly.

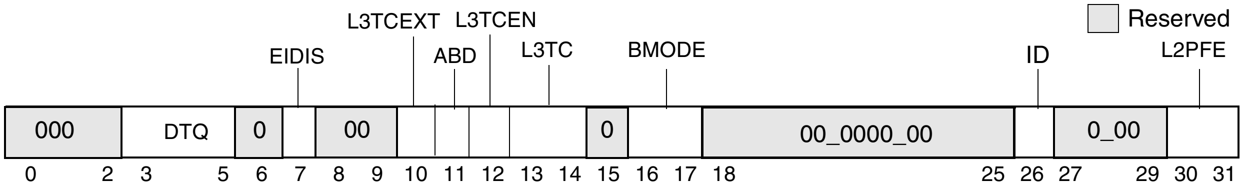


Figure 2-13. Memory Subsystem Control Register (MSSCR0)

Table 2-10 describes MSSCR0 fields.

Table 2-10. MSSCR0 Field Descriptions

Bits	Name	Function
0–2	—	Reserved
3–5	DTQ	DTQ size. Determines the maximum number of outstanding data bus transactions that the MPC7450 can support. See Chapter 9, “System Interface Operation,” for more information.The DTQ bit values are as follows: 000 8 entries 001 16 entries 010 2 entries 011 3 entries 100 4 entries 101 5 entries 110 6 entries 111 7 entries
6	—	Reserved
7	EIDIS	Disable external intervention in MPX bus mode 0 External interventions occur. 1 The MPC7450 performs external pushes instead of external interventions. External interventions are disabled.
8–9	—	Reserved
10	L3TCEXT	L3 turn around clockcount extension (MPC7457-Specific) 0 Used with MSSCR0[L3TC] to determine the L3 turnaround clock count. See L3CR[L3TC] field description. 1 Used with MSSCR0[L3TC] to determine the L3 turnaround clock count. See MSSCR0[L3TC] field description. Note, that the MSSCR0[10] bit is reserved on the MPC7450 and is used as an L3 turnaround clock count only on the MPC7457.

Table 2-10. MSSCR0 Field Descriptions (continued)

Bits	Name	Function
11	ABD	Address bus driven mode 0 Address bus driven mode disabled 1 Address bus driven mode enabled The read-only bit reflects the state of the $\overline{\text{BMODE0}}$ signal after $\overline{\text{HRSET}}$ negation and indicates whether the processor is address bus driven mode. See Section 9.3.2.1, “Address Bus Driven Mode,” for more information.
12	L3TCEN	L3 turnaround clock enable. 0 L3 turnaround clock disabled. 1 L3 turnaround clock is enabled. See Chapter 3, “L1, L2, and L3 Cache Operation,” for more information.
13–14	L3TC	L3 turnaround clock count. Both read-to-write and write-to-read turn around is affected. The following bit values determine the number of cycles the L3 waits between read and write transactions if L3TCEN is set.The following values are correct for the MPC7450. Note that only for the MPC7457, the following values are correct when MSSCR0[L3TCEXT] = 0: 00 2 L3CK*n* cycles 01 3 L3CK*n* cycles 10 4 L3CK*n* cycles 11 5 L3CK*n* cycles Also note that only for the MPC7457, the following values are correct when MSSCR0[L3TCEXT] = 1. These values are not used on the MPC7450. 00 6 L3CK*n* cycles 01 7 L3CK*n* cycles 10 8 L3CK*n* cycles 11 9 L3CK*n* cycles
15	—	Reserved.
16–17	BMODE	Bus mode (read-only). Reflects the inverse of the voltage levels on $\overline{\text{BMODE}}$[0:1] while $\overline{\text{HRESET}}$ is asserted. Indicates whether the system interface uses the 60x or MPX bus protocol as described in Chapter 9, “System Interface Operation.” 00 60x bus mode 01 Reserved 10 MPX bus mode 11 Reserved Note that the value on $\overline{\text{BMODE}}$[0:1] after reset negates determines other values of MSSCR0 as follows: $\overline{\text{BMODE0}}$ (post reset) → MSSCR0[ABD] $\overline{\text{BMODE1}}$ (post reset) → MSSCR0[ID]
18–25	—	Reserved. Normally cleared, used in debug, writing nonzero values may cause boundedly undefined results.
26	ID	Processor identification. Sets the processor ID to either processor 0 or 1. Determined by the inverse of the voltage levels on $\overline{\text{BMODE1}}$ while $\overline{\text{HRESET}}$ is negated. 0 $\overline{\text{BMODE1}}$ negated after $\overline{\text{HRESET}}$ negated 1 $\overline{\text{BMODE1}}$ asserted after $\overline{\text{HRESET}}$ negated In a multiprocessor system, one processor can be assigned by the $\overline{\text{BMODE1}}$ as processor 0 and all other processor can be assigned as processor 1. Then software can find processor 0 and use it to re-identify the other processors by writing unique values to the PIR of the other CPUs.

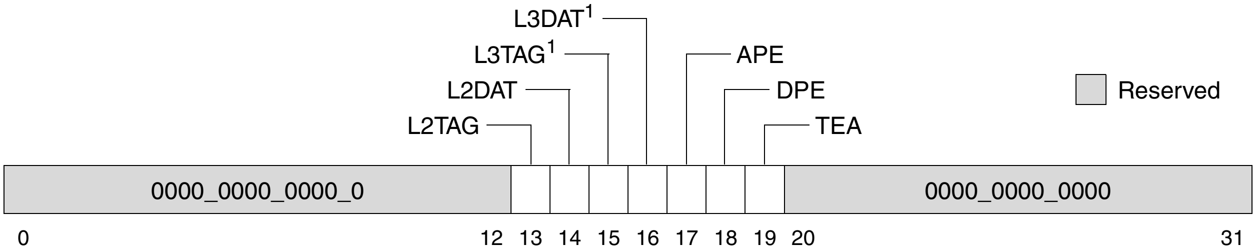
Table 2-10. MSSCR0 Field Descriptions (continued)

Bits	Name	Function
27–29	—	Reserved. Read as zeros.
30–31	L2PFE	L2 prefetching enabled. The following values determine the number of L2 prefetch engines enabled as follows: 00 L2 prefetching disabled, no prefetch engines 01 One prefetch engine enabled 10 Two prefetch engines enabled 11 Three prefetch engines enabled These bits enable alternate sector prefetching in the 2-sectored L2 cache; up to 3 outstanding prefetch engines may be active.

2.2.5.4 Memory Subsystem Status Register (MSSSR0)

The memory subsystem status register (MSSSR0), shown in Figure 2-14, is used to report parity in the L2 and L3 caches of the MPC7450. It is accessed as SPR 1015. The MSSSR0 is initialized to all 0s except for the read-only bits.

In the MPC7448, which has no L3 support, the MSS status register reports MSS enabled error status. Note that tag and data parity and data ECC errors are reported in the error detect (L2ERRDET) register whether or not error reporting is enabled. The corresponding bit in MSSSR0 is set only if error reporting is enabled.



[1] The L3 cache and the L3 cache interface are not supported in the MPC7441, MPC7445, MPC7447, MPC7447A, or MPC7448.

Figure 2-14. MSS Status Register (MSSSR0) for the MPC7450

The bits of MSSSR0 in the MPC7448 are shown in Figure 2-15.

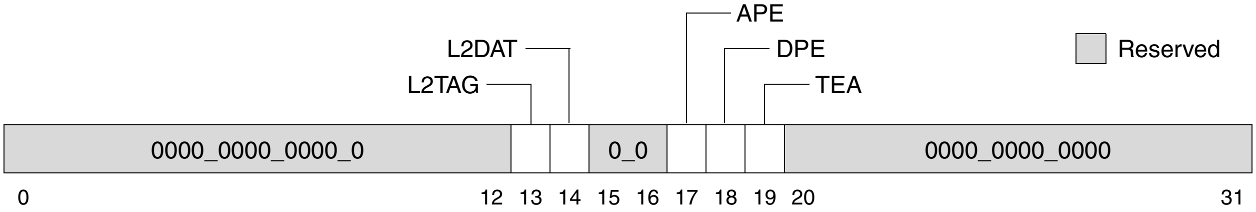


Figure 2-15. MSS Status Register (MSSSR0) for the MPC7448

Table 2-11 describes MSSSR0 fields.

Table 2-11. MSSSR0 Field Descriptions

Bits	Name	Description
0–12	—	Reserved. Normally cleared, used in debug, writing nonzero values may cause boundedly undefined results.
13	L2TAG	L2 tag parity error 0 L2 tag parity error not detected. 1 L2 tag parity error detected.
14	L2DAT	L2 data parity error 0 L2 data parity error not detected. 1 L2 data parity error detected.
15	—	Reserved in the MPC7448. Normally cleared, used in debug, writing nonzero values may cause boundedly undefined results.
	L3TAG	L3 tag parity error 0 L3 tag parity error not detected. 1 L3 tag parity error detected.
16	—	Reserved in the MPC7448. Normally cleared, used in debug, writing nonzero values may cause boundedly undefined results.
	L3DAT	L3 data parity error 0 L3 data parity error not detected. 1 L3 data parity error detected.
17	APE	Address bus parity error 0 Address bus parity error not detected. 1 Address bus parity error detected.
18	DPE	Data bus parity error 0 Data bus parity error not detected. 1 Data bus parity error detected.
19	TEA	Bus transfer error acknowledge 0 $\overline{\text{TEA}}$ not detected as asserted. 1 $\overline{\text{TEA}}$ detected as asserted.
20–31	—	Reserved

2.2.5.5 Instruction and Data Cache Registers

Several registers are used for configuring and controlling the various L1, L2, and L3 caches. Along with the cache registers (L2CR, L3CR, ICTRL, LDSTCR, and L3PM), HID0 is used in configuring the caches. Details of how the various cache registers are used is discussed below. See the Chapter 3, “L1, L2, and L3 Cache Operation,” for further details on configuring the cache.

2.2.5.5.1 L2 Cache Control Register (L2CR)

The L2 cache control register (L2CR), shown in Figure 2-16, is a supervisor-level, implementation-specific SPR used to configure and operate the L2 cache of the MPC7450. It is cleared by a hard reset or power-on reset.

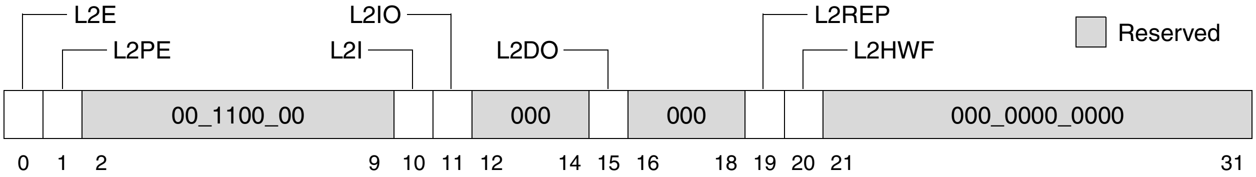


Figure 2-16. L2 Control Register (L2CR) for the MPC7450

For the MPC7448, the L2CR is shown in Figure 2-17. The functionality of the L2PE bit has changed significantly with the addition of ECC support. Tag parity in the MPC7448 is controlled separately through the TPARDIS bit in the L2ERRDIS register. Data parity can only be enabled through L2CR[L2PE] if ECC is disabled in the L2ERRDIS register.

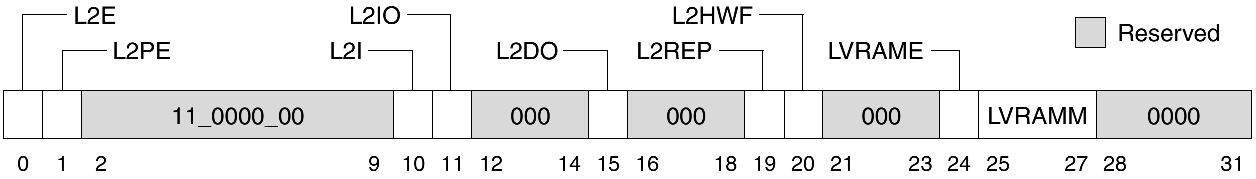


Figure 2-17. L2 Control Register (L2CR) for the MPC7448

The L2 cache interface is described in Chapter 3, "L1, L2, and L3 Cache Operation." The bits of the L2CR are described in Table 2-12.

Table 2-12. L2CR Field Descriptions

Bits	Name	Description
0	L2E	L2 enable. Used to enable the L2 cache. 0 The L2 cache is disabled and is not accessed for reads, snoops, or writes. 1 The L2 cache is enabled.
1	L2PE	L2 tag and data parity checking enable 0 L2 parity checking disabled. 1 L2 parity checking enabled. Enables or disables the checking of L2 tag and data parity.
		L2 data parity checking enable for the MPC7448 0 L2 data parity checking disabled. 1 L2 data parity checking enabled if L2ERRDIS[MBECCDIS]=1 and L2ERRDIS[SBECCDIS]=1. If ECC is enabled (L2ERRDIS[MBECCDIS]=0 or L2ERRDIS[SBECCDIS]=0), setting L2PE has no effect; ECC checking will still be performed. Note: MPC7450–MPC7447A used this bit to enable/disable tag parity checking as well as data parity checking. MPC7448 has moved tag parity enable/disable to the new L2ERRDIS register. Data parity can only be enabled with L2CR[L2PE] if ECC is disabled in the L2ERRDIS register. By default, tag parity and data ECC checking are enabled on MPC7448.
2–3	—	Reserved Must be set by software during initialization to ob00. For the MPC7448, will read as 0b11.
4–9	—	Reserved

Table 2-12. L2CR Field Descriptions (continued)

Bits	Name	Description
10	L2I	L2 global invalidate 0 L2 cache not invalidated globally 1 L2 cache invalidated globally Invalidates the L2 cache globally by clearing the L2 status bits. This bit must not be set while the L2 cache is enabled. Note that L2I is automatically cleared when the global invalidate completes.
11	L2IO	L2 instruction-only. Causes the L2 cache to allocate lines for instruction cache transactions only. 0 The L2 cache allocates entries for data accesses that miss. 1 The L2 cache does not allocate entries for data accesses that miss in the L2. Data accesses that hit, instruction accesses, and system accesses are unaffected. If L2DO and L2IO are both set, no new lines are allocated in the L2 cache, effectively locking the entire cache.
12	—	Reserved on the MPC7450, MPC7451, MPC7441, MPC7445, MPC7457, MPC7447, MPC7447A, and MPC7448.
	L3OH0	L3 output hold 0 (MPC7455-specific bit). This bit, in conjunction with L3OH1 (see Table 2-26), configures output hold time for address, data, and control signals driven by the MPC7455 to the L3 data RAMs. They should generally be set according to the SRAM's input hold time requirements. See the *MPC7455 RISC Microprocessor Hardware Specifications* for specific output hold times.
13–14	—	Reserved
15	L2DO	L2 data only. L2 cache lines are allocated for data cache transactions only. 0 The L2 cache allocates entries for instruction accesses that miss. 1 The L2 cache does not allocate entries for instruction accesses that miss in the L2. Instruction accesses that hit in the L2, data accesses, and system accesses are unaffected. If both L2DO and L2IO are set, no new lines are allocated in the L2 cache, effectively locking the entire cache.
16–18	—	Reserved
19	L2REP	L2 replacement algorithm. 0 Pseudo-random replacement algorithm (default) 1 3-bit counter replacement algorithm See Section 3.6.4.4, "L2 Cache Line Replacement Algorithms," for more information.
20	L2HWF	L2 hardware flush 0 The L2 is not being flushed. 1 A 0->1 transition on this bit triggers a global flush of the entire L2 cache. All modified lines will be cast out to main memory. The cache must be locked by setting L2CR[L2DO]=1 and L2CR[L2IO]=1 before setting this bit. See Section 3.6.3.1.5, "Flushing of L1, L2, and L3 Caches," for more information.
21–31	—	Reserved
21–23	—	Reserved on the MPC7448
24	—	Reserved for the MPC7450, MPC7441, MPC7451, MPC7455, MPC7445, MPC7457, MPC7447, and MPC7447A.
	LVRAME	LVRAM enable (MPC7448-specific bit). See hardware spec for details. 0 Disable LVRAM mode 1 Enable LVRAM mode

Table 2-12. L2CR Field Descriptions (continued)

Bits	Name	Description
25–27	—	Reserved for the MPC7450, MPC7441, MPC7451, MPC7455, MPC7445, MPC7457, MPC7447, and MPC7447A.
	LVRAMM	LVRAM mode (read-only) (MPC7448-specific bit). See hardware spec for details. 000 Reserved if LVRAM mode is enabled 001 Mode1 010 Mode2 011 Mode3 100 Mode4 101 Mode5 110 Mode 6 111 Mode 7
28–31	—	Reserved on the MPC7448

The L2CR register can be accessed with the **mtspr** and **mfspr** instructions using SPR 1017.

2.2.5.5.2 L2 Error Injection Mask High Register (L2ERRINJHI)—MPC7448-Specific

The L2 error injection mask high register (L2ERRINJHI), shown in Figure 2-18, is a supervisor-level SPR in the MPC7448 used for error injection of the high word of the data path.

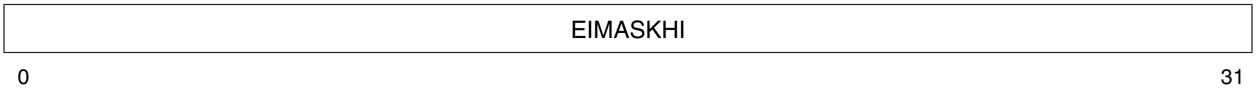


Figure 2-18. L2 Error Injection Mask High Register (L2ERRINJHI) for the MPC7448

Table 2-13 describes L2ERRINJHI[EIMASKHI].

Table 2-13. L2ERRINJHI Field Description for the MPC7448

Bits	Name	Description
0–31	EIMASKHI	Error injection mask for the high word of the data path. A set bit corresponding to a data path bit causes that bit on the data path to be inverted on cache writes if date array error injection is enabled by setting L2ERRINJCTL[DERRIEN] = 1.

2.2.5.5.3 L2 Error Injection Mask High Register (L2ERRINJLO)—MPC7448-Specific

The L2 error injection mask low register (L2ERRINJLO), shown in Figure 2-19, is a supervisor-level SPR in the MPC7448 used for error injection of the low word of the data path.

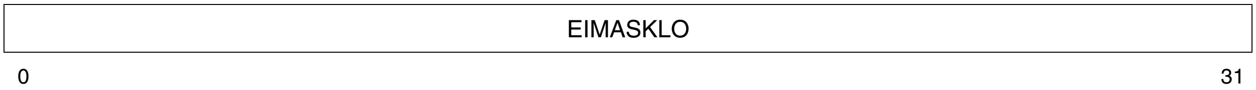


Figure 2-19. L2 Error Injection Mask Low Register (L2ERRINJLO) for the MPC7448

Table 2-14 describes L2ERRINJLO[EIMASKLO].

Table 2-14. L2ERRINJLO Field Description for the MPC7448

Bits	Name	Description
0–31	EIMASKLO	Error injection mask for the low word of the data path. A set bit corresponding to a data path bit causes that bit on the data path to be inverted on cache writes if data array error injection is enabled by setting L2ERRINJCTL[DERRIEN] = 1.

2.2.5.5.4 L2 Error Injection Mask Control Register (L2ERRINJCTL)—MPC7448-Specific

The L2 error injection mask control register (L2ERRINJCTL), shown in Figure 2-20, is a supervisor-level SPR used to configure error injection in the MPC7448.

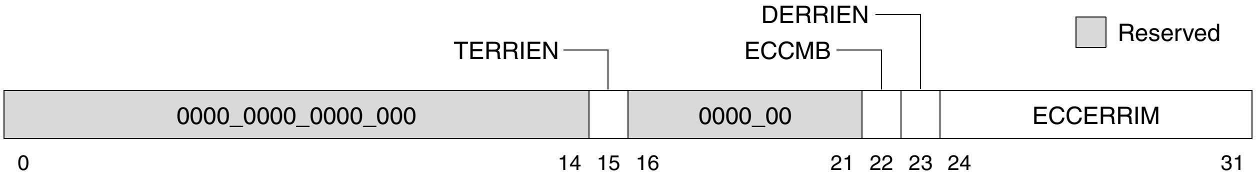


Figure 2-20. L2 Error Injection Mask Control Register (L2ERRINJCTL) for the MPC7448

Table 2-15 describes L2ERRINJCTL fields.

Table 2-15. L2ERRINJCTL Field Descriptions for the MPC7448

Bits	Name	Description
0–14	—	Reserved
15	TERRIEN	L2 tag array error injection enable 0 No tag errors are injected. 1 All subsequent entries written to the L2 tag array have the parity bit inverted.
16–21	—	Reserved
22	ECCMB	ECC mirror byte enable 0 ECC byte mirroring is disabled. 1 The most significant data path byte is mirrored onto the ECC byte if DERRIEN = 1.
23	DERRIEN	L2 data array error injection enable 0 No data errors are injected. 1 All subsequent entries written to the L2 data array have data or ECC bits inverted as specified in the data error injection masks and ECC error injection mask and/or data path byte mirrored onto ECC as specified by the ECC mirror byte enable bit, ECCMB. Note: if both ECC mirror byte and data error injection are enabled, ECC mask error injection is performed on the mirrored ECC.
24–31	ECCERRIM	Error injection mask for the ECC bits. A set bit corresponding to an ECC bit causes that bit to be inverted on cache writes if DERRIEN = 1.

2.2.5.5.5 L2 Error Capture Data High Register (L2CAPTDATAHI)—MPC7448-Specific

The L2 error capture data high register (L2CAPTDATAHI), shown in Figure 2-21, holds the high word of the L2 data that contains the detected error in the MPC7448.

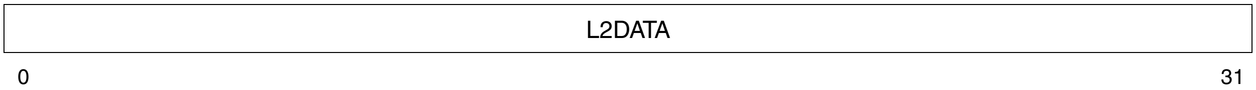


Figure 2-21. L2 Error Capture Data High Register (L2CAPTDATAHI) for the MPC7448

Table 2-16 describes L2CAPTDATAHI[L2DATA].

Table 2-16. L2CAPTDATAHI Field Description for the MPC7448

Bits	Name	Description
0–31	L2DATA	L2 data high word (read only)

2.2.5.5.6 L2 Error Capture Data Low Register (L2CAPTDATALO)—MPC7448-Specific

The L2 error capture data low register (L2CAPTDATALO), shown in Figure 2-22, holds the low word of the L2 data that contains the detected error in the MPC7448.

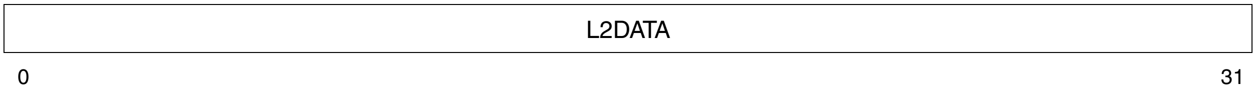


Figure 2-22. L2 Error Capture Data Low Register (L2CAPTDATALO) for the MPC7448

Table 2-17 describes L2CAPTDATALO[L2DATA].

Table 2-17. L2CAPTDATALO Field Description for the MPC7448

Bits	Name	Description
0–31	L2DATA	L2 data low word (read only)

2.2.5.5.7 L2 Error Syndrome Register (L2CAPTECC)—MPC7448-Specific

The L2 error syndrome register (L2CAPTECC), shown in Figure 2-23, is a supervisor-level SPR in the MPC7448 that contains the ECC syndrome and datapath ECC of the failing double word.

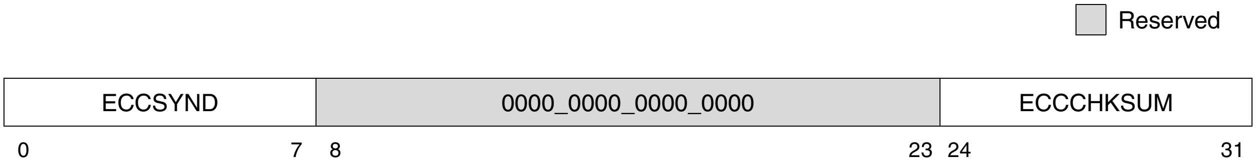


Figure 2-23. L2 Error Syndrome Register (L2CAPTECC) for the MPC7448

Table 2-18 describes L2CAPTECC fields.

Table 2-18. L2CAPTECC Field Descriptions for the MPC7448

Bits	Name	Description
0–7	ECCSYND	The calculated ECC syndrome of the failing double word (read only)
8–23	—	Reserved
24–31	ECCCHKSUM	The datapath ECC of the failing double word (read only)

2.2.5.5.8 L2 Error Detect Register (L2ERRDET)—MPC7448-Specific

The L2 error detect register (L2ERRDET), shown in Figure 2-24, is a supervisor-level SPR in the MPC7448 that shows the errors detected.

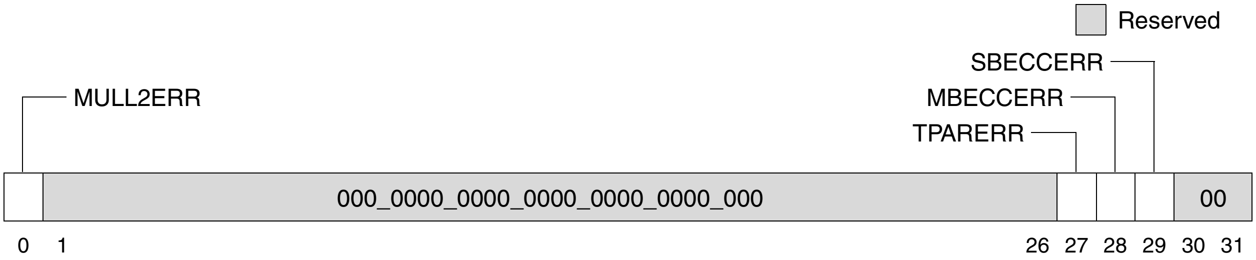


Figure 2-24. L2 Error Detect Register (L2ERRDET) for the MPC7448

Table 2-19 describes L2ERRDET fields.

Table 2-19. L2ERRDET Field Descriptions for the MPC7448

Bits	Name	Description
0	MULL2ERR	Multiple L2 errors (Bit reset, write-1-to-clear) 0 Multiple L2 errors of the same type were not detected 1 Multiple L2 errors of the same type were detected Note that setting this bit to 1 clears it to a value of 0.
1–26	—	Reserved
27	TPARERR	Tag parity error (Bit reset, write-1-to-clear) 0 Tag parity error was not detected 1 Tag parity error was detected Note that setting this bit to 1 clears it to a value of 0.
28	MBECCERR	Multiple-bit ECC error (Bit reset, write-1-to-clear) 0 Multiple-bit ECC errors were not detected 1 Multiple-bit ECC errors were detected Note that setting this bit to 1 clears it to a value of 0.
29	SBECCERR	Single-bit ECC error (Bit reset, write-1-to-clear) 0 Single-bit ECC error was not detected 1 Single-bit ECC error was detected Note that setting this bit to 1 clears it to a value of 0.
30–31	—	Reserved

2.2.5.5.9 L2 Error Disable Register (L2ERRDIS)—MPC7448-Specific

The L2 error disable register (L2ERRDIS), shown in Figure 2-25, is a supervisor-level SPR that disables and enables error detection in the MPC7448. Note that the L2 cache must be disabled and flushed before enabling or disabling ECC to ensure that no errors occur. See Section 3.6.3.4.1, “Enabling or Disabling ECC,” and Table 2-46 for the synchronization requirements required to enable or disable ECC.

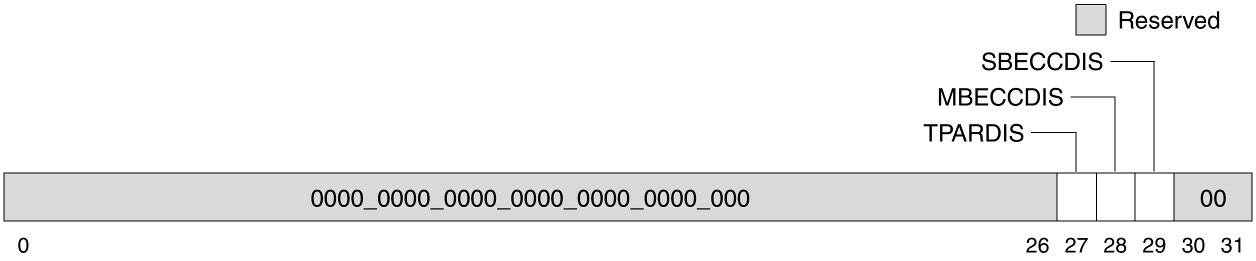


Figure 2-25. L2 Error Disable Register (L2ERRDIS) for the MPC7448

Table 2-20 describes L2ERRDIS fields.

Table 2-20. L2ERRDIS Field Descriptions for the MPC7448

Bits	Name	Description
0–26	—	Reserved
27	TPARDIS	Tag parity error disable 0 Tag parity error detection enabled 1 Tag parity error detection disabled
28	MBECCDIS	Multiple-bit ECC error disable 0 Multiple-bit ECC error detection enabled 1 Multiple-bit ECC error detection disabled
29	SBECCDIS	Single-bit ECC error disable 0 Single-bit ECC error detection enabled 1 Single-bit ECC error detection disabled
30–31	—	Reserved

2.2.5.5.10 L2 Error Interrupt Enable Register (L2ERRINTEN)—MPC7448-Specific

The L2 error interrupt enable register (L2ERRINTEN), shown in Figure 2-26, is a supervisor-level SPR used to enable L2 error interrupts in the MPC7448. When any of these error conditions exist and the corresponding bit in the L2ERRINTEN register is enabled, a machine check exception is generated.

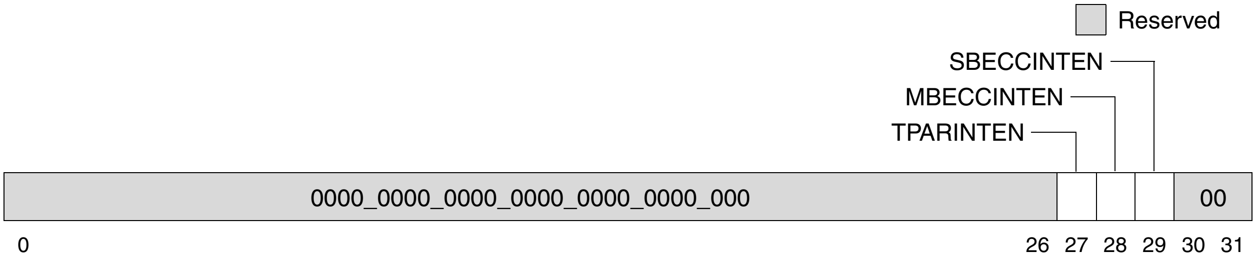


Figure 2-26. L2 Error Interrupt Enable Register (L2ERRINTEN) for the MPC7448

Table 2-21 describes L2ERRINTEN fields.

Table 2-21. L2ERRINTEN Field Descriptions for the MPC7448

Bits	Name	Description
0–26	—	Reserved
27	TPARINTEN	Tag parity error reporting enable 0 Tag parity error reporting disabled 1 Tag parity error reporting enabled.
28	MBECCINTEN	Multiple-bit ECC error reporting enable 0 Multiple-bit ECC error reporting disabled 1 Multiple-bit ECC error reporting enabled
29	SBECCINTEN	Single-bit ECC error reporting enable 0 Single-bit ECC error reporting disabled 1 Single-bit ECC error reporting enabled
30–31	—	Reserved

2.2.5.5.11 L2 Error Attributes Capture Register (L2ERRATTR)—MPC7448-Specific

The L2 error attributes capture register (L2ERRATTR), shown in Figure 2-27, is a supervisor-level SPR in the MPC7448 that describes the L2 error attributes. All the fields of the L2ERRATTR are read-only except for bit 31, VALINFO.

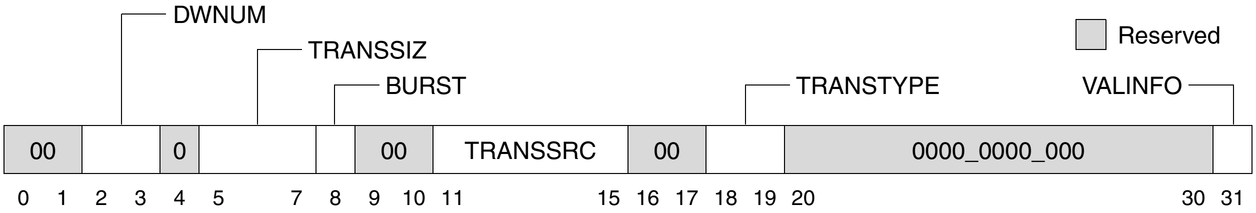


Figure 2-27. L2 Error Attributes Capture Register (L2ERRATTR) for the MPC7448

Table 2-22 describes L2ERRATTR fields.

Table 2-22. L2ERRATTR Field Descriptions for the MPC7448

Bits	Name	Description
0–1	—	Reserved (read only)
2–3	DWNUM	Double-word number of the detected error (read only, data ECC errors only)
4	—	Reserved
5–7	TRANSSIZ	Transaction size for detected error (read only) 000 8 bytes (single-beat) or reserved (burst) 001 1 byte (single-beat) or 16 bytes (burst) 010 2 bytes (single-beat) or 32 bytes (burst) 011 3 bytes (single beat) or reserved (burst) 100 4 bytes (single-beat) or reserved (burst) 101 5 bytes (single-beat) or reserved (burst) 110 6 bytes (single-beat) or reserved (burst) 111 7 bytes (single-beat) or reserved (burst)
8	BURST	Burst transaction for detected error (read only) 0 Single-beat (≤ 64 bits) transaction 1 Burst transaction
9–10	—	Reserved
11–15	TRANSSRC	Transaction source for detected error (read only) 00000External (system logic) 10000Processor (instruction) 10001Processor (data)
16–17	—	Reserved
18–19	TRANSTYPE	Transaction type for detected error (read only) 00 Snoop (tag/status read) 01 Write 10 Read 11 Reserved
20–30	—	Reserved
31	VALINFO	L2 capture registers valid 0 L2 capture registers contain no valid information or no enabled errors were detected. 1 L2 capture registers contain information of the first detected error that has reporting enabled. Software must clear this bit to unfreeze error capture so error detection hardware can overwrite the capture address/data/attributes for a newly detected error.

2.2.5.5.12 L2 Error Address Error Capture Register (L2ERRADDR)—MPC7448-Specific

The L2 error address error capture register (L2ERRADDR), shown in Figure 2-28, is a supervisor-level SPR in the MPC7448 that shows the L2 address corresponding to bits 4–35 of the detected error.

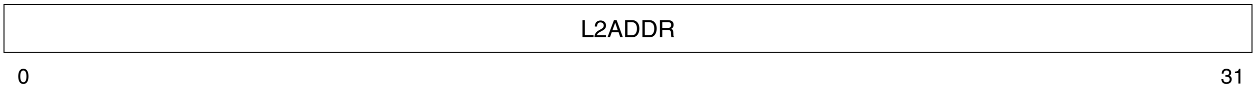


Figure 2-28. L2 Error Address Error Capture Register (L2ERRADDR) for the MPC7448

Table 2-23 describes L2ERRADDR.

Table 2-23. L2ERRADDR Field Description for the MPC7448

Bits	Name	Description
0–31	L2ADDR	L2 address[4:35] corresponding to detected error (read only)

2.2.5.5.13 L2 Error Address Error Capture Register (L2ERREADDR)—MPC7448-Specific

The L2 error address error capture register (L2ERREADDR), shown in Figure 2-29, is a supervisor-level SPR in the MPC7448 that shows the L2 address corresponding to bits 0–3 of the detected error.

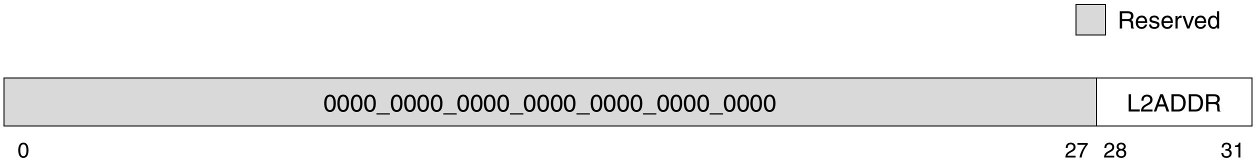


Figure 2-29. L2 Error Address Error Capture Register (L2ERREADDR) for the MPC7448

Table 2-24 describes L2ERREADDR.

Table 2-24. L2ERREADDR Field Description for the MPC7448

Bits	Name	Description
0–27	—	Reserved
28–31	L2EADDR	L2 address[0:3] corresponding to detected error (read only)

2.2.5.5.14 L2 Error Control Register (L2ERRCTL)—MPC7448-Specific

The L2 error control register (L2ERRCTL), shown in Figure 2-30, is a supervisor-level SPR in the MPC7448 that configures the L2 cache threshold and provides an L2 error count.

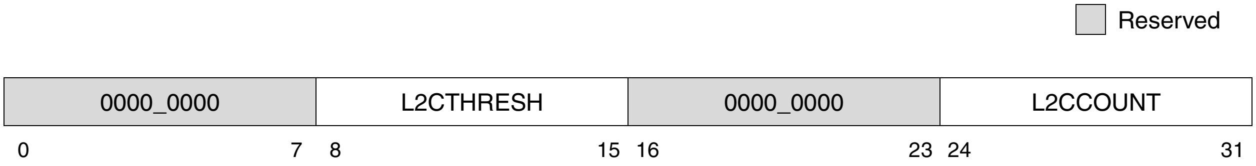


Figure 2-30. L2 Error Control Register (L2ERRCTL) for the MPC7448

Table 2-25 describes L2ERRCTL fields.

Table 2-25. L2ERRCTL Field Descriptions for the MPC7448

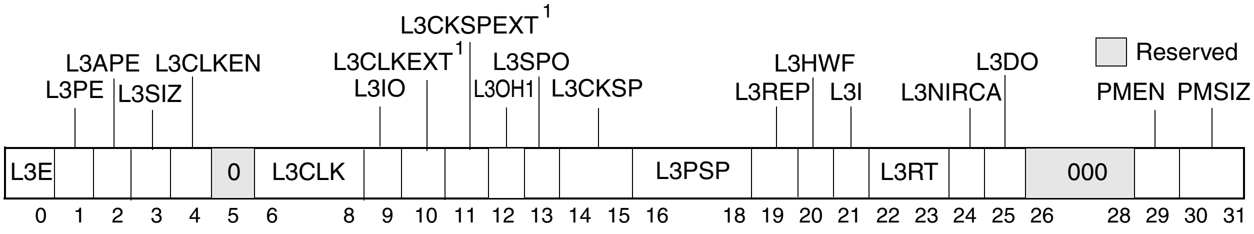
Bits	Name	Description
0–7	—	Reserved
8–15	L2CTHRESH	L2 cache threshold. Threshold value for the number of ECC single-bit errors that are detected before reporting an error condition. (read only)

Table 2-25. L2ERRCTL Field Descriptions for the MPC7448 (continued)

Bits	Name	Description
16–23	—	Reserved
24–31	L2CCOUNT	L2 count. Counts ECC single-bit errors that have been detected. If L2CCOUNT equals the ECC single-bit error trigger threshold (L2CTHRESH), an error is reported if single-bit error reporting is enabled (SPECCDIS = 0). Software can write to this field.

2.2.5.5.15 L3 Cache Control Register (L3CR)

The L3 cache control register (L3CR), shown in Figure 2-31, is a supervisor-level, implementation-specific SPR used to configure and operate the L3 cache. All L3CR bits are cleared by a hard reset or power-on reset. Note that the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448 have no L3 support.



[1] MPC7457-specific bit.

Figure 2-31. L3 Cache Control Register (L3CR) for the MPC7457

The L3 cache interface is described in Chapter 3, “L1, L2, and L3 Cache Operation.” The L3CR bits are described in Table 2-26.

Table 2-26. L3CR Field Descriptions

Bits	Name	Description
0	L3E	L3 enable 0 L3 cache operation (including snooping) disabled 1 L3 cache operation (including snooping) enabled Enables or disables L3 cache operation (including snooping) starting with the next transaction the L3 cache unit receives. Before enabling the L3 cache, the L3 clock must be configured through L3CR[L3CLK], and the L3CR[L3CLKEN] (see the hardware specifications for further details). Also, all other L3CR bits must be set appropriately. The L3 cache may need to be invalidated globally before the L3 cache is enabled.
1	L3PE	L3 data parity checking enable 0 L3 odd data parity checking disabled 1 L3 odd data parity checking enabled Enables odd parity checking for the L3 data RAM interface and on-chip tags. When L3PE is set, it allows a data parity error on the L3 interface or a parity error in the on-chip L3 tags to cause a checkstop if MSR[ME] = 0 or a machine check exception if MSR[ME] = 1. The MPC7450 always generates L3 data parity.

Table 2-26. L3CR Field Descriptions (continued)

Bits	Name	Description
2	L3APE	L3 address parity checking enable 0 L3 address parity checking disabled 1 L3 address parity checking enabled If L3CR[L3PE] = 1. enables odd parity checking for the L3 address bus interface and on-chip tags. The address parity is merged with the data parity on the L3 data parity interface pins. An address parity error on the L3 address bus will cause a checkstop if MSR[ME] = 0 or a machine check exception if MSR[ME] = 1. The MPC7450 only generates L3 address parity if L3CR[L3APE] = 1 and L3CR[L3PE] = 1.
3	L3SIZ	L3 size Should be set according to the size of the L3 cache as follows: 0 1 Mbyte 1 2 Mbyte
4	L3CLKEN	Enables the L3_CLK[0:1] signals 0 L3 clocks disabled 1 L3 clocks enabled A minimum of 100 MPC7450 clock cycles must transpire between the clearing and setting of this bit.
5	—	Reserved. Must be set by software during initialization (see Section 3.7.3.1, “Enabling the L3 Cache and L3 Initialization,” for details on when to set this bit).
6–8	L3CLK	L3 clock ratio (core-to-L3 frequency divider). Specifies the ratio between the core clock frequency and the frequency at which the L3 SRAM interface operates. See the hardware specifications for further details. The resulting L3 clock frequency cannot be slower than the clock frequency of the 60x/MPX bus interface. The following ratios are correct for the MPC7450: Note that for the MPC7457, the following ratios are correct when L3CR[L3CLKEXT] = 0: 000 ÷ 6 001 Reserved 010 ÷ 2 011 ÷ 2.5 100 ÷ 3 101 ÷ 3.5 110 ÷ 4 111 ÷ 5 Also note that for the MPC7457, the following ratios are correct when L3CR[L3CLKEXT] = 1. These ratios are not used on the MPC7450. 000 ÷ 7 001 ÷ 8 010 ÷ 4.5 011 ÷ 5.5 100 ÷ 6.5 101 ÷ 7.5 110 Reserved 111 Reserved Note these bits should only be changed after at least 100 MPC7450 clock cycles have transpired after L3CLKEN has been cleared.
9	L3IO	L3 instruction-only mode 0 Instruction-only operation in the L3 cache disabled 1 Instruction-only operation in the L3 cache enabled Enables instruction-only operation in the L3 cache. When this bit is set, only instruction accesses can be cached in the L3 cache. Data addresses already in the cache will still hit for the L3 data cache. When both L3CR[L3DO] and L3CR[L3IO] are set, the L3 cache is effectively locked.

Table 2-26. L3CR Field Descriptions (continued)

Bits	Name	Description
10	L3CLKEXT	L3 clock ratio extension (MPC7457-specific) 0 Used with L3CR[L3CLK] to determine the clock ratio encodings. See L3CR[L3CLK] field description. 1 Used with L3CR[L3CLK] to determine the other clock ratio encodings. See L3CR[L3CLK] field description. Note, that the L3CR[10] bit is reserved on the MPC7450 and is used as an L3 clock ratio extension only on the MPC7457.
11	L3CKSPEXT	L3 Clock sample point extension (MPC7457-specific) 0 Used with L3CR[L3CKSP] to determine the clock ratio encodings. See L3CR[L3CKSP] field description. 1 Used with L3CR[L3CKSP] to determine the other clock ratio encodings. See L3CR[L3CKSP] field description. Note, that the L3CR[11] bit is reserved on the MPC7450 and is used as an L3 clock sample point extension only on the MPC7457.
12	—	Reserved on the MPC7450, MPC7451, MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448.
	L3OH1	MPC7455: L3 output hold 1. This bit, in conjunction with L3OH0 (see Table 2-12), configures output hold time for address, data, and control signals driven by the MPC7455 to the L3 data RAMs. They should generally be set according to the SRAM's input hold time requirements. See the *MPC7455 RISC Microprocessor Hardware Specifications* for specific output hold times. All others: Reserved
13	L3SPO	L3 sample point override 0 L3 sample point override disabled 1 L3 sample point override enabled Adds one L3 clock of latency to a read operation, and may be required for future generation SRAMs.
14–15	L3CKSP	L3 clock sample point. Specifies in which L3 clock cycle the L3 accumulator samples data from the receive latches. See the hardware specifications for further clarification. The following values are correct for the MPC7450. Note that only for the MPC7457, the following values are correct when L3CR[L3CKSPEXT] = 0: 00 2 clocks 01 3 clocks 10 4 clocks 11 5 clocks Also note that only for the MPC7457, the following values are correct when L3CR[L3CKSPEXT] = 1. These values are not used on the MPC7450. 00 6 clocks 01 7 clocks 10 8 clocks 11 9 clocks

Table 2-26. L3CR Field Descriptions (continued)

Bits	Name	Description
16–18	L3PSP	L3 P-clock sample point. Specify the processor clock cycle in which the L3 accumulator samples data from the receive latches. See Section 3.7.3.8, "L3 Cache Clock and Timing Controls," and the hardware specifications for further clarification. 000 0 clocks 001 1 clock 010 2 clocks 011 3 clocks 100 4 clocks 101 5 clocks 110 Reserved on the MPC7450. For the MPC7457, it is 6 clocks. 111 Reserved on the MPC7450. For the MPC7457, it is 7 clocks.
19	L3REP	L3 replacement algorithm 0 When this bit is cleared, the default replacement algorithm is used 1 When this bit is set, the secondary replacement algorithm (3-bit running free counter) is used. For details on the replacement algorithm, see Section 3.7.7.4, "L3 Cache Replacement Selection."
20	L3HWF	L3 hardware flush 0 L3 hardware flush disabled 1 L3 hardware flush enabled When L3CR[L3HWF] is set, the L3 begins a flush by starting with way 0. Each modified block (sector) is cast out as it is flushed. After the first line in the first way is flushed, the next way (same index) is flushed. When all ways for a given index have been flushed, the index is incremented and same process occurs for line 1, etc. During a hardware flush, the L3 services both read hits and bus snooping. The hardware flush completes when all blocks in the L3 have a status of invalid. At this time, the processor automatically clears L3CR[L3HWF]. However, even though the hardware flush is considered complete, there may still be outstanding castouts queued in the BSQ waiting to be performed to the system interface. See Section 3.6.3.1.5, "Flushing of L1, L2, and L3 Caches," for more information.
21	L3I	L3 global invalidate 0 Do not globally invalidate the L3 1 Globally invalidate the L3 Invalidates the L3 cache globally by clearing the L3 status bits. This bit must not be set while the L3 cache is enabled. Note that L3I is automatically cleared when the global invalidate completes.
22–23	L3RT	L3 SRAM type. Configures the L3 SRAM interface for the type of synchronous SRAMs used: • MSUG dual data rate SRAMs that provide data synchronous to the L3_ECHO_CLK input signals to the MPC7450 and on each clock edge • Late-write SRAMs which are required by the MPC7450 to be of the pipelined (register-register) configurations • Pipeline burst SRAMs, referred to as PB2-type SRAMs For burst RAM selections, the MPC7450 does not use the burst feature of the SRAM; it generates an address for each access. 00 MSUG2 DDR SRAM 01 Pipelined (register-register) synchronous late-write SRAM 10 Reserved 11 PB2 SRAM

Table 2-26. L3CR Field Descriptions (continued)

Bits	Name	Description
24	L3NIRCA	L3 non-integer ratios clock adjustment for the SRAM. When this bit is set, the AC timing of L3_CLK[0:1] is changed. 0 L3 SRAM clock timing is unchanged (default). 1 The L3_CLK[0:1] signals occur earlier relative to the MPC7450 driving the L3 address, control and data buses in non-integer L3 clock ratios. Because of the way that the L3_CLK[0:1] signals are internally derived, these signals may be driven slightly later (one-eight of a core clock) with non-integer clock ratios than they would normally be with an integer L3 clock ratio. This can potentially cause AC hold timing problems on the L3 interface if the timing margins are very small. This signal corrects for this phenomenon by causing the MPC7450 to drive the L3_CLK[0:1] signals one-quarter of a core clock earlier at the expense of AC setup timing. See the hardware specifications for further clarification.
25	L3DO	L3 data-only mode 0 Data-only operation in the L3cache disabled 1 Data-only operation in the L3 cache enabled Enables data-only operation in the L3 cache. When this bit is set, only data accesses can be cached in the L3 cache. Instruction cache operations are serviced for instruction addresses already in the L3 cache; however, the L3 cache is not reloaded for instruction cache misses. Note that setting both L3CR[L3DO] and L3CR[L3IO] effectively locks the L3 cache.
26–28	—	Reserved
29	PMEN	Private memory enable 0 Private memory disabled 1 Private memory enabled When this bit is set, the MPC7450 does not manage the coherency of the contents of private memory. Thus, the software must manage addresses mapped to this range very carefully.
30–31	PMSIZ	Private memory size For the MPC7451, L3CR[31] is used: 0 1 Mbyte 1 2 Mbytes Note that L3CR[30] bit is reserved on the MPC7451 and MPC7455. For the MPC7457, L3CR[30—31] is used: 00 1 Mbyte 01 2 Mbytes 10 4 Mbytes 11 Reserved

The L3CR register can be accessed with the **mtspr** and **mfspr** instructions using SPR 1018.

2.2.5.5.16 L3 Cache Output Hold Control Register (L3OHCR)—MPC7457-Specific

The L3 cache output hold control register (L3OHCR), shown in Figure 2-32, is a supervisor-level, implementation-specific SPR used to control the output AC timing of the L3 cache interface of the MPC7457. All L3OHCR bits are cleared by a hard reset or power-on reset. For more information, see the *MPC7457 RISC Microprocessor Hardware Specifications*.

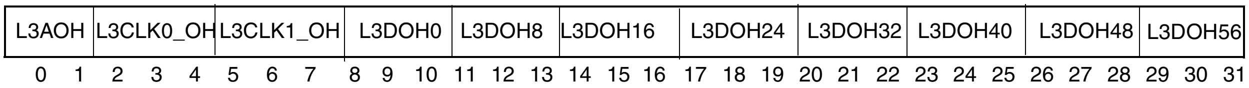


Figure 2-32. L3 Cache Output Hold Control Register (L3OHCR) for the MPC7457

The L3 cache interface is described in Chapter 3, “L1, L2, and L3 Cache Operation.” The L3OHCR bits are described in Table 2-27.

Table 2-27. L3OHCR Field Descriptions

Bits	Name	Description
0–1	L3AOH	L3 address output hold. These bits configure output hold time for address and control signals driven by the MPC7457 to the L3 data RAMs. They should generally be set according to the SRAM’s input hold time requirements. See the *MPC7457 RISC Microprocessor Hardware Specifications* for specific output hold times.
2–4	L3CLK0_OH	L3_CLK0 output hold. These bits configure output hold time for L3_CLK0 signal driven by the MPC7457 to the L3 data RAMs. They should generally be set according to the SRAM’s input hold time requirements. See the *MPC7457 RISC Microprocessor Hardware Specifications* for specific output hold times.
5–7	L3CLK1_OH	L3_CLK1 output hold. These bits configure output hold time for L3_CLK1 signal driven by the MPC7457 to the L3 data RAMs. They should generally be set according to the SRAM’s input hold time requirements. See the *MPC7457 RISC Microprocessor Hardware Specifications* for specific output hold times.
8–10	L3DOH0	L3_DATA[00:07]/L3_DP[0] output hold. These bits configure output hold time for L3_DATA[00:07] and L3_DP[0] signals driven by the MPC7457 to the L3 data RAMs. They should generally be set according to the SRAM’s input hold time requirements. See the *MPC7457 RISC Microprocessor Hardware Specifications* for specific output hold times.
11–13	L3DOH8	L3_DATA[08:15]/L3_DP[1] output hold. These bits configure output hold time for L3_DATA[8:15] and L3_DP[1] signals driven by the MPC7457 to the L3 data RAMs. They should generally be set according to the SRAM’s input hold time requirements. See the *MPC7457 RISC Microprocessor Hardware Specifications* for specific output hold times.
14–16	L3DOH16	L3_DATA[16:23]/L3_DP[2] output hold. These bits configure output hold time for L3_DATA[16:23] and L3_DP[2] signals driven by the MPC7457 to the L3 data RAMs. They should generally be set according to the SRAM’s input hold time requirements. See the *MPC7457 RISC Microprocessor Hardware Specifications* for specific output hold times.
17–19	L3DOH24	L3_DATA[24:31]/L3_DP[3] output hold. These bits configure output hold time for L3_DATA[24:31] and L3_DP[3] signals driven by the MPC7457 to the L3 data RAMs. They should generally be set according to the SRAM’s input hold time requirements. See the *MPC7457 RISC Microprocessor Hardware Specifications* for specific output hold times.
20–22	L3DOH32	L3_DATA[32:39]/L3_DP[4] output hold. These bits configure output hold time for L3_DATA[32:39] and L3_DP[4] signals driven by the MPC7457 to the L3 data RAMs. They should generally be set according to the SRAM’s input hold time requirements. See the *MPC7457 RISC Microprocessor Hardware Specification*s for specific output hold times.
23–25	L3DOH40	L3_DATA[40:47]/L3_DP[5] output hold. These bits configure output hold time for L3_DATA[40:47] and L3_DP[5] signals driven by the MPC7457 to the L3 data RAMs. They should generally be set according to the SRAM’s input hold time requirements. See the *MPC7457 RISC Microprocessor Hardware Specifications* for specific output hold times.
26–28	L3DOH48	L3_DATA[48:55]/L3_DP[6] output hold. These bits configure output hold time for L3_DATA[48:55] and L3_DP[6] signals driven by the MPC7457 to the L3 data RAMs. They should generally be set according to the SRAM’s input hold time requirements. See the *MPC7457 RISC Microprocessor Hardware Specifications* for specific output hold times.
29–31	L3DOH56	L3_DATA[56:63]/L3_DP[7] output hold. These bits configure output hold time for L3_DATA[56:63] and L3_DP[7]signals driven by the MPC7457 to the L3 data RAMs. They should generally be set according to the SRAM’s input hold time requirements. See the *MPC7457 RISC Microprocessor Hardware Specifications* for specific output hold times.

The L3OHCR register is specific to the MPC7457 and can be accessed with the **mtspr** and **mfspr** instructions using SPR 1000.

2.2.5.5.17 L3 Cache Input Timing Control (L3ITCR0)

The L3 cache input timing control register (L3ITCR0), shown in Figure 2-33, is a supervisor-level, implementation-specific SPR used to control the input AC timing of the L3 cache interface of the MPC7450. For the MPC7457, the L3ITCR0, shown in Figure 2-34, is used to control the input AC timing of L3_DATA[0:15] and L3_DP[0:1] signals of the L3 cache interface. All L3ITCR0 bits are cleared by a hard reset or power-on reset and configured when the L3 clock is enabled. Note: This register is intended for factory use. Writing to this register will override the default input AC timing of the L3 cache interface and may cause improper operation of the L3 cache.

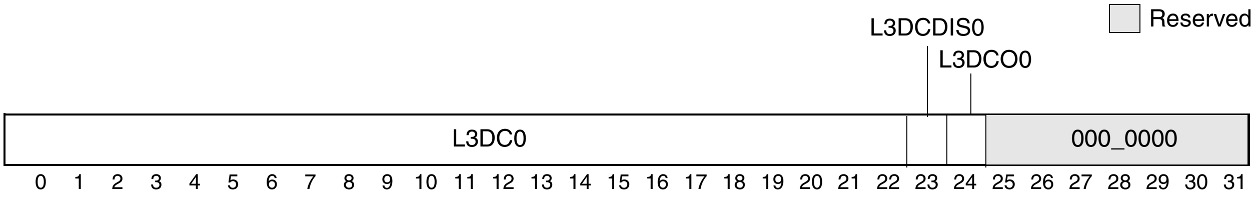


Figure 2-33. L3 Cache Control Register (L3ITCR0) for the MPC7451 and MPC7455

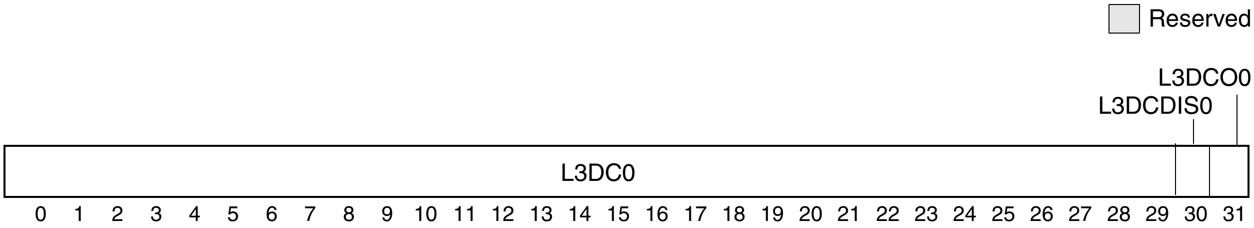


Figure 2-34. L3 Cache Control Register (L3ITCR0) for the MPC7457

The L3 cache interface is described in Chapter 3, “L1, L2, and L3 Cache Operation.” The L3ITCR0 bits for the MPC7451 and MPC7455 are described in Table 2-28.

Table 2-28. L3ITCR0 Field Descriptions for the MPC7451 and MPC7455

Bits	Name	Description
0–22	L3DC0	L3 delay count. These bits contain a delay counter value used to internally align the L3_ECHO_CLK inputs to data being returned from the SRAM.
23	L3DCDIS0	L3 delay counter disable. Setting this bit disables the automic delay count configuration. Always read as 0.
24	L3DCO0	L3 delay counter override. Setting this bit overrides the automatic configuration value of the delay count. Always read as 0.
25–31		Reserved.

The L3ITCR0 bits for the MPC7457 are described in Table 2-29.

Table 2-29. L3ITCR0 Field Descriptions for the MPC7457

Bits	Name	Description
0–29	L3DC0	L3 delay count. These bits contain a delay counter value used to internally align the L3_ECHO_CLK0 input to data being returned on L3_DATA[0:15] and L3_DP[0:1] from the SRAM.
30	L3DCDIS0	L3 delay counter disable. Setting this bit disables the automic delay count configuration. Always read as 0.
31	L3DCO0	L3 delay counter override. Setting this bit overrides the automatic configuration value of the delay count. Always read as 0.

The L3ITCR0 register can be accessed with the **mtspr** and **mfspr** instructions using SPR 984.

2.2.5.5.18 L3 Cache Input Timing Control (L3ITCR1)—MPC7457-Specific

The L3 cache input timing control register (L3ITCR1), shown in Figure 2-35, is a supervisor-level, implementation-specific SPR used to control the input AC timing of L3_DATA[16:31] and L3_DP[2:3] signals of the L3 cache interface of the MPC7457. All L3ITCR1 bits are cleared by a hard reset or power-on reset and configured when the L3 is enabled. Note: This register is intended for factory use. Writing to this register will override the default input AC timing of the L3 cache interface and may cause improper operation of the L3 cache.

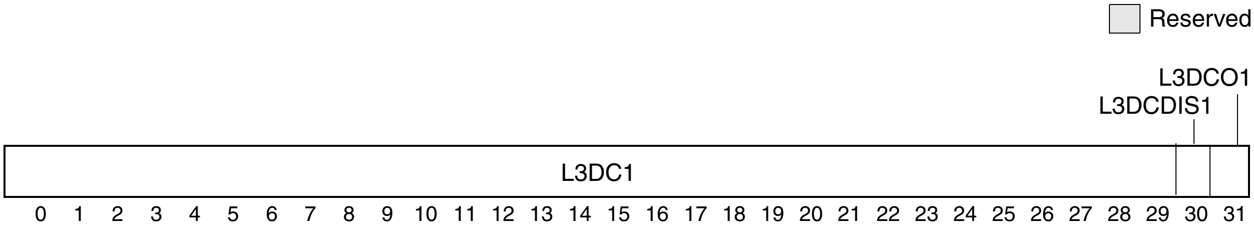


Figure 2-35. L3 Cache Control Register (L3ITCR1) for the MPC7457

The L3 cache interface is described in Chapter 3, “L1, L2, and L3 Cache Operation.” The L3ITCR1 bits for the MPC7457 are described in Table 2-30.

Table 2-30. L3ITCR1 Field Descriptions for the MPC7457

Bits	Name	Description
0–29	L3DC1	L3 delay count. These bits contain a delay counter value used to internally align the L3_ECHO_CLK0 input to data being returned on L3_DATA[0:15] and L3_DP[0:1] from the SRAM.
30	L3DCDIS1	L3 delay counter disable. Setting this bit disables the automic delay count configuration. Always read as 0.
31	L3DCO1	L3 delay counter override. Setting this bit overrides the automatic configuration value of the delay count. Always read as 0.

The L3CR register can be accessed with the **mtspr** and **mfspr** instructions using SPR 1001.

2.2.5.5.19 L3 Cache Input Timing Control (L3ITCR2)—MPC7457-Specific

The L3 cache input timing control register (L3ITCR2), shown in Figure 2-36, is a supervisor-level, implementation-specific SPR used to control the input AC timing of L3_DATA[32:47] and L3_DP[4:5] signals of the L3 cache interface of the MPC7457. All L3ITCR2 bits are cleared by a hard reset or power-on reset and configured when the L3 is enabled. Note: This register is intended for factory use. Writing to this register will override the default input AC timing of the L3 cache interface and may cause improper operation of the L3 cache.

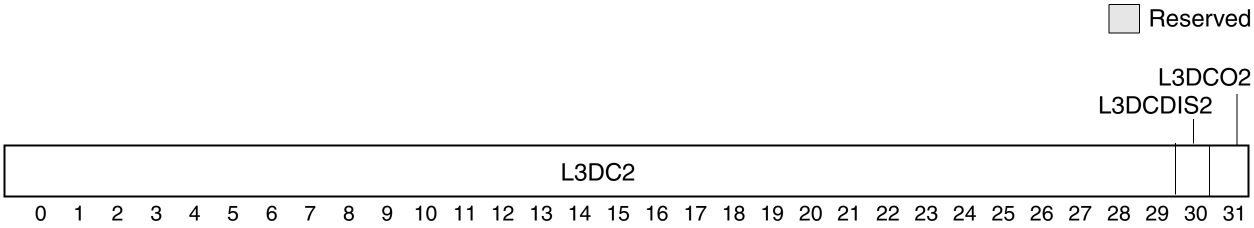


Figure 2-36. L3 Cache Control Register (L3ITCR2) for the MPC7457

The L3 cache interface is described in Chapter 3, “L1, L2, and L3 Cache Operation.” The L3ITCR2 bits for the MPC7457 are described in Table 2-31.

Table 2-31. L3ITCR2 Field Descriptions for the MPC7457

Bits	Name	Description
0–29	L3DC2	L3 delay count. These bits contain a delay counter value used to internally align the L3_ECHO_CLK0 input to data being returned on L3_DATA[0:15] and L3_DP[0:1] from the SRAM.
30	L3DCDIS2	L3 delay counter disable. Setting this bit disables the automic delay count configuration. Always read as 0.
31	L3DCO2	L3 delay counter override. Setting this bit overrides the automatic configuration value of the delay count. Always read as 0.

The L3ITCR2 register can be accessed with the **mtspr** and **mfspr** instructions using SPR 1002.

2.2.5.5.20 L3 Cache Input Timing Control (L3ITCR3)—MPC7457-Specific

The L3 cache input timing control register (L3ITCR3), shown in Figure 2-37, is a supervisor-level, implementation-specific SPR used to control the input AC timing of L3_DATA[48:63] and L3_DP[6:7] signals of the L3 cache interface of the MPC7457. All L3ITCR3 bits are cleared by a hard reset or power-on reset and configured when the L3 is enabled. Note: This register is intended for factory use. Writing to this register will override the default input AC timing of the L3 cache interface and may cause improper operation of the L3 cache.

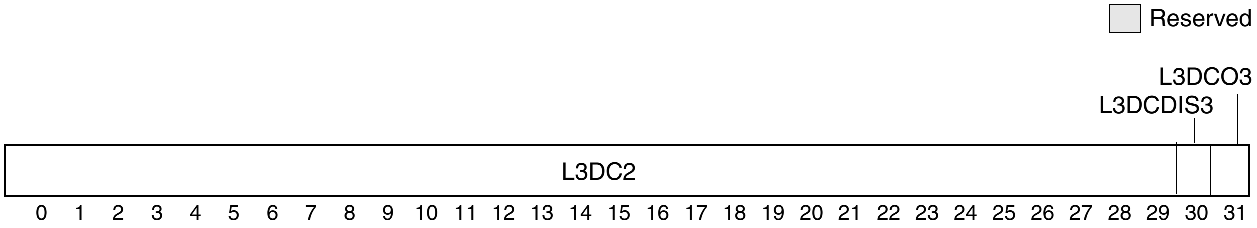


Figure 2-37. L3 Cache Control Register (L3ITCR3) for the MPC7457

The L3 cache interface is described in Chapter 3, “L1, L2, and L3 Cache Operation.” The L3ITCR3 bits for the MPC7457 are described in Table 2-32.

Table 2-32. L3ITCR3 Field Descriptions for the MPC7457

Bits	Name	Description
0–22	L3DC3	L3 delay count. These bits contain a delay counter value used to internally align the L3_ECHO_CLK inputs to data being returned from the SRAM.
23	L3DCDIS3	L3 delay counter disable. Setting this bit disables the automic delay count configuration. Always read as 0.
24	L3DCO3	L3 delay counter override. Setting this bit overrides the automatic configuration value of the delay count. Always read as 0.
25–31		Reserved.

The L3CR register can be accessed with the **mtspr** and **mfspr** instructions using SPR 1003.

2.2.5.5.21 Instruction Cache and Interrupt Control Register (ICTRL)

The instruction cache and interrupt control register (ICTRL), shown in Figure 2-38, is used in configuring interrupts and error reporting for the instruction and data caches. It is accessed as SPR 1011. Control and access to the ICTRL is through the privileged **mtspr**/**mfspr** instructions.

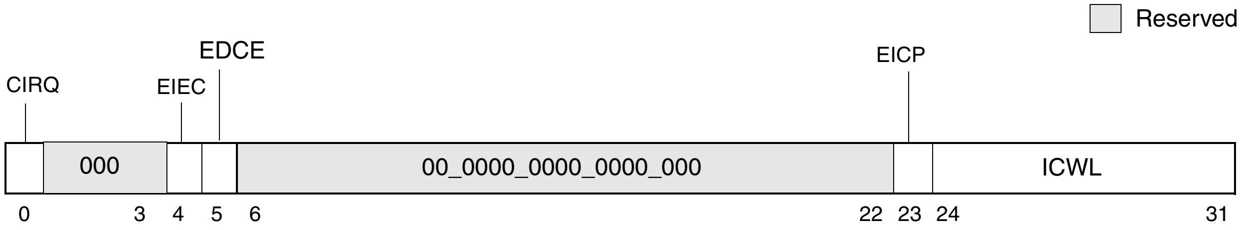


Figure 2-38. Instruction Cache and Interrupt Control Register (ICTRL)

Table 2-33 describes the bit fields for the ICTRL register.

Table 2-33. ICTRL Field Descriptions

Bits	Name	Description
0	CIRQ	CPU interrupt request 0 No processor interrupt request forwarded to exception handling. If software clears the CIRQ bit, it does not cancel a previously sent interrupt request. 1 Processor interrupt request sent to the exception mechanism. This interrupt request is combined with the external interrupt request (assertion of $\overline{\text{INT}}$). When external interrupts are enabled with the MSR[EE] bit and either this bit is set or $\overline{\text{INT}}$ is asserted, the MPC7450 takes the external interrupt exception. If there is more than one interrupt request pending (CIRQ and $\overline{\text{INT}}$ is asserted), only one interrupt is taken. When the external interrupt exception is taken, the ICTRL[CIRQ] bit is automatically cleared. Note that this mechanism allows a processor to interrupt itself. If software leaves CIRQ set while waiting for the interrupt to be taken, it can poll CIRQ to determine when the interrupt has been taken.
1–3	—	Reserved
4	EIEC[1]	Instruction cache parity error enable 0 When the bit is cleared, any parity error in the L1 instruction cache is masked and does not cause machine checks or checkstop 1 Enables instruction cache parity errors. When an instruction cache parity error occurs, a machine check exception is taken if MSR[ME] = 1. When this condition occurs, SRR1[1] is set. For details on the machine check exception see Section 4.6.2, “Machine Check Exception (0x00200).”
5	EDCE[2]	Data cache parity error enable 0 When the bit is cleared, any parity error in the L1 data cache is masked and does not cause machine checks or checkstop 1 Enables data cache parity errors. When a data cache parity error occurs, a machine check exception is taken if MSR[ME] = 1. When this condition occurs, SRR1[2] is set. For details on the machine check exception see Section 4.6.2, “Machine Check Exception (0x00200).”
6–8	—	Reserved. Normally cleared, used in debug, writing nonzero values may cause boundedly undefined results.
9–22	—	Reserved. Read as zeroes and ignores writes.

Table 2-33. ICTRL Field Descriptions (continued)

Bits	Name	Description
23	EICP	Enable instruction cache parity checking 0 Instruction cache parity disabled 1 When the EICP bit is set, the parity of any instructions fetched from the L1 instruction cache is checked. Any errors found are reported as instruction cache parity errors in SRR1. If EICE is also set, these instruction cache errors cause a machine check or checkstop. If either EICP or EICE is cleared, instruction cache parity is ignored. Note that when parity checking and error reporting are both enabled, errors are reported even on speculative fetches that are never actually executed. Correct instruction cache parity is always loaded into the L1 instruction cache regardless of whether checking is enabled or not.
24–31	ICWL[1]	Instruction cache way lock 0 Instruction cache way lock disabled. 1 Instruction cache way lock enabled. Each bit in ICWL corresponds to a way of the L1 instruction cache. Setting a bit locks the corresponding way in the instruction cache. Setting all 8 bits of ICWL is equivalent to locking the entire instruction cache. When all 8 ICWL bits are set, MPC7450 behaves the same as when HID0[ILOCK] is set. See Section 2.2.5.1, “Hardware Implementation-Dependent Register 0 (HID0),” for details. See Chapter 3, “L1, L2, and L3 Cache Operation,” for suggestions on how to keep the PLRU replacement algorithm symmetrical, and for synchronization requirements for modifying ICWL.

[1] A context synchronizing instruction must precede and follow a mtspr.

[2] A dssall and sync must precede a mtspr and then a sync and context synchronizing instruction must follow. Note that if a user is not using the AltiVec data streaming instructions, then a dssall is not necessary prior to accessing the ICTRL[EDCE] bit.

ICTRL can be accessed with the **mtspr** and **mfspr** instructions using SPR 1011.

2.2.5.5.22 Load/Store Control Register (LDSTCR)

The load/store control register (LDSTCR) provides a way to lock the ways for the L1 data cache. The LDSTCR is shown in Figure 2-39.

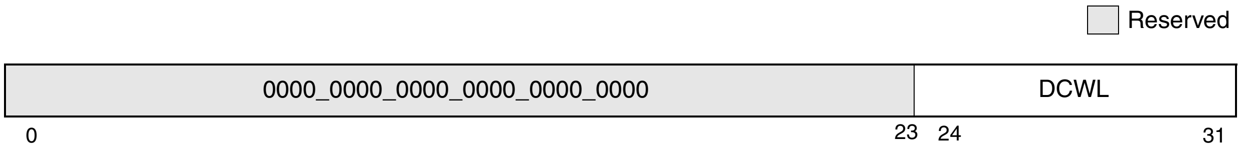


Figure 2-39. Load/Store Control Register (LDSTCR)

Table 2-34 describes the bit fields for the LDSTCR register.

Table 2-34. LDSTCR Field Descriptions

Bits	Name	Description
0–23	—	Reserved. Writing nonzero values may cause boundedly undefined results.
24–31	DCWL	Data cache way lock 0 Each cleared bit corresponds to a way not being locked in the L1 data cache. 1 Each set bit locks the corresponding way in the L1data cache. When DCWL[24–31] are all set, it is equivalent to locking the entire L1 data cache and the MPC7450 behaves the same as if HID0[DLOCK] is set. “Chapter 3, “L1, L2, and L3 Cache Operation,” describes how to keep the PLRU replacement algorithm symmetrical and for more information on synchronization requirements with LDSTCR.

The LDSTCR register can be accessed with the **mtspr** and **mfspr** instructions using SPR 1016. For synchronization requirements on the register see Section 2.4.2.4, “Synchronization.”

2.2.5.5.23 L3 Private Memory Address Register (L3PM)

The L3 private address register (L3PM), shown in Figure 2-40, is a supervisor-level, implementation-specific SPR used to configure the base address of the range of addresses that defines the L3 private memory space. It is cleared by a hard reset or power-on reset.

Note that the L3CR[PMEN] and L3CR[PMSIZ] bits control aspects of the MPC7450 private memory feature. If extended addressing is disabled, the upper four bits of PMBA must be zero in order to be able to match the internal value of A0–A3 (which are zero). Refer to Section 3.7.8, “L3 Private Memory Operation,” for more details on the L3 private memory.

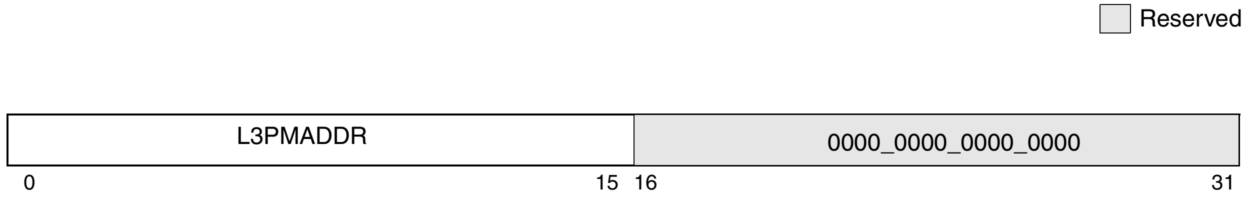


Figure 2-40. L3 Private Memory Address Register (L3PM)

The L3PM bits are described in Table 2-35.

Table 2-35. L3PM Field Descriptions

Bits	Name	Description
0–15	L3PMADDR	L3 base address of L3 private memory. L3PMADDR contain the base address of the range of addresses used in the L3 private memory. Specific bits of the L3PM[L3PMADDR] field are used based on the memory size as follows: 1 MB L3PM[0–15] 2 MB L3PM[0–14]
16–31	—	Reserved

The L3PM register can be accessed with the **mtspr** and **mfspr** instructions using SPR 983. For synchronization requirements on the register see Section 2.4.2.4, “Synchronization.”

2.2.5.6 Instruction Address Breakpoint Register (IABR)

The instruction address breakpoint register (IABR), shown in Table 2-41, supports the instruction address breakpoint exception. When this exception is enabled, instruction fetch addresses are compared with an effective address stored in the IABR. If the word specified in the IABR is fetched, the instruction breakpoint handler is invoked. The instruction that triggers the breakpoint does not execute before the handler is invoked. For more information, see Section 4.6.16, “Instruction Address Breakpoint Exception (0x01300).” The IABR can be accessed with **mtspr** and **mfspr** using SPR 1010. The MPC7450 requires that an **mtspr**[IABR] be followed by a context synchronizing instruction. The MPC7450 may not generate a breakpoint response for that context synchronizing instruction if the breakpoint was enabled by **mtspr**[IABR] immediately preceding it. The MPC7450 cannot block a breakpoint response on the context synchronizing instruction if the breakpoint was disabled by **mtspr**[IABR] immediately preceding it. For more information on synchronization see Section 2.4.2.4.1, “Context Synchronization.”

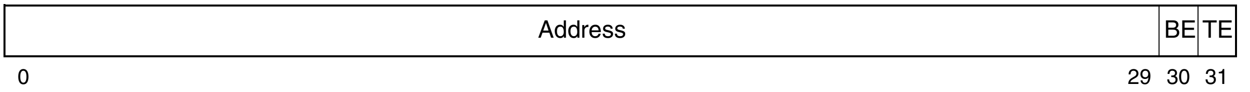


Figure 2-41. Instruction Address Breakpoint Register (IABR)

The IABR bits are described in Table 2-36.

Table 2-36. IABR Field Descriptions

Bits[1]	Name	Description
0–29	Address	Word instruction breakpoint address to be compared with EA[0–29] of the next instruction
30	BE	Breakpoint enabled. Setting this bit enables breakpoint address checking.
31	TE	Translation enable IABR[TE] must equal MSR[IR] in order for a match to be signaled. When IABR[TE] and MSR[IR] = 0 or when IABR[TE] and MSR[IR] = 1, a match is signaled.

[1] A context synchronizing instruction must follow a mtspr.

2.2.5.7 Memory Management Registers Used for Software Table Searching

This section describes the registers used by the MPC7450 when software searching is enabled (HID0[STEN] = 1) and a TLB miss exception occurs. Software table searching is described in detail in Chapter 5, “Memory Management.”

2.2.5.7.1 TLB Miss Register (TLBMISS)

The TLBMISS register is automatically loaded by the MPC7450 when software searching is enabled (HID0[STEN] = 1) and a TLB miss exception occurs. Its contents are used by the TLB miss exception handlers (the software table search routines) to start the search process. Note that the MPC7450 always loads a big-endian address into the TLBMISS register. This register is read-only. The TLBMISS register has the format shown in Figure 2-42 for the MPC7450.

Figure 2-42. TLBMISS Register for MPC7450

Table 2-37 described the bits in the TLBMISS register.

Table 2-37. TLBMISS Register—Field and Bit Descriptions for the MPC7450

Bits	Name	Description
0–30	PAGE	Effective page address Stores EA[0–30] of the access that caused the TLB Miss exception.
31	LRU	Least recently used way of the addressed TLB set The LRU bit can be loaded into bit 31 of **r**B, prior to execution of **tlbli** or **tlbld** to select the way to be replaced for a TLB miss. However, this value should be inverted in **r**B prior to execution of **tlbli** or **tlbld** for a TLB miss exception caused by the need to update the C-bit.

TLBMISS can be accessed with **mtspr** and **mfspr** using SPR 980.

2.2.5.7.2 Page Table Entry Registers (PTEHI and PTELO)

The PTEHI and PTELO registers are used by the **tlbld** and **tlbli** instructions to create a TLB entry. When software table searching is enabled (HID0[STEN] = 1) and a TLB miss exception occurs, the bits of the page table entry (PTE) for this access are located by software and saved in the PTE registers. Figure 2-43 shows the format for two supervisor registers, PTEHI and PTELO, respectively.

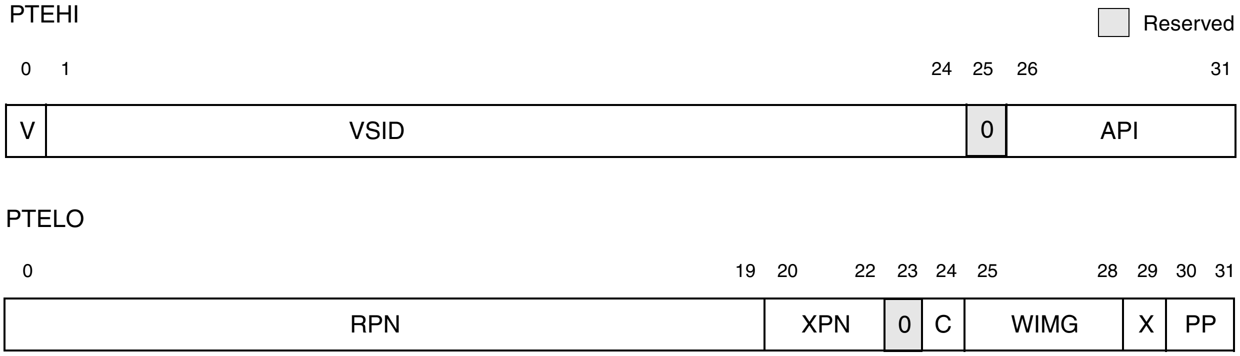


Figure 2-43. PTEHI and PTELO Registers—Extended Addressing

Note that the contents of PTEHI are automatically loaded when any of the three software table search exceptions is taken. PTELO is loaded by the software table search routines (the TLB miss exception handlers) based on the valid PTE located in the page tables prior to execution of the **tlbli** or **tlbld** instruction.

Table 2-38 lists the corresponding bit definitions for the PTEHI and PTELO registers.

Table 2-38. PTEHI and PTELO Bit Definitions

Register	Bit	Name	Description
PTEHI	0	V	Entry valid (V = 1) or invalid (V = 0). Always set by the processor on a TLB miss exception.
	1–24	VSID	Virtual segment ID. The corresponding SR[VSID] field is copied to this field.
	25	—	Reserved. Corresponds to the hash function identifier in PTE.
	26–31	API	Abbreviated page index. TLB miss exceptions will set this field with bits from TLBMISS[4–9] which are bits from the effective address for the access that caused the software table search operation. The **tlbld** and **tlbli** instructions will ignore the API bits in PTEHI register and get the API from instruction's operand**, r**B. However, for future compatibility, the API in **r**B should match the PTEHI[API].
PTELO	0–19	RPN	Physical page number
	20–22	XPN	Extended page number The XPN field provides the physical address bits, PA[0–2].
	23	—	Reserved
	24	C	Changed bit
	25–28	WIMG	Memory / cache control bits
	29	X	Extended page number The X field provides the physical address bit 3, PA[3].
	30–31	PP	Page protection bits

Note that PTELO[23] corresponds to the reference bit in a PTE. The reference bit is not stored in the page tables, so this bit is ignored in the PTELO register. All the other bits in PTELO correspond to the bits in the low word of the PTE. When extended addressing is not enabled, (HID0[XAEN] = 0), the software must clear the PTELO[XPI] and PTELO[X] bits; otherwise whatever values are in the fields become the four most-significant bits of the physical address. **Note:** The PTEHI register is accessed with **mtspr** and **mfspr** as SPR 981 and PTELO is accessed as SPR 982.

2.2.5.8 Thermal Management Register

The MPC7450 provides an instruction cache throttling mechanism to effectively reduce the instruction execution rate without the complexity and overhead of dynamic clock control. When used with the dynamic power management, instruction cache throttling provides the system designer with a flexible way to control device temperature while allowing the processor to continue operating.

2.2.5.8.1 Instruction Cache Throttling Control Register (ICTC)

Reducing the rate of instruction fetching can control junction temperature without the complexity and overhead of dynamic clock control. System software can control instruction forwarding by writing a nonzero value to the ICTC register, a supervisor-level register shown in Figure 2-44. The overall junction temperature reduction comes from the dynamic power management of each functional unit when the MPC7450 is idle in between instruction fetches. Phase-locked loop (PLL) and delay-locked loop (DLL) configurations are unchanged.

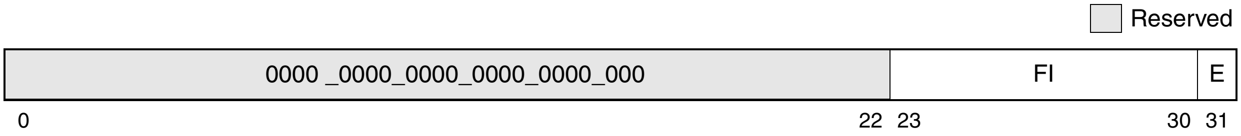


Figure 2-44. Instruction Cache Throttling Control Register (ICTC)

Table 2-39 describes the bit fields for the ICTC register.

Table 2-39. ICTC Field Descriptions

Bits	Name	Description
0–22	—	Reserved. The bits should be cleared.
23–30	INTERVAL	Instruction forwarding interval expressed in processor clocks. When throttling is enabled, the interval field specifies the minimum number of cycles between instructions being dispatched. (MPC7450 dispatches one instruction every INTERVAL cycle.) The minimum interval for throttling control is 2 cycles. 0x00, 0x01, 0x02 One instruction dispatches every 2 processor clocks. 0x03 One instruction dispatches every 3 processor clocks ... 0xFF One instruction dispatches every 255 processor clocks.
31	E	Enable instruction throttling 0 Instructions dispatch normally. 1 Only one instruction dispatches every INTERVAL cycles.

Instruction cache throttling is enabled by setting ICTC[E] and writing the instruction forwarding interval into ICTC[INTERVAL]. Note when instruction cache throttling is enabled to reduce overall junction temperature, the performance does degrade. A context synchronizing instruction should be executed after a move to the ICTC register to ensure that it has taken effect. Enabling, disabling, and changing the instruction forwarding interval affect instruction forwarding immediately.

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

2.2.5.9 Performance Monitor Registers

This section describes the registers used by the performance monitor, which is described in Chapter 11, “Performance Monitor.”

2.2.5.9.1 Monitor Mode Control Register 0 (MMCR0)

The monitor mode control register 0 (MMCR0), shown in Figure 2-45, is a 32-bit SPR provided to specify events to be counted and recorded. If the state of MSR[PR] and MSR[PMM] matches a state specified in MMCR0, then counting is enabled see Section 11.4, “Event Counting,” for further details. The MMCR0 can be accessed only in supervisor mode. User-level software can read the contents of MMCR0 by issuing an **mfspr** instruction to UMMCR0, described in Section 2.2.5.9.2, “User Monitor Mode Control Register 0 (UMMCR0).”

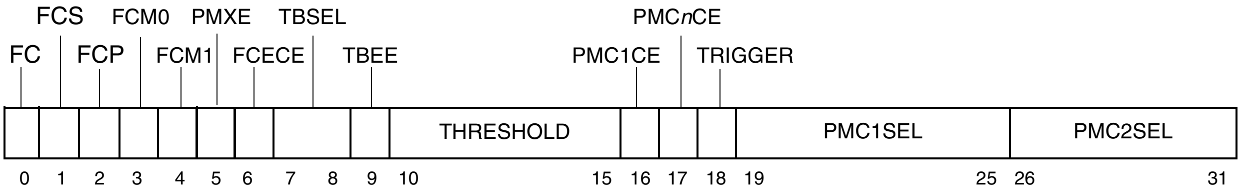


Figure 2-45. Monitor Mode Control Register 0 (MMCR0)

This register is automatically cleared at power-up. Reading this register does not change its contents. Table 2-40 describes MMCR0 fields.

Table 2-40. MMCR0 Field Descriptions

Bits	Name	Description
0	FC	Freeze counters 0 The PMCs are incremented (if permitted by other MMCR bits). 1 The PMCs are not incremented (performance monitor counting is disabled). The processor sets this bit when an enabled condition or event occurs and MMCR0[FCECE] = 1. Note that SIAR is not updated if performance monitor counting is disabled.
1	FCS	Freeze counters in supervisor mode 0 The PMCs are incremented (if permitted by other MMCR bits). 1 The PMCs are not incremented if MSR[PR] = 0.
2	FCP	Freeze counters in user mode 0 The PMCs are incremented (if permitted by other MMCR bits). 1 The PMCs are not incremented if MSR[PR] = 1.
3	FCM1	Freeze counters while mark = 1 0 The PMCs are incremented (if permitted by other MMCR bits). 1 The PMCs are not incremented if MSR[PMM] = 1.
4	FCM0	Freeze counters while mark = 0 0 The PMCs are incremented (if permitted by other MMCR bits). 1 The PMCs are not incremented if MSR[PMM] = 0.
5	PMXE	Performance monitor exception enable 0 Performance monitor exceptions are disabled. 1 Performance monitor exceptions are enabled until a performance monitor exception occurs, at which time MMCR0[PMXE] is cleared. Software can clear PMXE to prevent performance monitor exceptions. Software can also set PMXE and then poll it to determine whether an enabled condition or event occurred.
6	FCECE	Freeze counters on enabled condition or event 0 The PMCs are incremented (if permitted by other MMCR bits). 1 The PMCs are incremented (if permitted by other MMCR bits) until an enabled condition or event occurs when MMCR0[TRIGGER] = 0, at which time MMCR0[FC] is set. If the enabled condition or event occurs when MMCR0[TRIGGER] = 1, FCECE is treated as if it were 0. The use of the trigger and freeze counter conditions depends on the enabled conditions and events described in Section 11.2, “Performance Monitor Exception.”

Table 2-40. MMCR0 Field Descriptions (continued)

Bits	Name	Description
7–8	TBSEL	Time base selector. Selects the time base bit that can cause a time base transition event (the event occurs when the selected bit changes from 0 to 1). 00 TBL[31] 01 TBL[23] 10 TBL[19] 11 TBL[15] Time base transition events can be used to periodically collect information about processor activity. In multiprocessor systems in which the TB registers are synchronized among processors, time base transition events can be used to correlate the performance monitor data obtained by the several processors. For this use, software must specify the same TBSEL value for all the processors in the system. Because the time-base frequency is implementation-dependent, software should invoke a system service program to obtain the frequency before choosing a value for TBSEL.
9	TBEE	Time base event enable 0 Time-base transition events are disabled. 1 Time-base transition events are enabled. A time-base transition is signaled to the performance monitor if the TB bit specified in MMCR0[TBSEL] changes from 0 to 1. Time-base transition events can be used to freeze the counters (MMCR0[FCECE]), trigger the counters (MMCR0[TRIGGER]), or signal an exception (MMCR0[PMXE]). Changing the bits specified in MMCR0[TBSEL] while MMCR0[TBEE] is enabled may cause a false 0 to 1 transition that signals the specified action (freeze, trigger, or exception) to occur immediately.
10–15	THRESHOLD	Threshold. Contains a threshold value between 0 to 63. Two types of thresholds can be counted. The first type counts any event that lasts longer than the threshold value and uses MMCR2[THRESHMULT] to scale the threshold value by 2 or 32. The second type counts only the events that exceed the threshold value. This type does not use MMCR2[THRESHMULT] to scale the threshold value. By varying the threshold value, software can obtain a profile of the characteristics of the events subject to the threshold. For example, if PMC1 counts cache misses for which the duration exceeds the threshold value, software can obtain the distribution of cache miss durations for a given program by monitoring the program repeatedly using a different threshold value each time.
16	PMC1CE	PMC1 condition enable. Controls whether counter negative conditions due to a negative value in PMC1 are enabled. 0 Counter negative conditions for PMC1 are disabled. 1 Counter negative conditions for PMC1 are enabled. These events can be used to freeze the counters (MMCR0[FCECE]), trigger the counters (MMCR0[TRIGGER]), or signal an exception (MMCR0[PMXE]).
17	PMC*n*CE	PMC*n* condition enable. Controls whether counter negative conditions due to a negative value in any PMC*n* (that is, in any PMC except PMC1) are enabled. 0 Counter negative conditions for all PMC*n*s are disabled. 1 Counter negative conditions for all PMC*n*s are enabled. These events can be used to freeze the counters (MMCR0[FCECE]), trigger the counters (MMCR0[TRIGGER]), or signal an exception (MMCR0[PMXE]).

Table 2-40. MMCR0 Field Descriptions (continued)

Bits	Name	Description
18	TRIGGER	Trigger 0 The PMCs are incremented (if permitted by other MMCR bits). 1 PMC1 is incremented (if permitted by other MMCR bits). The PMC*n*s are not incremented until PMC1 is negative or an enabled timebase or event occurs, at which time the PMC*n*s resume incrementing (if permitted by other MMCR bits) and MMCR0[TRIGGER] is cleared. The description of FCECE explains the interaction between TRIGGER and FCECE. Uses of TRIGGER include the following: • Resume counting in the PMC*n*s when PMC1 becomes negative without causing a performance monitor exception. Then freeze all PMCs (and optionally cause a performance monitor exception) when a PMC*n* becomes negative. The PMC*n*s then reflect the events that occurred after PMC1 became negative and before PMC*n* becomes negative. This use requires the following MMCR0 bit settings. – TRIGGER = 1 – PMC1CE = 0 – PMC*n*CE = 1 – TBEE = 0 – FCECE = 1 – PMXE = 1 (if a performance monitor exception is desired) • Resume counting in the PMC*n*s when PMC1 becomes negative, and cause a performance monitor exception without freezing any PMCs. The PMC*n*s then reflect the events that occurred between the time PMC1 became negative and the time the interrupt handler reads them. This use requires the following MMCR0 bit settings. – TRIGGER = 1 – PMC1CE = 1 – TBEE = 0 – FCECE = 0 – PMXE = 1 The use of the trigger and freeze counter conditions depends on the enabled conditions and events described in Section 11.2, “Performance Monitor Exception.”
19–25	PMC1SEL	PMC1 selector. Contains a code (one of at most 128 values) that identifies the event to be counted in PMC1. See Table 11-9.
26–31	PMC2SEL	PMC2 selector. Contains a code (one of at most 64 values) that identifies the event to be counted in PMC2. See Table 11-10.

MMCR0 can be accessed with **mtspr** and **mfspr** using SPR 952.

2.2.5.9.2 User Monitor Mode Control Register 0 (UMMCR0)

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

2.2.5.9.3 Monitor Mode Control Register 1 (MMCR1)

The monitor mode control register 1 (MMCR1) functions as an event selector for performance monitor counter registers 3, 4, 5, and 6 (PMC3, PMC4, PMC5, PMC6). The MMCR1 register is shown in Figure 2-46.

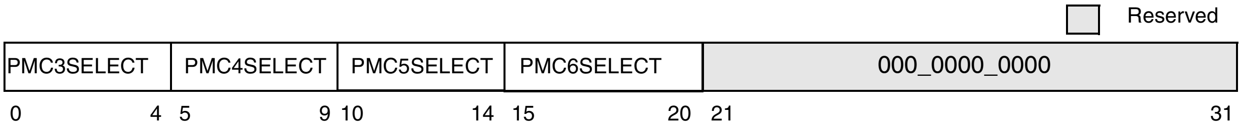


Figure 2-46. Monitor Mode Control Register 1 (MMCR1)

Bit settings for MMCR1 are shown in Table 2-41. The corresponding events are described in Section 2.2.5.9.8, “Performance Monitor Counter Registers (PMC1–PMC6).”

Table 2-41. MMCR1 Field Descriptions

Bits	Name	Description
0–4	PMC3SELECT	PMC3 selector. Contains a code (1 of at most 32 values) that identifies the event to be counted in PMC3. See Table 11-11.
5–9	PMC4SELECT	PMC4 selector. Contains a code (1 of at most 32 values) that identifies the event to be counted in PMC4. See Table 11-12.
10–14	PMC5SELECT	PMC5 selector. Contains a code (1 of at most 32 values) that identifies the event to be counted in PMC5. See Table 11-13.
15–20	PMC6SELECT	PMC6 selector. Contains a code (1 of at most 64 values) that identifies the event to be counted in PMC6. See Table 11-14.
21–31	—	Reserved

MMCR1 can be accessed with **mtspr** and **mfspr** using SPR 956. User-level software can read the contents of MMCR1 by issuing an **mfspr** instruction to UMMCR1, described in Section 2.2.5.9.4, “User Monitor Mode Control Register 1 (UMMCR1).”

2.2.5.9.4 User Monitor Mode Control Register 1 (UMMCR1)

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

2.2.5.9.5 Monitor Mode Control Register 2 (MMCR2)

The monitor mode control register 2 (MMCR2) functions as an event selector for performance monitor counter registers 3 and 4 (PMC3 and PMC4). The MMCR2 register is shown in Figure 2-47.

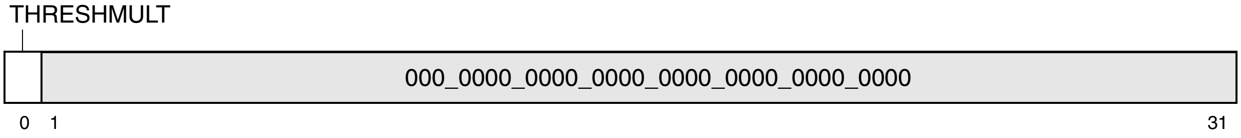


Figure 2-47. Monitor Mode Control Register 2 (MMCR2)

Table 2-42 describes MMCR2 fields.

Table 2-42. MMCR2 Field Descriptions

Bits	Name	Description
0	THRESHMULT	Threshold multiplier. Used to extend the range of the THRESHOLD field, MMCR0[10–15]. 0 Threshold field is multiplied by 2 1 Threshold field is multiplied by 32
1–31	—	Reserved

MMCR2 can be accessed with **mtspr** and **mfspr** using SPR 944. User-level software can read the contents of MMCR2 by issuing an **mfspr** instruction to UMMCR2, described in Section 2.2.5.9.6, “User Monitor Mode Control Register 2 (UMMCR2).”

2.2.5.9.6 User Monitor Mode Control Register 2 (UMMCR2)

The contents of MMCR2 are reflected to UMMCR2, which can be read by user-level software. UMMCR2 can be accessed with the **mfspr** instruction using SPR 928.

2.2.5.9.7 Breakpoint Address Mask Register (BAMR)

The breakpoint address mask register (BAMR), shown in Figure 2-48, is used in conjunction with the events that monitor IABR hits.

Figure 2-48. Breakpoint Address Mask Register (BAMR)

Table 2-43 describes BAMR fields.

Table 2-43. BAMR Field Descriptions

Bit	Name	Description
0–29	MASK[1]	Used with PMC1 event (PMC1 event 42) that monitor IABR hits. The addresses to be compared for an IABR match are affected by the value in BAMR: • IABR hit (PMC1, event 42) occurs if IABR_CMP (that is, IABR AND BAMR) = instruction_address_compare (that is, EA AND BAMR) IABR_CMP[0–29] = IABR[0–29] AND BAMR[0–29] instruction_addr_cmp[0–29] = instruction_addr[0–29] AND BAMR[0–29] Be aware that breakpoint event 42 of PMC1 can be used to trigger performance monitor exceptions when the performance monitor detects an enabled overflow. This feature supports debug purposes and occurs only when IABR[30] is set. To avoid taking one of the above interrupts, make sure that IABR[30] is cleared.
30–31	—	Reserved

[1] A context synchronizing instruction must follow the mtspr.

BAMR can be accessed with **mtspr** and **mfspr** using SPR 951. For synchronization requirements on the register see Section 2.4.2.4, “Synchronization.”

2.2.5.9.8 Performance Monitor Counter Registers (PMC1–PMC6)

PMC1–PMC6, shown in Figure 2-49, are 32-bit counters that can be programmed to generate a performance monitor exception when they overflow.

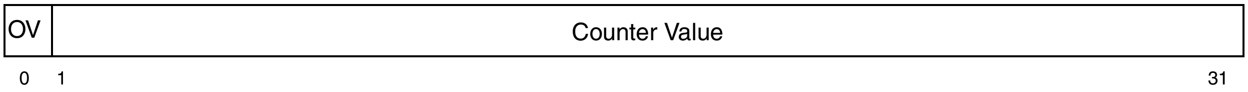


Figure 2-49. Performance Monitor Counter Registers (PMC1–PMC6)

The bits contained in the PMC registers are described in Table 2-44.

Table 2-44. PMC*n* Field Descriptions

Bits	Name	Description
0	OV	Overflow When this bit is set, it indicates that this counter has overflowed and reached its maximum value so that PMCn[OV] = 1.
1–31	Counter value	Counter value Indicates the number of occurrences of the specified event.

Counters overflow when the high-order (sign) bit becomes set; that is, they reach the value 2,147,483,648 (0x8000_0000). However, an exception is not generated unless both MMCR0[PMXE] and either MMCR0[PMC1CE] or MMCR0[PMC*c*CE] are also set as appropriate.

Note that the exception can be masked by clearing MSR[EE]; the performance monitor condition may occur with MSR[EE] cleared, but the exception is not taken until MSR[EE] is set. Setting MMCR0[FCECE] forces counters to stop counting when a counter exception or any enabled condition or event occurs. Setting MMCR0[TRIGGER] forces counters PMC*n* ($n > 1$), to begin counting when PMC1 goes negative or an enabled condition or event occurs.

Software is expected to use the **mtspr** instruction to explicitly set PMC to non-overflowed values. Setting an overflowed value may cause an erroneous exception. For example, if both MMCR0[PMXE] and either MMCR0[PMC1CE] or MMCR0[PMC*n*CE] are set and the **mtspr** instruction loads an overflow value, an exception may be taken without an event counting having taken place.

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

- PMC1 is SPR 953
- PMC2 is SPR 954
- PMC3 is SPR 957
- PMC4 is SPR 958
- PMC5 is SPR 945
- PMC6 is SPR 946

2.2.5.9.9 User Performance Monitor Counter Registers (UPMC1–UPMC6)

The contents of the PMC1–PMC6 are reflected to UPMC1–UPMC6, which can be read by user-level software. The UPMC registers can be read with **mfspr** using the following SPR numbers:

- UPMC1 is SPR 937
- UPMC2 is SPR 938
- UPMC3 is SPR 941
- UPMC4 is SPR 942
- UPMC5 is SPR 929
- UPMC6 is SPR 930

2.2.5.9.10 Sampled Instruction Address Register (SIAR)

The sampled instruction address register (SIAR) is a supervisor-level register that contains the effective address of the last instruction to complete before the performance monitor exception is signaled. The SIAR is shown in Figure 2-50.

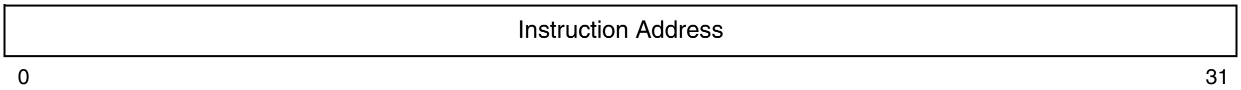


Figure 2-50. Sampled Instruction Address Registers (SIAR)

Note that SIAR is not updated in any of the following conditions:

- Performance monitor counting has been disabled by setting MMCR0[FC].
- Performance monitor exception has been disabled by clearing MMCR0[PMXE]

SIAR can be accessed with the **mtspr** and **mfspr** instructions using SPR 955.

2.2.5.9.11 User-Sampled Instruction Address Register (USIAR)

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

2.2.5.9.12 Sampled Data Address Register (SDAR) and User-Sampled Data Address Register (USDAR)

The MPC7450 does not implement the sampled data address register (SDAR) or the user-level, read-only USDA registers. Note that in previous processors the SDAR and USDAR registers could be written to by boot code without causing an exception, this is not the case in the MPC7450. A **mtspr** or **mfspr** SDAR or USDAR instruction causes a program exception.

2.2.6 Reset Settings

Table 2-45 shows the state of the registers and other resources after a hard reset and before the first instruction is fetched from address 0xFFF0_0100 (the system reset exception vector). When a register is not initialized at hard reset, the setting is undefined.

Table 2-45. Settings Caused by Hard Reset (Used at Power-On)

Resource	Setting
BAMR	0x0000_0000
BATs	Undefined
Caches (L1/L2)	Disabled. The caches are not invalidated and must be invalidated in software before they are enabled.
CR	0x0000_0000
CTR	0x0000_0000
DABR	Breakpoint is disabled. Address is undefined.
DAR	0x0000_0000
DEC	0xFFFF_FFFF
DSISR	0x0000_0000
EAR	0x0000_0000
FPRs	Undefined
FPSCR	0x0000_0000
GPRs	Undefined
HID0	0x8000_0000
HID1	0x000*n*_*n*080 (note that bits 15–19 are set to match the settings of PLL_CFG[0:4] [1]at reset)
IABR	0x0000_0000 (breakpoint is disabled)
ICTC	0x0000_0000
ICTRL	0x0000_0000
L2CAPTDATAHI	0x0000_0000
L2CAPTDATALO	0x0000_0000
L2CAPTECC	0x0000_0000
L2CR	For the MPC7441, MPC7445, MPC7447, MPC7447A, MPC7450, MPC7451, MPC7455, and MPC7457, 0x0000_0000 For the MPC7448, 0x3000_0000
L2ERRADDR	0x0000_0000
L2ERRATTR	0x0000_0000
L2ERRCTL	0x0000_0000
L2ERRDET	0x0000_0000
L2ERRDIS	0x0000_0000
L2ERREADDR	0x0000_0000
L2ERRINJCTL	0x0000_0000
L2ERRINJHI	0x0000_0000
L2ERRINJLO	0x0000_0000

Table 2-45. Settings Caused by Hard Reset (Used at Power-On) (continued)

Resource	Setting
L2ERRINTEN	0x0000_0000
L3CR	0x0000_0000
L3ITCR*n*	Undefined [2]
L3OHCR	0x0000_0000
L3PM	0x0000_0000
LDSTCR	0x0000_0000
LR	0x0000_0000
MMCR*n*	0x0000_0000
MSSCR0	0x0040_0000 0x0000_0000 (except that the ABD (bit 11) and BMODE (bits 16–17) are set depending on setting of $\overline{\text{BMODE}}$[0:1] at reset)
MSSSR0	0x0000_0000
MSR	0x0000_0040 (only IP set)
PIR	0x0000_0000
PMC*n*	Undefined
PTEHI	0x0000_0000
PTELO	0x0000_0000
PVR	For the MPC7451, 0x8000_xxxx, where xxxx depends on the revision level, starting at 0200. For the MPC7441, 0x8000_xxxx, where xxxx depends on the revision level, starting at 0200. For the MPC7455, 0x8001_xxxx, where xxxx depends on the revision level, starting at 0100. For the MPC7445, 0x8001_xxxx, where xxxx depends on the revision level, starting at 0100. For the MPC7457, 0x8002_xxxx, where xxxx depends on the revision level, starting at 0100. For the MPC7447, 0x8002_xxxx, where xxxx depends on the revision level, starting at 0100. For the MPC7447A, 0x8003_xxxx, where xxxx depends on the revision level, starting at 0100. For the MPC7448, 0x8004_xxxx, where xxxx depends on the revision level, starting at 0100.
Reservation address	Undefined
Reservation flag	Cleared
SDR1	0x0000_0000
SIAR	0x0000_0000
SPRG0–SPGR7	0x0000_0000
SRs	Undefined
SRR0	0x0000_0000
SRR1	0x0000_0000
TBU and TBL	0x0000_0000
TLBs	Undefined
TLBMISS	0x0000_0000
UMMCR*n*	0x0000_0000

Table 2-45. Settings Caused by Hard Reset (Used at Power-On) (continued)

Resource	Setting
UPMC*n*	0x0000_0000
USIAR	0x0000_0000
VRs	Undefined
VRSAVE	0x0000_0000
VSCR	0x0001_0000
XER	0x0000_0000

[1] PLL_CFG[0:5], bits 14–19 on the MPC7448.

[2] Initialized when L3 clocks are enabled.

2.3 Operand Conventions

This section describes the operand conventions as they are represented in two levels of the PowerPC architecture—UISA and VEA. Detailed descriptions are provided of conventions used for storing values in registers and memory, accessing PowerPC registers, and representation of data in these registers.

2.3.1 Floating-Point Execution Models—UISA

The IEEE 754 standard defines conventions for 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 can have single-precision operands but always produce double-precision results.
- Single-precision arithmetic instructions require all operands to be single-precision and always produce single-precision results.

For arithmetic instructions, conversion from double- to single-precision must be done explicitly by software, while conversion from single- to double-precision is done implicitly by the processor.

All implementations of the PowerPC architecture 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 has a value of 1:

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

2.3.2 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 can be bytes, half words, words, double words, quad 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.

2.3.3 Alignment and Misaligned Accesses

The operand of a single-register memory access instruction has an alignment boundary equal to its length. An operand's address is misaligned if it is not a multiple of its width.

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

Some instructions require their memory operands to have certain alignment. In addition, alignment can affect performance. For single-register memory access instructions, the best performance is obtained when memory operands are aligned.

Instructions are 32 bits (one word) long and must be word-aligned.

The MPC7450 does not provide hardware support for floating-point memory that is not word-aligned. If a floating-point operand is not word-aligned, the MPC7450 invokes an alignment exception, and it is left up to software to break up the offending memory access operation appropriately. In addition, some non-double-word–aligned memory accesses suffer performance degradation as compared to an aligned access of the same type.

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

2.3.4 Floating-Point Operands

The MPC7450 provides hardware support for all single- and double-precision floating-point operations for most value representations and all rounding modes. This architecture provides for hardware to implement a floating-point system as defined in ANSI/IEEE Standard 754-1985, *IEEE Standard for Binary Floating Point Arithmetic*. Detailed information about the floating-point execution model can be found in Chapter 3, “Operand Conventions,” in the *Programming Environments Manual*.

The MPC7450 supports non-IEEE mode when FPSCR[29] is set. In this mode, denormalized numbers are treated in a non-IEEE conforming manner. This is accomplished by delivering results that are forced to the value zero.

2.4 Instruction Set Summary

This chapter describes instructions and addressing modes defined for the MPC7450. These instructions are divided into the following functional categories:

- Integer instructions—These include arithmetic and logical instructions. For more information, see Section 2.4.4.1, “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 2.4.4.2, “Floating-Point Instructions.”

Load and store instructions—These include integer and floating-point load and store instructions. For more information, see Section 2.4.4.3, “Load and Store Instructions.”

- Flow control instructions—These include branching instructions, condition register logical instructions, trap instructions, and other instructions that affect the instruction flow. For more information, see Section 2.4.4.4, “Branch and Flow Control Instructions.”
- Processor control instructions—These instructions are used for synchronizing memory accesses and managing segment registers. For more information, see Section 2.4.4.6, “Processor Control Instructions,” Section 2.4.5.1, “Processor Control Instructions,” and Section 2.4.6.2, “Processor Control Instructions.”
- Memory synchronization instructions—These instructions are used for memory synchronizing. See Section 2.4.4.7, “Memory Synchronization Instructions,” and Section 2.4.5.2, “Memory Synchronization Instructions,” for more information.
- Memory control instructions—These instructions provide control of caches and TLBs. For more information, see Section 2.4.5.3, “Memory Control Instructions,” and Section 2.4.6.3, “Memory Control Instructions.”
- External control instructions—These include instructions for use with special input/output devices. For more information, see Section 2.4.5.4, “Optional External Control Instructions.”
- AltiVec instructions–AltiVec technology does not have optional instructions defined, so all instructions listed in the *AltiVec Technology Programming Environments Manual* are implemented for MPC7450. Instructions that are implementation specific are described in Section 2.7.2, “AltiVec Instructions with Specific Implementations for the MPC7450.”

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 for scheduling instructions most effectively, is provided in Chapter 6, “Instruction Timing.”

Integer instructions operate on word operands. Floating-point instructions operate on single-precision and double-precision floating-point operands. AltiVec instructions operate on byte, half-word, word, and quad-word operands. The PowerPC architecture uses instructions that are 4 bytes long and word-aligned. It provides for byte, half-word, and word operand loads and stores between memory and a set of 32 general-purpose registers (GPRs). It provides for word and double-word operand loads and stores between memory and a set of 32 floating-point registers (FPRs). It also provides for byte, half-word, word, and quad-word operand loads and stores between memory and a set of 32 vector registers (VRs).

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 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 mnemonics. Programs written to be portable across the various assemblers for the PowerPC architecture should not assume the existence of mnemonics not described in that document.

2.4.1 Classes of Instructions

The MPC7450 instructions belong to one of the following three classes:

- Defined
- Illegal
- Reserved

Note that while the definitions of these terms are consistent among the processors that implement the PowerPC architecture, the assignment of these classifications is not. For example, PowerPC instructions defined for 64-bit implementations are treated as illegal by 32-bit implementations such as the MPC7450.

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

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

2.4.1.1 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 can vary between implementations and between execution attempts in the same implementation.

2.4.1.2 Defined Instruction Class

Defined instructions are guaranteed to be supported in all implementations of the PowerPC architecture, except as stated in the instruction descriptions in Chapter 8, “Instruction Set,” of the *Programming Environments Manual*. The MPC7450 provides hardware support for all instructions defined for 32-bit implementations. It does not support the optional **fsqrt**, **fsqrts**, and **tlbia** instructions.

A processor invokes the illegal instruction error handler (part of the program exception) when it encounters a PowerPC instruction that has not been implemented. The instruction can be emulated in software, as required.

A defined instruction can have invalid forms. The MPC7450 provides limited support for instructions represented in an invalid form.

2.4.1.3 Illegal Instruction Class

Illegal instructions can be grouped into the following categories:

- Instructions not defined in the PowerPC architecture.The following primary opcodes are defined as illegal, but can be used in future extensions to the architecture:

  1, 5, 6, 9, 22, 56, 57, 60, 61

  Future versions of the PowerPC architecture can define any of these instructions to perform new functions.
- Instructions defined in the PowerPC architecture but not implemented in a specific implementation. For example, instructions that can be executed on 64-bit processors that implement the PowerPC architecture are considered illegal by 32-bit processors such as the MPC7450.

  The following primary opcodes are defined for 64-bit implementations only and are illegal on the MPC7450:

  2, 30, 58, 62
- All unused extended opcodes are illegal. The unused extended opcodes can be determined from information in Section A.4, “Instructions Sorted by Opcode (Binary),” and Section 2.4.1.4, “Reserved Instruction Class.” Notice that extended opcodes for instructions defined only for 64-bit implementations are illegal in 32-bit implementations, and vice versa. The following primary opcodes have unused extended opcodes:

  17, 19, 31, 59, 63 (Primary opcodes 30 and 62 are illegal for all 32-bit implementations, but as 64-bit opcodes, they have some unused extended opcodes.)
- An instruction consisting of only zeros is guaranteed to be an illegal instruction. This increases the probability that an attempt to execute data or memory that was not initialized invokes the system illegal instruction error handler (a program exception). Note that if only the primary opcode consists of all zeros, the instruction is considered a reserved instruction, as described in Section 2.4.1.4, “Reserved Instruction Class.”

The MPC7450 invokes the system illegal instruction error handler (a program exception) when it detects any instruction from this class or any instructions defined only for 64-bit implementations.

See Section 4.6.7, “Program Exception (0x00700),” for additional information about illegal and invalid instruction exceptions. Except for an instruction consisting of binary zeros, illegal instructions are available for additions to the PowerPC architecture.

2.4.1.4 Reserved Instruction Class

Reserved instructions are allocated to specific implementation-dependent purposes not defined by the PowerPC architecture. Attempting to execute a reserved instruction that has not been implemented invokes the illegal instruction error handler (a program exception). See “Program Exception (0x0_0700),” in Chapter 6, “Exceptions,” in the *Programming Environments Manual* for information about illegal and invalid instruction exceptions.

The PowerPC architecture defines four types of reserved instructions:

- Instructions in the POWER architecture not part of the PowerPC UISA. For details on POWER architecture incompatibilities and how they are handled by processors that implement the PowerPC architecture, see Appendix B, “POWER Architecture Cross Reference,” in the *Programming Environments Manual*.
- Implementation-specific instructions required for the processor to conform to the PowerPC architecture (none of these are implemented in the MPC7450)
- All other implementation-specific instructions
- Architecturally allowed extended opcodes

2.4.2 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*.

2.4.2.1 Memory Addressing

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

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

2.4.2.2 Memory Operands

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

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

2.4.2.3 Effective Address Calculation

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

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

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

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

Refer to Section 2.4.4.3.2, “Integer Load and Store Address Generation,” for a detailed description of effective address generation for load and store operations.

Branch instructions have three categories of effective address generation:

- Immediate
- Link register indirect
- Count register indirect

2.4.2.4 Synchronization

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

2.4.2.4.1 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.

Modifying certain registers requires software synchronization to follow certain register dependencies. Table 2-46 defines specific synchronization procedures that are required when using various SPRs and specific bits within SPRs. Context synchronizing instructions that can be used are: **isync**, **sc**, **rfi**, and any exception other than system reset and machine check. If multiple bits are being modified that have different synchronization requirements, the most restrictive requirements can be used. However, a **mtspr** instruction to modify either HID0[ICE] or HID0[ICFI] should not also modify other HID0 bits that requires synchronization.

Table 2-46. Control Registers Synchronization Requirements

Register	Bits	Synchronization Requirements
BAMR	Any	A context synchronizing instruction must follow the **mtspr**.
DABR	Any	A **dssall** and **sync** must precede the **mtspr** and then a **sync** and a context synchronizing instruction must follow. Note that if a user is not using the AltiVec data streaming instructions, then a **dssall** is not necessary prior to accessing the register.
DBATs	Any	A **dssall** and **sync** must precede the **mtspr** and then a **sync** and a context synchronizing instruction must follow. Note that if a user is not using the AltiVec data streaming instructions, then a **dssall** is not necessary prior to accessing the register.
EAR	Any	A **dssall** and **sync** must precede the **mtspr** and then a **sync** and a context synchronizing instruction must follow. Note that if a user is not using the AltiVec data streaming instructions, then a **dssall** is not necessary prior to accessing register.
HID0	BHTCLR	A context synchronizing instruction must precede a **mtspr** and a branch instruction should follow. The branch instruction may be either conditional or unconditional. It ensures that all subsequent branch instructions see the newly initialized BHT values. For correct results, the BHT should be disabled (HID0[BHT] = 0) before setting BHTCLR.
	BHT	A context synchronizing instruction must follow the **mtspr**.
	BTIC	
	DPM	
	FOLD	
	LRSTK	
	NAP	
	NHR	
	SLEEP	
	SPD	
	TBEN	

Table 2-46. Control Registers Synchronization Requirements (continued)

Register	Bits	Synchronization Requirements
HID0 (continued)	DCE	A **dssall** and **sync** must precede a **mtspr** and then a **sync** and context synchronizing instruction must follow. Note that if a user is not using the AltiVec data streaming instructions, then a **dssall** is not necessary prior to accessing the HID0{DCE] or HID0[DCFI] bit.
	DCFI	
	DLOCK	
	NOPDST	
	STEN	
	ICE	A context synchronizing instruction must immediately follow a **mtspr**. A **mtspr** instruction for HID0 should not modify either of these bits at the same time it modifies another bit that requires additional synchronization.
	ICFI	
	ILOCK	A context synchronizing instruction must precede and follow a **mtspr**.
	NOPTI	A **mtspr** must follow a **sync** and a context synchronizing instruction.
	SGE	
	XAEN	**A dssall** and **sync** must precede a **mtspr** and then a **sync** and a context-synchronizing instruction must follow. Alteration of HID0[XAEN] must be done with caches and translation disabled. The caches and TLBs must be flushed before they are re-enabled after the XAEN bit is altered. Note that if a user is not using the AltiVec data streaming instructions, then a **dssall** is not necessary prior to accessing the HID0[XAEN] bit.
HID1	Any	A **sync** and context synchronizing instruction must follow a **mtspr**.
IABR	Any	A context synchronizing instruction must follow a **mtspr**.
IBATs	Any	A context synchronizing instruction must follow a **mtspr**.
ICTRL	EDCE	A **dssall** and **sync** must precede a **mtspr** and then a **sync** and context synchronizing instruction must follow. Note that if a user is not using the AltiVec data streaming instructions, then a **dssall** is not necessary prior to accessing the ICTRL[EDCE] bit.
	ICWL	A context synchronizing instruction must precede and follow a **mtspr**.
	EICE	
L2ERRDIS	Any	A **sync** must precede a **mtspr** and then a **sync** and **isync** must follow.
LDSTCR	Any	A **dssall** and **sync** must precede a **mtspr** and then a **sync** and context synchronizing instruction must follow. Note that if a user is not using the AltiVec data streaming instructions, then a **dssall** is not necessary prior to accessing the register.

Table 2-46. Control Registers Synchronization Requirements (continued)

Register	Bits	Synchronization Requirements
MSR	BE	A context synchronizing instruction must follow a **mtmsr** instruction.
	VEC	
	FE0	
	FE1	
	FP	
	SE	
	IR	A context synchronizing instruction must follow a **mtmsr**. When changing the MSR[IR] bit the context synchronizing instruction must reside at both the untranslated and the translated address following the **mtmsr**.
	DR	A **dssall** and **sync** must precede a **mtmsr** and then a **sync** and context synchronizing instruction must follow. Note that if a user is not using the AltiVec data streaming instructions, then a **dssall** is not necessary prior to accessing the MSR[DR] or MSR[PR] bit.
	PR	
	LE	**A dssall** and **sync** must precede an **rfi** to guarantee a solid context boundary. Note that if a user is not using the AltiVec data streaming instructions, then a **dssall** is not necessary prior to accessing the MSR[LE] bit.
	POW	A **dssall** and **sync** must precede a **mtmsr** instruction and then a context synchronizing instruction must follow.
MSSCR0	Any	A **dssall** and **sync** must precede a **mtspr** instruction and then a **sync** and context synchronizing instruction must follow. Note that if a user is not using the AltiVec data streaming instructions, then a **dssall** is not necessary prior to accessing the register.
SDR1	Any	A **dssall** and **sync** must precede a **mtspr** and then a **sync** and context synchronizing instruction must follow. Note that if a user is not using the AltiVec data streaming instructions, then a **dssall** is not necessary prior to accessing the register.
L3PM	Any	A **sync** must precede a **mtsp**r instruction and then a **sync** and context synchronizing instruction must follow. Note that if a user is not using the AltiVec data streaming instructions, then a **dssall** is not necessary prior to accessing the register.
SR0 – SR15	Any	A **dssall** and **sync** must precede a **mtsr** or **mtsrin** instruction and then a **sync** and context synchronizing instruction must follow. Note that if a user is not using the AltiVec data streaming instructions, then a **dssall** is not necessary prior to accessing the register.
Other registers or bits	—	No special synchronization requirements.

2.4.2.4.2 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 **sync** and **isync**, before the instruction completes. For example, the Move to Machine State Register (**mtmsr**) instruction is execution synchronizing. It ensures that all preceding instructions have completed execution and cannot cause an exception before the instruction executes, but does not ensure 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.

2.4.2.4.3 Instruction-Related Exceptions

There are two kinds of exceptions in the MPC7450—those caused directly by the execution of an instruction and those caused by an asynchronous event (or interrupts). Either can cause components of the system software to be invoked.

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

- An attempt to execute an illegal instruction causes the illegal instruction (program exception) handler to be invoked. An attempt by a user-level program to execute the supervisor-level instructions listed below causes the privileged instruction (program exception) handler to be invoked. The MPC7450 provides the following supervisor-level instructions—**dcbi**, **mfmsr**, **mfspr**, **mfsr**, **mfsrin**, **mtmsr**, **mtspr**, **mtsr**, **mtsrin**, **rfi**, **tlbie**, and **tlbsync**. Note that the privilege level of the **mfspr** and **mtspr** instructions depends on the SPR encoding.
- Any **mtspr**, **mfspr**, or **mftb** instruction with an invalid SPR (or TBR) field causes an illegal type program exception. Likewise, a program exception is taken if user-level software tries to access a supervisor-level SPR. An **mtspr** instruction executing in supervisor mode (MSR[PR] = 0) with the SPR field specifying PVR (read-only register) executes as a no-op.
- An attempt to access memory that is not available (page fault) causes the ISI or DSI exception handler to be invoked.
- The execution of an **sc** instruction invokes the system call exception handler that permits a program to request the system to perform a service.
- The execution of a trap instruction invokes the program exception trap handler.
- The execution of an instruction that causes a floating-point exception while exceptions are enabled in the MSR invokes the program exception handler.

A detailed description of exception conditions is provided in Chapter 4, “Exceptions.”

2.4.3 Instruction Set Overview

This section provides a brief overview of the PowerPC instructions implemented in the MPC7450 and highlights any special information with respect to how the MPC7450 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, are provided for the convenience of the programmer, and do not necessarily reflect the PowerPC architecture specification.

Note that some instructions have the following optional features:

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

2.4.4 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.

2.4.4.1 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, the XER register, and condition register (CR) fields.

2.4.4.1.1 Integer Arithmetic Instructions

Table 2-47 lists the integer arithmetic instructions for the processors that implement the PowerPC architecture.

Table 2-47. Integer Arithmetic Instructions

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

Table 2-47. Integer Arithmetic Instructions (continued)

Name	Mnemonic	Syntax
Multiply Low Word	**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.

The UISA states that an implementation that executes instructions that set the overflow enable bit (OE) or the carry bit (CA) can either execute these instructions slowly or prevent execution of the subsequent instruction until the operation completes. Chapter 6, "Instruction Timing," describes how the MPC7450 handles CR dependencies. The summary overflow bit (SO) and overflow bit (OV) in the XER register are set to reflect an overflow condition of a 32-bit result. This can happen only when OE = 1.

2.4.4.1.2 Integer Compare Instructions

The integer compare instructions algebraically or logically compare the contents of register **r**A with either the zero-extended value of the UIMM operand, the sign-extended value of 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 2-48 summarizes the integer compare instructions.

Table 2-48. Integer Compare Instructions

Name	Mnemonic	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 **crf**D, using an explicit field number.

For information on simplified mnemonics for the integer compare instructions see Appendix F, "Simplified Mnemonics," in the *Programming Environments Manual*.

2.4.4.1.3 Integer Logical Instructions

The logical instructions shown in Table 2-49 perform bit-parallel operations on the specified operands. Logical instructions with the CR updating enabled (uses dot suffix) and instructions **andi.** and **andis.** set

CR field CR0 to characterize the result of the logical operation. Logical instructions do not affect XER[SO], XER[OV], or XER[CA].

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

Table 2-49. Integer Logical Instructions

Name	Mnemonic	Syntax	Implementation Notes
AND Immediate	**andi.**	**r**A,**r**S,UIMM	—
AND Immediate Shifted	**andis.**	**r**A,**r**S,UIMM	—
OR Immediate	**ori**	**r**A,**r**S,UIMM	The PowerPC architecture defines **ori r0,r0,0** as the preferred form for the no-op instruction. The dispatcher discards this instruction and only dispatches it to the completion queue, but not to any execution unit.
OR Immediate Shifted	**oris**	**r**A,**r**S,UIMM	—
XOR Immediate	**xori**	**r**A,**r**S,UIMM	—
XOR Immediate Shifted	**xoris**	**r**A,**r**S,UIMM	—
AND	**and** (**and**.)	**r**A,**r**S,**r**B	—
OR	**or (or.)**	**r**A,**r**S,**r**B	—
XOR	**xor (xor.)**	**r**A,**r**S,**r**B	—
NAND	**nand (nand.)**	**r**A,**r**S,**r**B	—
NOR	**nor (nor.)**	**r**A,**r**S,**r**B	—
Equivalent	**eqv (eqv.)**	**r**A,**r**S,**r**B	—
AND with Complement	**andc (andc.)**	**r**A,**r**S,**r**B	—
OR with Complement	**orc (orc.)**	**r**A,**r**S,**r**B	—
Extend Sign Byte	**extsb (extsb.)**	**r**A,**r**S	—
Extend Sign Half Word	**extsh (extsh.)**	**r**A,**r**S	—
Count Leading Zeros Word	**cntlzw (cntlzw.)**	**r**A,**r**S	—

2.4.4.1.4 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 summarized in Table 2-50.

Table 2-50. Integer Rotate Instructions

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

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

Multiple-precision shifts can be programmed as shown in Appendix C, “Multiple-Precision Shifts,” in the *Programming Environments Manual*. The integer shift instructions are summarized in Table 2-51.

Table 2-51. Integer Shift Instructions

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

2.4.4.2 Floating-Point Instructions

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

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

See Section 2.4.4.3, “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 (FPSCR[NI]).

2.4.4.2.1 Floating-Point Arithmetic Instructions

The floating-point arithmetic instructions are summarized in Table 2-52.

Table 2-52. Floating-Point Arithmetic Instructions

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

[1] These instructions are optional in the PowerPC architecture.

All single-precision arithmetic instructions are performed using a double-precision format. The floating-point architecture is a single-pass implementation for double-precision products. In most cases, a single-precision instruction using only single-precision operands, in double-precision format, has the same latency as its double-precision equivalent.

2.4.4.2.2 Floating-Point Multiply-Add Instructions

These instructions combine multiply and add operations without an intermediate rounding operation. The floating-point multiply-add instructions are summarized in Table 2-53.

Table 2-53. Floating-Point Multiply-Add Instructions

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

2.4.4.2.3 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 convert instructions convert a 64-bit double-precision floating-point number to a 32-bit signed integer number.

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

Table 2-54. Floating-Point Rounding and Conversion Instructions

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

2.4.4.2.4 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 summarized in Table 2-55.

Table 2-55. Floating-Point Compare Instructions

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

2.4.4.2.5 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 summarized in Table 2-56.

Table 2-56. Floating-Point Status and Control Register Instructions

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

Implementation Note—The PowerPC architecture states that in some implementations, the Move to FPSCR Fields (**mtfsf**) instruction can perform more slowly when only some of the fields are updated as opposed to all of the fields. In the MPC7450, there is no degradation of performance.

2.4.4.2.6 Floating-Point Move Instructions

Floating-point move instructions copy data from one FPR to another. The floating-point move instructions do not modify the FPSCR. The CR update option in these instructions controls the placing of result status into CR1. Table 2-57 summarizes the floating-point move instructions.

Table 2-57. Floating-Point Move Instructions

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

2.4.4.3 Load and Store Instructions

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

- Integer load instructions
- Integer store instructions
- Integer load and store with byte-reverse instructions
- Integer load and store multiple instructions
- Floating-point load instructions
- Floating-point store instructions
- Memory synchronization instructions

Implementation Note—The following describes how the MPC7450 handles misalignment:

The MPC7450 provides hardware support for misaligned memory accesses. It performs those accesses within a single cycle if the operand lies within a double-word boundary. Misaligned memory accesses that cross a double-word boundary degrade performance.

Although many misaligned memory accesses are supported in hardware, the frequent use of them is discouraged because they can compromise the overall performance of the processor. Only one outstanding misalignment at a time is supported which means it is non-pipelined.

Accesses that cross a translation boundary can be restarted. That is, a misaligned access that crosses a page boundary is completely restarted if the second portion of the access causes a page fault. This can cause the first access to be repeated.

On some processors, such as the MPC603e, a TLB reload operation causes an instruction restart. On the MPC7450, TLB reloads are performed transparently (if hardware table search operations are enabled—HID0[STEN] = 0) and only a page fault causes a restart. If software table searching is enabled (HID0[STEN] = 1) on the MPC7450, a TLB miss causes an instruction restart (as it causes a TLB miss exception)

2.4.4.3.1 Self-Modifying Code

When a processor modifies a memory location that can be contained in the instruction cache, software must ensure that memory updates are visible to the instruction fetching mechanism. This can be achieved by executing the following instruction sequence (using either **dcbst** or **dcbf**):

```
dcbst (or dcbf)|update memory
sync           |wait for update
icbi           |remove (invalidate) copy in instruction cache
sync           |ensure that ICBI invalidate at the icache has completed
isync          |remove copy in own instruction buffer
```

These operations are required because the data cache is a write-back cache. Because instruction fetching bypasses the data cache, changes to items in the data cache can not be reflected in memory until the fetch operations complete. The **sync** after the **icbi** is required to ensure that the **icbi** invalidation has completed in the instruction cache.

Special care must be taken to avoid coherency paradoxes in systems that implement unified secondary caches (like the MPC7450), 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*.

2.4.4.3.2 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 2.4.2.3, “Effective Address Calculation,” for information about calculating effective addresses. Note that in some implementations, operations that are not naturally aligned can suffer performance degradation. Refer to Section 4.6.6, “Alignment Exception (0x00600),” for additional information about load and store address alignment exceptions.

2.4.4.3.3 Register Indirect Integer Load Instructions

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

Implementation Notes—The following notes describe the MPC7450 implementation of integer load instructions:

- The PowerPC architecture cautions programmers that some implementations of the architecture can execute the load half algebraic (**lha**, **lhax**) instructions with greater latency than other types of load instructions. This is not the case for the MPC7450; these instructions operate with the same latency as other load instructions.

- The PowerPC architecture cautions programmers that some implementations of the architecture can run the load/store byte-reverse (**lhbrx**, **lbrx**, **sthbrx**, **stwbrx**) instructions with greater latency than other types of load/store instructions. This is not the case for the MPC7450. These instructions operate with the same latency as the other load/store instructions.
- The PowerPC architecture describes some preferred instruction forms for load and store multiple instructions and integer move assist instructions that can perform better than other forms in some implementations. None of these preferred forms affect instruction performance on the MPC7450. Usage of load/store string instruction is discouraged.
- The PowerPC architecture defines the **lwarx** and **stwcx.** as a way to update memory atomically. In the MPC7450, reservations are made on behalf of aligned 32-byte sections of the memory address space. Executing **lwarx** and **stwcx.** to a page marked write-through does cause a DSI exception if the page is marked cacheable write-through (WIM = 10x) or caching-inhibited (WIM = x1x), but as with other memory accesses, DSI exceptions can result for other reasons such as a protection violations or page faults.

Table 2-58 summarizes the integer load instructions.

Table 2-58. Integer Load Instructions

Name	Mnemonic	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

2.4.4.3.4 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 EA (effective address). Many store instructions have an update form, in which **r**A is updated with the EA. For these forms, the following rules apply:

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

The PowerPC architecture defines store with update instructions with **r**A = 0 as an invalid form. In addition, it defines integer store instructions with the CR update option enabled (Rc field, bit 31, in the instruction encoding = 1) to be an invalid form. Table 2-59 summarizes the integer store instructions.

Table 2-59. Integer Store Instructions

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

2.4.4.3.5 Integer Store Gathering

The MPC7450 performs store gathering for write-through accesses to nonguarded space or to cache-inhibited stores to nonguarded space if the requirements described in Section 3.1.2.3, “Store Gathering/Merging,” are met. These stores are combined in the load/store unit (LSU) to form a double word or quad word and are sent out on the system bus as a single operation. However, stores can be gathered only if the successive stores that meet the criteria are queued and pending. The MPC7450 also performs store merging as described in Section 3.1.2.3, “Store Gathering/Merging.”

Store gathering takes place regardless of the address order of the stores. The store gathering and merging feature is enabled by setting HID0[SGE].

If store gathering is enabled and the stores do not fall under the above categories, an **eieio** or **sync** instruction must be used to prevent two stores from being gathered.

2.4.4.3.6 Integer Load and Store with Byte-Reverse Instructions

Table 2-60 describes integer load and store with byte-reverse instructions. When used in a 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 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*.

Table 2-60. Integer Load and Store with Byte-Reverse Instructions

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

2.4.4.3.7 Integer Load and Store Multiple Instructions

The load/store multiple instructions are used to move blocks of data to and from the GPRs. The load multiple and store multiple instructions can have operands that require memory accesses crossing a 4-Kbyte page boundary. As a result, these instructions can be interrupted by a DSI exception associated with the address translation of the second page. Table 2-61 summarizes the integer load and store multiple instructions.

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

Table 2-61. Integer Load and Store Multiple Instructions

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

2.4.4.3.8 Integer Load and Store String Instructions

The integer load and store string instructions allow movement of data from memory to registers or from registers to memory without concern for alignment. These instructions can be used for a short move between arbitrary memory locations or to initiate a long move between misaligned memory fields. However, in some implementations, these instructions are likely to have greater latency and take longer to execute, perhaps much longer, than a sequence of individual load or store instructions that produce the same results. Table 2-62 summarizes the integer load and store string instructions.

Table 2-62. Integer Load and Store String Instructions

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

In the MPC7450 implementation operating with little-endian byte order, execution of a load or string instruction will take an alignment exception.

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

For load/store string operations, the MPC7450 does not combine register values to reduce the number of discrete accesses. However, if store gathering is enabled and the accesses fall under the criteria for store gathering the stores can be combined to enhance performance. At a minimum, additional cache access cycles are required. Usage of load/store string instructions is discouraged.

2.4.4.3.9 Floating-Point Load and Store Address Generation

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

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 an operand into an FPR.

Implementation Note—The MPC7450 treats exceptions as follows:

- The FPU can be run in two different modes—Ignore exceptions mode (MSR[FE0] = MSR[FE1] = 0) and precise mode (any other settings for MSR[FE0,FE1]). For the MPC7450, ignore exceptions mode allows floating-point instructions to complete earlier and thus can provide better performance than precise mode.

The floating-point load and store indexed instructions (**lfsx**, **lfsux**, **lfdx**, **lfdux**, **stfsx**, **stfsux**, **stfdx**, **stfdux**) are invalid when the Rc bit is one. The PowerPC architecture defines a load with update instruction with **r**A = 0 as an invalid form. Table 2-63 summarizes the floating-point load instructions.

Table 2-63. Floating-Point Load Instructions

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

Table 2-63. Floating-Point Load Instructions (continued)

Name	Mnemonic	Syntax
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

2.4.4.3.10 Floating-Point Store Instructions

This section describes floating-point store instructions. There are three basic forms of the store instruction—single-precision, double-precision, and integer. The integer form is supported by the optional **stfiwx** instruction. 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. Table 2-64 summarizes the floating-point store instructions.

Table 2-64. Floating-Point Store Instructions

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

[1] The **stfiwx** instruction is optional to the PowerPC architecture

Some floating-point store instructions require conversions in the LSU. Table 2-65 shows conversions the LSU makes when executing a Store Floating-Point Single instruction.

Table 2-65. Store Floating-Point Single Behavior

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

Table 2-65. Store Floating-Point Single Behavior (continued)

FPR Precision	Data Type	Action

Double	Normalized	If (exp ≤ 896)
		then
		Denormalize and Store
		else
		Store

Double	Denormalized	Store zero
Double	Zero, infinity, QNaN	Store
Double	SNaN	Store

Table 2-66 shows the conversions made when performing a Store Floating-Point Double instruction. Most entries in the table indicate that the floating-point value is simply stored. Only in a few cases are any other actions taken.

Table 2-66. Store Floating-Point Double Behavior

FPR Precision	Data Type	Action
Single	Normalized	Store
Single	Denormalized	Normalize and Store
Single	Zero, infinity, QNaN	Store
Single	SNaN	Store
Double	Normalized	Store
Double	Denormalized	Store
Double	Zero, infinity, QNaN	Store
Double	SNaN	Store

Architecturally, all floating-point numbers are represented in double-precision format within the MPC7450. Execution of a store floating-point single (**stfs**, **stfsu**, **stfsx**, **stfsux**) instruction requires conversion from double- to single-precision format. If the exponent is not greater than 896, this conversion requires denormalization. The MPC7450 supports this denormalization by shifting the mantissa one bit at a time. Anywhere from 1 to 23 clock cycles are required to complete the denormalization, depending upon the value to be stored.

Because of how floating-point numbers are implemented in the MPC7450, there is also a case when execution of a store floating-point double (**stfd**, **stfdu**, **stfdx**, **stfdux**) instruction can require internal shifting of the mantissa. This case occurs when the operand of a store floating-point double instruction is a denormalized single-precision value. The value could be the result of a load floating-point single instruction, a single-precision arithmetic instruction, or a floating round to single-precision instruction. In these cases, shifting the mantissa takes from 1 to 23 clock cycles, depending upon the value to be stored. These cycles are incurred during the store.

2.4.4.4 Branch and Flow Control Instructions

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

2.4.4.4.1 Branch Instruction Address Calculation

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

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

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

Note that in the MPC7450, all branch instructions (**b**, **ba**, **bl**, **bla**, **bc**, **bca**, **bcl**, **bcla**, **bclr**, **bclrl**, **bcctr**, **bcctrl**) are executed in the BPU and condition register logical instructions (**crand**, **cror**, **crxor**, **crnand**, **crnor**, **crandc**, **creqv**, **crorc**, and **mcrf**) are executed by the IU2. Some of these instructions can redirect instruction execution conditionally on the value of CR, CTR, or LR bits. When the CR bits resolve, the branch instruction is either marked as correct or mispredicted. Correcting a mispredicted branch requires that the MPC7450 flush speculatively executed instructions and restore the machine state to immediately after the branch. This correction can be done when all non-speculative instructions older than the mispredicting branch have completed.

2.4.4.4.2 Branch Instructions

Table 2-67 lists the branch instructions provided by the processors that implement the PowerPC architecture. 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 2-67. Branch Instructions

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

2.4.4.4.3 Condition Register Logical Instructions

Condition register logical instructions, shown in Table 2-68, and the Move Condition Register Field (**mcrf**) instruction are also defined as flow control instructions.

Table 2-68. Condition Register Logical Instructions

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

Note that if the LR update option is enabled for any of these instructions, the PowerPC architecture defines these forms of the instructions as invalid.

2.4.4.4.4 Trap Instructions

The trap instructions shown in Table 2-69 are provided to test for a specified set of conditions. If any of the conditions tested by a trap instruction are met, the system trap type program exception is taken. For more information, see Section 4.6.7, “Program Exception (0x00700).” If the tested conditions are not met, instruction execution continues normally.

Table 2-69. Trap Instructions

Name	Mnemonic	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.

2.4.4.5 System Linkage Instruction—UISA

The System Call (**sc**) instruction permits a program to call on the system to perform a service; see Table 2-70 and also Section 2.4.6.1, “System Linkage Instructions,” for additional information.

Table 2-70. System Linkage Instruction—UISA

Name	Mnemonic	Syntax
System Call	**sc**	—

Executing this instruction causes the system call exception handler to be evoked. For more information, see Section 4.6.10, “System Call Exception (0x00C00).”

2.4.4.6 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). See Section 2.4.5.1, “Processor Control Instructions,” for the **mftb** instruction and Section 2.4.6.2, “Processor Control Instructions,” for information about the instructions used for reading from and writing to the MSR and SPRs.

2.4.4.6.1 Move To/From Condition Register Instructions

Table 2-71 summarizes the instructions for reading from or writing to the condition register.

Table 2-71. Move To/From Condition Register Instructions

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

Implementation Note—The PowerPC architecture indicates that in some implementations the Move to Condition Register Fields (**mtcrf**) instruction can perform more slowly when only a portion of the fields are updated as opposed to all of the fields. The condition register access latency for the MPC7450 is the same in both cases, if multiple fields are affected. Note that **mtcrf** single field is handled in the IU1s and latency may be lower if a **mtcrf** multi is split into its component single field pieces by the compiler.

2.4.4.6.2 Move To/From Special-Purpose Register Instructions

Table 2-72 lists the **mtspr** and **mfspr** instructions.

Table 2-72. Move To/From Special-Purpose Register Instructions (UISA)

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

Table 2-73 lists the SPR numbers for user-level PowerPC SPR accesses.

Encodings for the MPC7450-specific user-level SPRs are listed in Table 2-74.

Table 2-73. User-Level PowerPC SPR Encodings

Register Name	SPR[1]			Access	mfspr/mtspr
	Decimal	spr[5–9]	spr[0–4]		
CTR	9	00000	01001	User (UISA)	Both
LR	8	00000	01000	User (UISA)	Both

Table 2-73. User-Level PowerPC SPR Encodings (continued)

Register Name	SPR[1]			Access	mfspr/mtspr
	Decimal	spr[5–9]	spr[0–4]		
TBL[2]	268	01000	01100	User (VEA)	**mftb**
TBU [2]	269	01000	01101	User (VEA)	**mftb**
VRSAVE[3]	256	01000	00000	User (AltiVec/UISA)	Both
XER	1	00000	00001	User (UISA)	Both

[1] 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.

[2] The TB registers are referred to as TBRs rather than SPRs and can be written to using the mtspr instruction in supervisor mode and the TBR numbers here. The TB registers can be read in user mode using either the mftb instruction and specifying TBR 268 for TBL and TBR 269 for TBU.

[3] Register defined by the AltiVec Technology.

Table 2-74. User-Level SPR Encodings for MPC7450-Defined Registers

Register Name	SPR [1]			Access	mfspr/mtspr
	Decimal	spr[5–9]	spr[0–4]		
UMMCR0	936	11101	01000	User	**mfspr**
UMMCR1	940	11101	01100	User	**mfspr**
UMMCR2	928	11101	00000	User	**mfspr**
UPMC1	937	11101	01001	User	**mfspr**
UPMC2	938	11101	01010	User	**mfspr**
UPMC3	941	11101	01101	User	**mfspr**
UPMC4	942	11101	01110	User	**mfspr**
UPMC5	929	11101	00001	User	**mfspr**
UPMC6	930	11101	00010	User	**mfspr**
USIAR	939	11101	01011	User	**mfspr**

[1] 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.

2.4.4.7 Memory Synchronization Instructions

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 Section 3.3.3.6, “Atomic Memory References,” for additional information about these instructions and about related aspects of memory synchronization. See Table 2-75 for a summary.

Table 2-75. Memory Synchronization Instructions—UISA

Name	Mnemonic	Syntax	Implementation Notes
Load Word and Reserve Indexed	**lwarx** [1]	**r**D,**r**A,**r**B	Programmers can use **lwarx** with **stwcx.** to emulate common semaphore operations such as test and set, compare and swap, exchange memory, and fetch and add. Both instructions must use the same EA. Reservation granularity is implementation-dependent. The MPC7450 makes reservations on behalf of aligned 32-byte sections of the memory address space. Executing **lwarx** and **stwcx.** to a page marked write-through (WIMG = 10xx) or caching-inhibited (WIMG = x1xx) or when the data cache is disabled or locked causes a DSI exception. If the location is not word-aligned, an alignment exception occurs. The **stwcx.** instruction is the only load/store instruction with a valid form if Rc is set. If Rc is zero, executing **stwcx.** sets CR0 to an undefined value.
Store Word Conditional Indexed	**stwcx.**[1]	**r**S,**r**A,**r**B	
Synchronize	**sync**	—	Because it delays execution of subsequent instructions until all previous instructions complete to where they cannot cause an exception, **sync** is a barrier against store gathering. Additionally, all load/store cache/bus activities initiated by prior instructions are completed. Touch load operations (**dcbt**, **dcbtst**) must complete address translation, but need not complete on the bus. The **sync** completes after a successful broadcast on the system bus. The latency of **sync** depends on the processor state when it is dispatched and on various system-level situations. Note that, frequent use of **sync** will degrade performance.

[1] Note that the MPC7450 implements the **lwarx** and **stwcx.** as defined in the PowerPC architecture version 1.10. The execution of an **lwarx** or **stwcx.** instructions to memory marked write-through or cache-inhibited will cause a DSI exception.

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

See Section 2.4.5.2, “Memory Synchronization Instructions,” for details about additional memory synchronization (**eieio**) instructions.

In the PowerPC architecture, the Rc bit must be zero for most load and store instructions. If Rc is set, the instruction form is invalid for **sync** and **lwarx** instructions. If the MPC7450 encounters one of these invalid instruction forms, it sets CR0 to an undefined value.

2.4.5 PowerPC VEA Instructions

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

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

2.4.5.1 Processor Control Instructions

In addition to the move to condition register instructions (specified by the UISA), the VEA defines the **mftb** instruction (user-level instruction) for reading the contents of the time base register; see Chapter 3, “L1, L2, and L3 Cache Operation,” for more information. Table 2-76 shows the **mftb** instruction.

Table 2-76. Move From Time Base Instruction

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

Simplified mnemonics are provided for the **mftb** instruction so it can be coded with the TBR name as part of the mnemonic rather than requiring it to be coded as an operand. See Appendix F, “Simplified Mnemonics,” in the *Programming Environments Manual* for simplified mnemonic examples and for simplified mnemonics for Move from Time Base (**mftb**) and Move from Time Base Upper (**mftbu**), which are variants of the **mftb** instruction rather than of **mfspr**. The **mftb** instruction serves as both a basic and simplified mnemonic. Assemblers recognize an **mftb** mnemonic with two operands as the basic form, and an **mftb** mnemonic with one operand as the simplified form.

Implementation Note—In the MPC7450, note the following:

- The MPC7450 allows user-mode read access to the time base counter through the use of the Move from Time Base (**mftb**) instruction. As a 32-bit implementation of the PowerPC architecture, the MPC7450 can access TBU and TBL separately only.
- The time base counter is clocked at a frequency that is one-fourth that of the bus clock. Counting is enabled by assertion of the time base enable (TBEN) input signal.

2.4.5.2 Memory Synchronization Instructions

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, “L1, L2, and L3 Cache Operation,” for more information about these instructions and about related aspects of memory synchronization.

In addition to the **sync** instruction (specified by UISA), the VEA defines the Enforce In-Order Execution of I/O (**eieio**) and Instruction Synchronize (**isync**) instructions. The number of cycles required to complete an **eieio** instruction depends on system parameters and on the processor's state when the instruction is issued. As a result, frequent use of this instruction can degrade performance. Note that the broadcast of these instructions on the bus is controlled by the HID1[SYNCBE] bit.

Table 2-77 describes the memory synchronization instructions defined by the VEA.

Table 2-77. Memory Synchronization Instructions—VEA

Name	Mnemonic	Syntax	Implementation Notes
Enforce In-Order Execution of I/O	**eieio**	—	The **eieio** instruction is dispatched to the LSU and executes after all previous cache-inhibited or write-through accesses are performed; all subsequent instructions that generate such accesses execute after **eieio**. As the **eieio** operation doesn't affect the caches, it bypasses the L2 and L3 caches and is forwarded to the bus. An EIEIO operation is broadcast on the external bus to enforce ordering in the external memory system. Because the MPC7450 does reorder noncacheable accesses, **eieio** may be needed to force ordering. However, if store gathering is enabled and an **eieio** is detected in a store queue, stores are not gathered. Broadcasting **eieio** prevents external devices, such as a bus bridge chip, from gathering stores.
Instruction Synchronize	**isync**	—	The **isync** instruction is refetch serializing; that is, it causes the MPC7450 to wait for all prior instructions to complete first then executes which purges all instructions from the processor and then refetches the next instruction. The **isync** instruction is not executed until all previous instructions complete to the point where they cannot cause an exception. The **isync** instruction does not wait for all pending stores in the store queue to complete. Any instruction after an **isync** sees all effects of prior instructions occurring before the **isync**.

2.4.5.3 Memory Control Instructions

Memory control instructions can be classified as follows:

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

This section describes the user-level cache management instructions defined by the VEA. See Section 2.4.6.3, “Memory Control Instructions,” for information about supervisor-level cache, segment register manipulation, and translation lookaside buffer management instructions. For a complete description of the bus operations caused by cache control instructions, see Section 3.8.2, “Bus Operations Caused by Cache Control Instructions.”

2.4.5.3.1 User-Level Cache Instructions

The instructions summarized in this section help user-level programs manage on-chip caches if they are implemented. See Chapter 3, “L1, L2, and L3 Cache Operation,” for more information about cache topics. The following sections describe how these operations are treated with respect to the MPC7450’s caches.

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

Note that the MPC7450 interprets cache control instructions (**icbi**, **dcbi**, **dcbf**, **dcbz**, and **dcbst**) as if they pertain only to the local L1, and L2, and L3 caches. A **dcbz** (with M set) is always broadcast on the bus interface if it does not hit as modified in any on-chip cache.

All cache control instructions to direct-store space are no-ops. For information how cache control instructions affect the L2 cache, see 3.6.4, “L2 Cache Operation.”

Table 2-78 summarizes the cache instructions defined by the VEA. Note that these instructions are accessible to user-level programs.

Table 2-78. User-Level Cache Instructions

Name	Mnemonic	Syntax	Implementation Notes
Data Cache Block Touch [1]	**dcbt**	**r**A,**r**B	The VEA defines this instruction to allow for potential system performance enhancements through the use of software-initiated prefetch hints. Implementations are not required to take any action based on execution of this instruction, but they can prefetch the cache block corresponding to the EA into their cache. When **dcbt** executes, the MPC7450 checks for protection violations (as for a load instruction). This instruction is treated as a no-op for the following cases: • The access causes a protection violation. • The page is mapped cache-inhibited or direct-store (T = 1). • The cache is locked or disabled • HID0[NOPTI] = 1 Otherwise, if no data is in the cache location, the MPC7450 requests a cache line fill. Data brought into the cache is validated as if it were a load instruction. The memory reference of a **dcbt** sets the reference bit.
Data Cache Block Touch for Store [1]	**dcbtst**	**r**A,**r**B	This instruction **dcbtst** can be noped by setting HID0[NOPTI]. The **dcbtst** instruction behaves similarly to a **dcbt** instruction, except that the line fill request on the bus is signaled as read or read-claim, and the data is marked as exclusive in the L1 data cache if there is no shared response on the bus. More specifically, the following cases occur depending on where the line currently exists or does not exist in the MPC7450. • dcbtst hits in the L1 data cache. In this case, the dcbtst does nothing and the state of the line in the cache is not changed. Thus, if the line was in the shared state, a subsequent store hits on this shared line and incur the associated latency penalties. • dcbtst misses in the L1 data cache and hits in the L2 or L3 cache. In this case, the dcbtst will reload the L1 data cache with the state found in the L2 cache. Again, if the line was in the shared state in the L2, a subsequent store will hit on this shared line and incur the associated latency penalties. • dcbtst misses in L1 data cache, L2, and L3 caches. In this case, MPC7450 will request the line from memory with read or read-claim and reload the L1 data cache in the exclusive state. As subsequent store will hit on exclusive and can perform the store to the L1 data cache immediately. In addition, a **dcbtst** instruction will be no-oped if the target address of the **dcbtst** is mapped as write-through.

Table 2-78. User-Level Cache Instructions (continued)

Name	Mnemonic	Syntax	Implementation Notes
Data Cache Block Set to Zero	**dcbz**	**r**A,**r**B	The EA is computed, translated, and checked for protection violations. For cache hits, 32 bytes of zeros are written to the cache block and the tag is marked modified. For cache misses with the replacement block marked not modified, the zero reload is performed and the cache block is marked modified. However, if the replacement block is marked modified, the contents are written back to memory first. The instruction takes an alignment exception if the cache is locked or disabled or if the cache is marked WT or CI. If WIMG = xx1x (coherency enforced), the address is broadcast to the bus before the zero reload fill. The exception priorities (from highest to lowest) are as follows: 1 Cache disabled—Alignment exception 2 Cache is locked—Alignment exception 3 Page marked write-through or cache-inhibited—alignment exception 4 BAT protection violation—DSI exception 5 TLB protection violation—DSI exception **dcbz** is broadcast if WIMG = xx1x (coherency enforced).
Data Cache Block Allocate	**dcba**	**r**A,**r**B	The EA is computed, translated, and checked for protection violations. For cache hits, 32 bytes of zeros are written to the cache block and the tag is marked modified. For cache misses with the replacement block marked non-dirty, the zero reload is performed and the cache block is marked modified. However, if the replacement block is marked modified, the contents are written back to memory first. The instruction performs a no-op if the cache is locked or disabled or if the cache is marked WT or CI. If WIMG =xx1x (coherency enforced), the address is broadcast to the bus before the zero reload fill. A no-op occurs for the following: • Cache is disabled • Cache is locked • Page marked write-through or cache-inhibited • BAT protection violation • TLB protection violation **dcba** is broadcast if WIMG = xx1x (coherency enforced).
Data Cache Block Store	**dcbst**	**r**A,**r**B	The EA is computed, translated, and checked for protection violations. • For cache hits with the tag marked not modified, no further action is taken. • For cache hits with the tag marked modified, the cache block is written back to memory and marked exclusive. If WIMG = xx1x (coherency enforced) **dcbst** is broadcast. The instruction acts like a load with respect to address translation and memory protection. It executes regardless of whether the cache is disabled or locked. The exception priorities (from highest to lowest) for **dcbst** are as follows: 1 BAT protection violation—DSI exception 2 TLB protection violation—DSI exception

Table 2-78. User-Level Cache Instructions (continued)

Name	Mnemonic	Syntax	Implementation Notes
Data Cache Block Flush	**dcbf**	**r**A,**r**B	The EA is computed, translated, and checked for protection violations: • For cache hits with the tag marked modified, the cache block is written back to memory and the cache entry is invalidated. • For cache hits with the tag marked not modified, the entry is invalidated. • For cache misses, no further action is taken. A **dcbf** is broadcast if WIMG = xx1x (coherency enforced).The instruction acts like a load with respect to address translation and memory protection. It executes regardless of whether the cache is disabled or locked. The exception priorities (from highest to lowest) for **dcbf** are as follows: 1 BAT protection violation—DSI exception 2 TLB protection violation—DSI exception
Instruction Cache Block Invalidate	**icbi**	**r**A,**r**B	This instruction is broadcast on the bus if WIMG = xx1x. **icbi** should always be followed by a **sync** and an **isync** to make sure that the effects of the **icbi** are seen by the instruction fetches following the **icbi** itself.

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

2.4.5.4 Optional External Control Instructions

The PowerPC architecture defines an optional external control feature that, if implemented, is supported by the two external control instructions, **eciwx** and **ecowx**. These instructions allow a user-level program to communicate with a special-purpose device. These instructions are provided in the MPC7450 and are summarized in Table 2-79.

Table 2-79. External Control Instructions

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

The **eciwx**/**ecowx** instructions let a system designer map special devices in an alternative way. The MMU translation of the EA is not used to select the special device, since it is used in most instructions such as loads and stores. Rather, the EA is used as an address operand that is passed to the device over the address bus. Four other signals (the burst and size signals on the system bus) are used to select the device; these four signals output the 4-bit resource ID (RID) field located in the EAR. The **eciwx** instruction also loads a word from the data bus that is output by the special device. For more information about the relationship between these instructions and the system interface, refer to Chapter 8, “Signal Descriptions.”

2.4.6 PowerPC OEA Instructions

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

2.4.6.1 System Linkage Instructions

This section describes the system linkage instructions (see Table 2-80). The user-level **sc** instruction lets a user program call on the system to perform a service and causes the processor to take a system call exception. The supervisor-level **rfi** instruction is used for returning from an exception handler.

Table 2-80. System Linkage Instructions—OEA

Name	Mnemonic	Syntax	Implementation Notes
System Call	**sc**	—	The **sc** instruction is context-synchronizing.
Return from Interrupt	**rfi**	—	The **rfi** instruction is context-synchronizing. For the MPC7450, this means the **rfi** instruction works its way to the final stage of the execution pipeline, updates architected registers, and redirects the instruction flow.

2.4.6.2 Processor Control Instructions

The instructions listed in Table 2-81 provide access to the segment registers for 32-bit implementations. These instructions operate completely independently of the MSR[IR] and MSR[DR] bit settings. Refer to "Synchronization Requirements for Special Registers and for Lookaside Buffers," in Chapter 2, "Register Set," of the *Programming Environments Manual* for serialization requirements and other recommended precautions to observe when manipulating the segment registers.

Table 2-81. Segment Register Manipulation Instructions (OEA)

Name	Mnemonic	Syntax	Implementation Notes
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	—

The processor control instructions used to access the MSR and the SPRs are discussed in this section. Table 2-82 lists instructions for accessing the MSR.

Table 2-82. Move To/From Machine State Register Instructions

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

The OEA defines encodings of **mtspr** and **mfspr** to provide access to supervisor-level registers. The instructions are listed in Table 2-83.

Table 2-83. Move To/From Special-Purpose Register Instructions (OEA)

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

Encodings for the architecture-defined SPRs are listed in Table 2-73. Encodings for MPC7450-specific, supervisor-level SPRs are listed in Table 2-74. Simplified mnemonics are provided for **mtspr** and **mfspr** in Appendix F, "Simplified Mnemonics," in the *Programming Environments Manual*.

Table 2-84 lists the SPR numbers for supervisor-level PowerPC SPR accesses.

Table 2-84. Supervisor-Level PowerPC SPR Encodings

Register Name	SPR[1]			Access	mfspr/mtspr
	Decimal	spr[5–9]	spr[0–4]		
DABR[2]	1013	11111	10101	Supervisor (OEA)	Both
DAR	19	00000	10011	Supervisor (OEA)	Both
DBAT0L	537	10000	11001	Supervisor (OEA)	Both
DBAT0U	536	10000	11000	Supervisor (OEA)	Both
DBAT1L	539	10000	11011	Supervisor (OEA)	Both
DBAT1U	538	10000	11010	Supervisor (OEA)	Both
DBAT2L	541	10000	11101	Supervisor (OEA)	Both
DBAT2U	540	10000	11100	Supervisor (OEA)	Both
DBAT3L	543	10000	11111	Supervisor (OEA)	Both
DBAT3U	542	10000	11110	Supervisor (OEA)	Both
DEC	22	00000	10110	Supervisor (OEA)	Both
DSISR	18	00000	10010	Supervisor (OEA)	Both
EAR [2]	282	01000	11010	Supervisor (OEA)	Both
IBAT0L	529	10000	10001	Supervisor (OEA)	Both
IBAT0U	528	10000	10000	Supervisor (OEA)	Both
IBAT1L	531	10000	10011	Supervisor (OEA)	Both
IBAT1U	530	10000	10010	Supervisor (OEA)	Both
IBAT2L	533	10000	10101	Supervisor (OEA)	Both
IBAT2U	532	10000	10100	Supervisor (OEA)	Both
IBAT3L	535	10000	10111	Supervisor (OEA)	Both
IBAT3U	534	10000	10110	Supervisor (OEA)	Both

Table 2-84. Supervisor-Level PowerPC SPR Encodings (continued)

Register Name	SPR[1]			Access	mfspr/mtspr
	Decimal	spr[5–9]	spr[0–4]		
MMCR0 [2]	952	11101	11000	Supervisor (OEA)	Both
MMCR1 [2]	956	11101	11100	Supervisor (OEA)	Both
PIR [2]	1023	11111	11111	Supervisor (OEA)	Both
PMC1 [2]	953	11101	11001	Supervisor (OEA)	Both
PMC2 [2]	954	11101	11010	Supervisor (OEA)	Both
PMC3 [2]	957	11101	11101	Supervisor (OEA)	Both
PMC4 [2]	958	11101	11110	Supervisor (OEA)	Both
PMC5 [2]	945	11101	10001	Supervisor (OEA)	Both
PMC6 [2]	946	11101	10010	Supervisor (OEA)	Both
PVR	287	01000	11111	Supervisor (OEA)	**mfspr**
SDR1	25	00000	11001	Supervisor (OEA)	Both
SIAR [2]	955	11101	11011	Supervisor (OEA)	Both
SPRG0	272	01000	10000	Supervisor (OEA)	Both
SPRG1	273	01000	10001	Supervisor (OEA)	Both
SPRG2	274	01000	10010	Supervisor (OEA)	Both
SPRG3	275	01000	10011	Supervisor (OEA)	Both
SRR0	26	00000	11010	Supervisor (OEA)	Both
SRR1	27	00000	11011	Supervisor (OEA)	Both
TBL[3]	284	01000	11100	Supervisor (OEA)	**mtspr**
TBU [3]	285	01000	11101	Supervisor (OEA)	**mtspr**

[1] 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.

[2] Optional register defined by the PowerPC architecture.

[3] The TB registers are referred to as TBRs rather than SPRs and can be written to using the mtspr instruction in supervisor mode and the TBR numbers here. The TB registers can be read in user mode using the mftb instruction and specifying TBR 268 for TBL and TBR 269 for TBU.

Encodings for the supervisor-level MPC7450-specific SPRs are listed in Table 2-85.

Table 2-85. Supervisor-Level SPR Encodings for MPC7450-Defined Registers

Register Name	SPR [1]			Access	mfspr/mtspr
	Decimal	spr[5–9]	spr[0–4]		
BAMR	951	11101	10111	Supervisor (OEA)	Both
DBAT4L[2]	569	10001	11001	Supervisor (OEA)	Both
DBAT4U [2]	568	10001	11000	Supervisor (OEA)	Both
DBAT5L [2]	571	10001	11011	Supervisor (OEA)	Both
DBAT5U [2]	570	10001	11010	Supervisor (OEA)	Both
DBAT6L [2]	573	10001	11101	Supervisor (OEA)	Both
DBAT6U [2]	572	10001	11100	Supervisor (OEA)	Both
DBAT7L [2]	575	10001	11111	Supervisor (OEA)	Both
DBAT7U [2]	574	10001	11110	Supervisor (OEA)	Both
HID0	1008	11111	10000	Supervisor (OEA)	Both
HID1	1009	11111	10001	Supervisor (OEA)	Both
IABR	1010	11111	10010	Supervisor (OEA)	Both
IBAT4L [2]	561	10001	10001	Supervisor (OEA)	Both
IBAT4U [2]	560	10001	10000	Supervisor (OEA)	Both
IBAT5L [2]	563	10001	10011	Supervisor (OEA)	Both
IBAT5U [2]	562	10001	10010	Supervisor (OEA)	Both
IBAT6L [2]	565	10001	10101	Supervisor (OEA)	Both
IBAT6U [2]	564	10001	10100	Supervisor (OEA)	Both
IBAT7L [2]	567	10001	10111	Supervisor (OEA)	Both
IBAT7U [2]	566	10001	10110	Supervisor (OEA)	Both
ICTC	1019	11111	11011	Supervisor (OEA)	Both
ICTRL	1011	11111	10011	Supervisor (OEA)	Both
L2CR	1017	11111	11001	Supervisor (OEA)	Both
L2ERRADDR [3]	995	11111	00011	Supervisor (OEA)	**mfspr**
L2ERRATTR [3]	994	11111	00010	Supervisor (OEA)	Both
L2ERREADDR [3]	996	11111	00100	Supervisor (OEA)	**mfspr**
L2CAPTDATAHI [3]	988	11110	11100	Supervisor (OEA)	**mfspr**
L2CAPTDATALO [3]	989	11110	11101	Supervisor (OEA)	**mfspr**
L2CAPTECC [3]	990	11110	11110	Supervisor (OEA)	**mfspr**
L2ERRCTL [3]	997	11111	00101	Supervisor (OEA)	Both

Table 2-85. Supervisor-Level SPR Encodings for MPC7450-Defined Registers (continued)

Register Name	SPR [1]			Access	mfspr/mtspr
	Decimal	spr[5–9]	spr[0–4]		
L2ERRINJCTL [3]	987	11110	11011	Supervisor (OEA)	Both
L2ERRINJHI [3]	985	11110	11001	Supervisor (OEA)	Both
L2ERRINJLO [3]	986	11110	11010	Supervisor (OEA)	Both
L2ERRDET [3]	991	11110	11111	Supervisor (OEA)	Special [4]
L2ERRDIS [3]	992	11111	00000	Supervisor (OEA)	Both
L2ERRINTEN [3]	993	11111	00001	Supervisor (OEA)	Both
L3CR [5]	1018	11111	11010	Supervisor (OEA)	Both
L3ITCR0 [5]	984	11111	11000	Supervisor (OEA)	Both
L3ITCR1 [6]	1001	11111	01001	Supervisor (OEA)	Both
L3ITCR2 [6]	1002	11111	01010	Supervisor (OEA)	Both
L3ITCR3 [6]	1003	11111	01011	Supervisor (OEA)	Both
L3OHCR [6]	1000	11111	01000	Supervisor (OEA)	Both
L3PM [5]	983	11110	10111	Supervisor (OEA)	Both
LDSTCR	1016	11111	1000	Supervisor (OEA)	Both
MMCR2	944	11101	10000	Supervisor (OEA)	Both
MSSCR0	1014	11111	10110	Supervisor (OEA)	Both
MSSSR0	1015	11111	10111	Supervisor (OEA)	Both
PTEHI	981	11110	10101	Supervisor (OEA)	Both
PTELO	982	11110	10110	Supervisor (OEA)	Both
SPRG4 [2]	276	01000	10100	Supervisor (OEA)	Both
SPRG5 [2]	277	01000	10101	Supervisor (OEA)	Both
SPRG6 [2]	278	01000	100110	Supervisor (OEA)	Both
SPRG7 [2]	279	01000	10111	Supervisor (OEA)	Both
SVR [7]	286	01000	11110	Supervisor (OEA)	**mfspr**
TLBMISS	980	11110	10100	Supervisor (OEA)	Both

[1] 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.

[2] MPC7445-, MPC7447-, MPC7455-, and MPC7457-specific register that may not be supported on other processors that implement the PowerPC architecture.

[3] MPC7448-specific register.

[4] Most bits are bit reset/write 1 clear. A write of 0 to a bit does not change it. A write of 1 to a bit clears it. Reads act normally.

[5] MPC7451-, MPC7455-, MPC7457-specific register.

[6] MPC7457-specific register.

[7] MPC7448-specific register.

2.4.6.3 Memory Control Instructions

Memory control instructions include the following:

- Cache management instructions (supervisor-level and user-level)
- Translation lookaside buffer management instructions

This section describes supervisor-level memory control instructions. Section 2.4.5.3, “Memory Control Instructions,” describes user-level memory control instructions.

2.4.6.3.1 Supervisor-Level Cache Management Instruction

Table 2-86 lists the only supervisor-level cache management instruction.

Table 2-86. Supervisor-Level Cache Management Instruction

Name	Mnemonic	Syntax	Implementation Notes
Data Cache Block Invalidate	**dcbi**	**r**A,**r**B	The **dcbi** instruction is executed identically to the **dcbf** instruction except that it is privileged (supervisor-only). See Section 2.4.5.3.1, “User-Level Cache Instructions.”

See Section 2.4.5.3.1, “User-Level Cache Instructions,” for cache instructions that provide user-level programs the ability to manage the on-chip caches. If the effective address references a direct-store segment, the instruction is treated as a no-op.

2.4.6.3.2 Translation Lookaside Buffer Management Instructions

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

Implementation Note—The MPC7450 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 when HID0[STEN] = 1.

For more information on **tlbld** and **tlbli** refer to Section 2.4.8, “Implementation-Specific Instructions.”

See Chapter 5, “Memory Management,” for more information about TLB operations. Table 2-87 summarizes the operation of the TLB instructions in the MPC7450. Note that the broadcast of **tlbie** and **tlbsync** instructions is enabled by the setting of HID1[SYNCBE].

Table 2-87. Translation Lookaside Buffer Management Instruction

Name	Mnemonic	Syntax	Implementation Notes
TLB Invalidate Entry	**tlbie**	**r**B	Invalidates both ways in both instruction and data TLB entries at the index provided by EA[14–19]. It executes regardless of the MSR[DR] and MSR[IR] settings. To invalidate all entries in both TLBs, the programmer should issue 64 **tlbie** instructions that each successively increment this field.
Load Data TLB Entry	**tlbld**	**r**B	Load Data TLB Entry Loads fields from the PTEHI and PTELO and the EA in **r**B to the way defined in **r**B[31].
Load Instruction TLB Entry	**tlbli**	**r**B	Load Instruction TLB Entry Loads fields from the PTEHI and PTELO and the EA in **r**B to the way defined in **r**B[31].
TLB Synchronize	**tlbsync**	—	TLBSYNC is broadcast.

Implementation Note—The **tlbia** instruction is optional for an implementation if its effects can be achieved through some other mechanism. Therefore, it is not implemented on the MPC7450. As described above, **tlbie** can be used to invalidate a particular index of the TLB based on EA[14–19]—a sequence of 64 **tlbie** instructions followed by a **tlbsync** instruction invalidates all the TLB structures (for EA[14–19] = 0, 1, 2, . . . , 63). Attempting to execute **tlbia** causes an illegal instruction program exception.

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

2.4.7 Recommended Simplified Mnemonics

The description of each instruction includes the mnemonic and a formatted list of operands. PowerPC-architecture-compliant assemblers support the mnemonics and operand lists. To simplify assembly language programming, a set of simplified mnemonics and symbols is provided for some of the most frequently-used instructions; refer to Appendix F, “Simplified Mnemonics,” in the *Programming Environments Manual* for a complete list. 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.

2.4.8 Implementation-Specific Instructions

This section provides the details for the two MPC7450 implementation-specific instructions—**tlbld** and **tlbli**.

tlbld tlbld

Load Data TLB Entry Integer Unit

tlbld **r**B

▨ Reserved

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

```
EA ← (rB)
TLB entry created from PTEHI and PTELO
DTLB entry selected by EA[14–19] and rB[31] ← created TLB entry
```

The EA is the contents of **r**B. The tlbld instruction loads the contents of the PTEHI special purpose register and PTELO special purpose register into the selected data TLB entry. The set of the data TLB to be loaded is determined by EA[14–19]. The way to be loaded is determined by rB[31]. EA[10–13] are stored in the tag portion of the TLB and are used to match a new EA when a new EA is being translated.

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

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

Note that if extended addressing is not enabled (HID0[XAEN] = 0), then PTELO[20–22] and PTELO[29] should be cleared (zero) by software when executing a **tlbld** instruction.

This is a supervisor-level instruction; it is also a MPC7450-specific instruction, and not part of the PowerPC instruction set.

Other registers altered:

- None

tlbli tlbli

Load Instruction TLB Entry Integer Unit

tlbli **r**B

▨ Reserved

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

```
EA ← (rB)
TLB entry created from PTEHI and PTELO
ITLB entry selected by EA[14–19] and rB[31] ← created TLB entry
```

The EA is the contents of **r**B. The **tlbli** instruction loads an instruction TLB entry. The **tlbli** instruction loads the contents of the PTEHI special purpose register and PTELO special purpose register into a selected instruction TLB entry. The set of the instruction TLB to be loaded is determined by EA[14–19]. The way to be loaded is determined by rB[31]. EA[10–13] are stored in the tag portion of the TLB and are used to match a new EA when a new EA is being translated.

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

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

Note that if extended addressing is not enabled (HID0[XAEN]=0) then PTELO[20–22] and PTELO[29] should be cleared (set to zero) by software when executing a **tlbli** 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 MPC7450-specific instruction, and not part of the PowerPC instruction set.

Other registers altered:

- None

2.5 AltiVec Instructions

The following sections provide a general summary of the instructions and addressing modes defined by the AltiVec Instruction Set Architecture (ISA). For specific details on the AltiVec instructions see the *AltiVec Technology Programming Environments Manual* and Chapter 7, “AltiVec Technology Implementation.” AltiVec instructions belong primarily to the UISA, unless otherwise noted. AltiVec instructions are divided into the following categories:

- Vector integer arithmetic instructions—These include arithmetic, logical, compare, rotate and shift instructions, described in Section 2.4.4.1, “Integer Instructions.”

- Vector floating-point arithmetic instructions—These floating-point arithmetic instructions and floating-point modes are described in Section 2.4.4.2, “Floating-Point Instructions.”
- Vector load and store instructions—These load and store instructions for vector registers are described in Section 2.6.3, “Vector Load and Store Instructions.”
- Vector permutation and formatting instructions—These include pack, unpack, merge, splat, permute, select, and shift instructions, and are described in Section 2.6.5, “Vector Permutation and Formatting Instructions.”
- Processor control instructions—These instructions are used to read and write from the AltiVec status and control register, and are described in Section 2.4.4.6, “Processor Control Instructions.”
- Memory control instructions—These instructions are used for managing caches (user level and supervisor level), and are described in Section 2.7.1, “AltiVec Vector Memory Control Instructions.”

This grouping of instructions does not necessarily indicate the execution unit that processes a particular instruction or group of instructions within a processor implementation.

Integer instructions operate on byte, half-word, and word operands. Floating-point instructions operate on single-precision operands. The AltiVec ISA uses instructions that are four bytes long and word-aligned. It provides for byte, half-word, word, and quad-word operand fetches and stores between memory and the vector registers (VRs).

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 AltiVec ISA supports both big-endian and little-endian byte ordering. The default byte and bit ordering is big-endian; see “Byte Ordering,” in Chapter 3, “Operand Conventions,” of the *AltiVec Technology Programming Environments Manual* for more information.

2.6 AltiVec UISA Instructions

This section describes the instructions defined in the AltiVec user instruction set architecture (UISA).

2.6.1 Vector Integer Instructions

The following are categories for vector integer instructions:

- Vector integer arithmetic instructions
- Vector integer compare instructions
- Vector integer logical instructions
- Vector integer rotate and shift instructions

Integer instructions use the content of VRs as source operands and also place results into VRs. Setting the Rc bit of a vector compare instruction causes the CR6 field of the PowerPC condition register (CR) to be updated; refer to Section 2.6.1.2, “Vector Integer Compare Instructions,” for more details.

The AltiVec integer instructions treat source operands as signed integers unless the instruction is explicitly identified as performing an unsigned operation. For example, both the Vector Add Unsigned Word Modulo (**vadduwm**) and Vector Multiply Odd Unsigned Byte (**vmuloub**) instructions interpret the operands as unsigned integers.

2.6.1.1 Vector Integer Arithmetic Instructions

Table 2-88 lists the integer arithmetic instructions for the processors that implement the PowerPC architecture.

Table 2-88. Vector Integer Arithmetic Instructions

Name	Mnemonic	Syntax
Vector Add Unsigned Integer [b,h,w] Modulo1	**vaddubm** **vadduhm** **vadduwm**	vD,vA,vB
Vector Add Unsigned Integer [b,h,w] Saturate	**vaddubs** **vadduhs** **vadduws**	vD,vA,vB
Vector Add Signed Integer [b.h.w] Saturate	**vaddsbs** **vaddshs** **vaddsws**	vD,vA,vB
Vector Add and Write Carry-out Unsigned Word	**vaddcuw**	vD,vA,vB
Vector Subtract Unsigned Integer Modulo	**vsububm** **vsubuhm** **vsubuwm**	vD,vA,vB
Vector Subtract Unsigned Integer Saturate	**vsububs** **vsubuhs** **vsubuws**	vD,vA,vB
Vector Subtract Signed Integer Saturate	**vsubsbs** **vsubshs** **vsubsws**	vD,vA,vB
Vector Subtract and Write Carry-out Unsigned Word	**vsubcuw**	vD,vA,vB
Vector Multiply Odd Unsigned Integer [b,h] Modulo	**vmuloub** **vmulouh**	vD,vA,vB
Vector Multiply Odd Signed Integer [b,h] Modulo	**vmulosb** **vmulosh**	vD,vA,vB
Vector Multiply Even Unsigned Integer [b,h] Modulo	**vmuleub** **vmuleuh**	vD,vA,vB
Vector Multiply Even Signed Integer [b,h] Modulo	**vmulesb** **vmulesh**	vD,vA,vB
Vector Multiply-High and Add Signed Half-Word Saturate	**vmhaddshs**	vD,vA,vB, vC
Vector Multiply-High Round and Add Signed Half-Word Saturate	**vmhraddshs**	vD,vA,vB,vC
Vector Multiply-Low and Add Unsigned Half-Word Modulo	**vmladduhm**	vD,vA,vB,vC

Table 2-88. Vector Integer Arithmetic Instructions (continued)

Name	Mnemonic	Syntax
Vector Multiply-Sum Unsigned Integer [b,h] Modulo	**vmsumubm** **vmsumuhm**	**v**D,**v**A,**v**B,**v**C
Vector Multiply-Sum Signed Half-Word Saturate	**vmsumshs**	**v**D,**v**A,**v**B,**v**C
Vector Multiply-Sum Unsigned Half-Word Saturate	**vmsumuhs**	**v**D,**v**A,**v**B,**v**C
Vector Multiply-Sum Mixed Byte Modulo	**vmsummbm**	**v**D,**v**A,**v**B,**v**C
Vector Multiply-Sum Signed Half-Word Modulo	**vmsumshm**	**v**D,**v**A,**v**B,**v**C
Vector Sum Across Signed Word Saturate	**vsumsws**	**v**D,**v**A,**v**B
Vector Sum Across Partial (1/2) Signed Word Saturate	**vsum2sws**	**v**D,**v**A,**v**B
Vector Sum Across Partial (1/4) Unsigned Byte Saturate	**vsum4ubs**	**v**D,**v**A,**v**B
Vector Sum Across Partial (1/4) Signed Integer Saturate	**vsum4sbs** **vsum4shs**	**v**D,**v**A,**v**B
Vector Average Unsigned Integer	**vavgub** **vavguh** **vavguw**	**v**D,**v**A,**v**B
Vector Average Signed Integer	**vavgsb** **vavgsh** **vavgsw**	**v**D,**v**A,**v**B
Vector Maximum Unsigned Integer	**vmaxub** **vmaxuh** **vmaxuw**	**v**D,**v**A,**v**B
Vector Maximum Signed Integer	**vmaxsb** **vmaxsh** **vmaxsw**	**v**D,**v**A,**v**B
Vector Minimum Unsigned Integer	**vminub** **vminuh** **vminuw**	**v**D,**v**A,**v**B
Vector Minimum Signed Integer	**vminsb** **vminsh** **vminsw**	**v**D,**v**A,**v**B

2.6.1.2 Vector Integer Compare Instructions

The vector integer compare instructions algebraically or logically compare the contents of the elements in vector register **v**A with the contents of the elements in **v**B. Each compare result vector is comprised of TRUE (0xFF, 0xFFFF, 0xFFFF_FFFF) or FALSE (0x00, 0x0000, 0x0000_0000) elements of the size specified by the compare source operand element (byte, half word, or word). The result vector can be directed to any VR and can be manipulated with any of the instructions as normal data (for example, combining condition results).

Vector compares provide equal-to and greater-than predicates. Others are synthesized from these by logically combining or inverting result vectors.

The integer compare instructions (shown in Table 2-90) can optionally set the CR6 field of the PowerPC condition register. If Rc = 1 in the vector integer compare instruction, then CR6 is set to reflect the result of the comparison, as follows in Table 2-89.

Table 2-89. CR6 Field Bit Settings for Vector Integer Compare Instructions

CR Bit	CR6 Bit	Vector Compare
24	0	1 Relation is true for all element pairs (that is, **v**D is set to all ones)
25	1	0
26	2	1 Relation is false for all element pairs (that is, register **v**D is cleared)
27	3	0

Table 2-90 summarizes the vector integer compare instructions.

Table 2-90. Vector Integer Compare Instructions

Name	Mnemonic	Syntax
Vector Compare Greater than Unsigned Integer	**vcmpgtub**[.] **vcmpgtuh**[.] **vcmpgtuw**[.]	**v**D,**v**A,**v**B
Vector Compare Greater than Signed Integer	**vcmpgtsb**[.] **vcmpgtsh**[.] **vcmpgtsw**[.]	**v**D,**v**A,**v**B
Vector Compare Equal to Unsigned Integer	**vcmpequb**[.] **vcmpequh**[.] **vcmpequw**[.]	**v**D,**v**A,**v**B

2.6.1.3 Vector Integer Logical Instructions

The vector integer logical instructions shown in Table 2-91 perform bit-parallel operations on the operands.

Table 2-91. Vector Integer Logical Instructions

Name	Mnemonic	Syntax
Vector Logical AND	**vand**	**v**D,**v**A,**v**B
Vector Logical OR	**vor**	**v**D,**v**A,**v**B
Vector Logical XOR	**vxor**	**v**D,**v**A,**v**B
Vector Logical AND with Complement	**vandc**	**v**D,**v**A,**v**B
Vector Logical NOR	**vnor**	**v**D,**v**A,**v**B

2.6.1.4 Vector Integer Rotate and Shift Instructions

The vector integer rotate instructions are summarized in Table 2-92.

Table 2-92. Vector Integer Rotate Instructions

Name	Mnemonic	Syntax
Vector Rotate Left Integer	**vrlb** **vrlh** **vrlw**	**v**D,**v**A,**v**B

The vector integer shift instructions are summarized in Table 2-93.

Table 2-93. Vector Integer Shift Instructions

Name	Mnemonic	Syntax
Vector Shift Left Integer	**vslb** **vslh** **vslw**	**v**D,**v**A,**v**B
Vector Shift Right Integer	**vsrb** **vsrh** **vsrw**	**v**D,**v**A,**v**B
Vector Shift Right Algebraic Integer	**vsrab** **vsrah** **vsraw**	**v**D,**v**A,**v**B

2.6.2 Vector Floating-Point Instructions

This section describes the vector floating-point instructions that include the following:

- Vector floating-point arithmetic instructions
- Vector floating-point rounding and conversion instructions
- Vector floating-point compare instructions
- Vector floating-point estimate instructions

The AltiVec floating-point data format complies with the ANSI/IEEE-754 standard as defined for single precision. A quantity in this format represents a signed normalized number, a signed denormalized number, a signed zero, a signed infinity, a quiet not a number (QNaN), or a signaling NaN (SNaN). Operations conform to the description in the section “AltiVec Floating-Point Instructions-UISA,” in Chapter 3, “Operand Conventions,” of the *AltiVec Technology Programming Environments Manual*.

The AltiVec ISA does not report IEEE exceptions but rather produces default results as specified by the Java/IEEE/C9X Standard; for further details on exceptions see “Floating-Point Exceptions,” in Chapter 3, “Operand Conventions,” of the *AltiVec Technology Programming Environments Manual*.

2.6.2.1 Vector Floating-Point Arithmetic Instructions

The floating-point arithmetic instructions are summarized in Table 2-94.

Table 2-94. Vector Floating-Point Arithmetic Instructions

Name	Mnemonic	Syntax
Vector Add Floating-Point	**vaddfp**	**v**D,**v**A,**v**B
Vector Subtract Floating-Point	**vsubfp**	**v**D,**v**A,**v**B
Vector Maximum Floating-Point	**vmaxfp**	**v**D,**v**A,**v**B
Vector Minimum Floating-Point	**vminfp**	**v**D,**v**A,**v**B

2.6.2.2 Vector Floating-Point Multiply-Add Instructions

Vector multiply-add instructions are critically important to performance because a multiply followed by a data dependent addition is the most common idiom in DSP algorithms. In most implementations, floating-point multiply-add instructions perform with the same latency as either a multiply or add alone, thus doubling performance in comparing to the otherwise serial multiply and adds.

AltiVec floating-point multiply-add instructions fuse (a multiply-add fuse implies that the full product participates in the add operation without rounding, only the final result rounds). This not only simplifies the implementation and reduces latency (by eliminating the intermediate rounding) but also increases the accuracy compared to separate multiply and adds.

The floating-point multiply-add instructions are summarized in Table 2-95.

Table 2-95. Vector Floating-Point Multiply-Add Instructions

Name	Mnemonic	Syntax
Vector Multiply-Add Floating-Point	**vmaddfp**	**v**D,**v**A,**v**C,**v**B
Vector Negative Multiply-Subtract Floating-Point	**vnmsubfp**	**v**D,**v**A,**v**C,**v**B

2.6.2.3 Vector Floating-Point Rounding and Conversion Instructions

All AltiVec floating-point arithmetic instructions use the IEEE default rounding mode round-to-nearest. The AltiVec ISA does not provide the IEEE directed rounding modes.

The AltiVec ISA provides separate instructions for converting floating-point numbers to integral floating-point values for all IEEE rounding modes as follows:

- Round-to-nearest (**vrfin**) (round)
- Round-toward-zero (**vrfiz**) (truncate)
- Round-toward-minus-infinity (**vrfim**) (floor)
- Round-toward-positive-infinity (**vrfip**) (ceiling)

Floating-point conversions to integers (**vctuxs**, **vctsxs**) use round-toward-zero (truncate) rounding. The floating-point rounding instructions are shown in Table 2-96.

Table 2-96. Vector Floating-Point Rounding and Conversion Instructions

Name	Mnemonic	Syntax
Vector Round to Floating-Point Integer Nearest	**vrfin**	vD,vB
Vector Round to Floating-Point Integer toward Zero	**vrfiz**	vD,vB
Vector Round to Floating-Point Integer toward Positive Infinity	**vrfip**	vD,vB
Vector Round to Floating-Point Integer toward Minus Infinity	**vrfim**	vD,vB
Vector Convert from Unsigned Fixed-Point Word	**vcfux**	vD,vB,UIMM
Vector Convert from Signed Fixed-Point Word	**vcfsx**	vD,vB,UIMM
Vector Convert to Unsigned Fixed-Point Word Saturate	**vctuxs**	vD,vB,UIMM
Vector Convert to Signed Fixed-Point Word Saturate	**vctsxs**	vD,vB,UIMM

2.6.2.4 Vector Floating-Point Compare Instructions

The floating-point compare instructions are summarized in Table 2-97.

Table 2-97. Vector Floating-Point Compare Instructions

Name	Mnemonic	Syntax
Vector Compare Greater Than Floating-Point [Record]	**vcmpgtfp**[.]	vD,vA,vB
Vector Compare Equal to Floating-Point [Record]	**vcmpeqfp**[.]	vD,vA,vB
Vector Compare Greater Than or Equal to Floating-Point [Record]	**vcmpgeqfp**[.]	vD,vA,vB
Vector Compare Bounds Floating-Point [Record]	**vcmpbfp**[.]	vD,vA,vB

2.6.2.5 Vector Floating-Point Estimate Instructions

The floating-point estimate instructions are summarized in Table 2-98.

Table 2-98. Vector Floating-Point Estimate Instructions

Name	Mnemonic	Syntax
Vector Reciprocal Estimate Floating-Point	**vrefp**	vD,vB
Vector Reciprocal Square Root Estimate Floating-Point	**vrsqrtefp**	vD,vB
Vector Log2 Estimate Floating-Point	**vlogefp**	vD,vB
Vector 2 Raised to the Exponent Estimate Floating-Point	**vexptefp**	vD,vB

2.6.3 Vector Load and Store Instructions

Only very basic load and store operations are provided in the AltiVec ISA. This keeps the circuitry in the memory path fast so the latency of memory operations is minimized. Instead, a powerful set of field manipulation instructions are provided to manipulate data into the desired alignment and arrangement after the data has been brought into the VRs.

Load vector indexed (**lvx**, **lvxl**) and store vector indexed (**stvx**, **stvxl**) instructions transfer an aligned quad-word vector between memory and VRs. Load vector element indexed (**lvebx**, **lvehx**, **lvewx**) and store vector element indexed instructions (**stvebx**, **stvehx**, **stvewx**) transfer byte, half-word, and word scalar elements between memory and VRs.

2.6.3.1 Vector Load Instructions

For vector load instructions, the byte, half word, word, or quad word addressed by the EA (effective address) is loaded into **v**D.

The default byte and bit ordering is big-endian as in the PowerPC architecture; see "Byte Ordering," in Chapter 3, "Operand Conventions," of the *AltiVec Technology Programming Environments Manual* for information about little-endian byte ordering.

Table 2-99 summarizes the vector load instructions.

Table 2-99. Vector Integer Load Instructions

Name	Mnemonic	Syntax
Load Vector Element Integer Indexed	**lvebx** **lvehx** **lvewx**	**v**D,**r**A,**r**B
Load Vector Element Indexed	**lvx**	**v**D,**r**A,**r**B
Load Vector Element Indexed LRU [1]	**lvxl**	**v**D,**r**A,**r**B

[1] On the MPC7450, lvxl and stvxl are interpreted to be transient. See Section 7.1.2.3, "Data Stream Touch Instructions."

2.6.3.2 Vector Load Instructions Supporting Alignment

The **lvsl** and **lvsr** instructions can be used to create the permute control vector to be used by a subsequent **vperm** instruction. Let X and Y be the contents of **v**A and **v**B specified by **vperm**. The control vector created by **lvsl** causes the **vperm** to select the high-order 16 bytes of the result of shifting the 32-byte value X || Y left by sh bytes (sh = the value in EA[60–63]). The control vector created by **lvsr** causes the **vperm** to select the low-order 16 bytes of the result of shifting X || Y right by sh bytes.

Table 2-100 summarizes the vector alignment instructions.

Table 2-100. Vector Load Instructions Supporting Alignment

Name	Mnemonic	Syntax
Load Vector for Shift Left	**lvsl**	**v**D,**r**A,**r**B
Load Vector for Shift Right	**lvsr**	**v**D,**r**A,**r**B

2.6.3.3 Vector Store Instructions

For vector store instructions, the contents of the VR used as a source (**v**S) are stored into the byte, half word, word or quad word in memory addressed by the effective address (EA). Table 2-101 provides a summary of the vector store instructions.

Table 2-101. Vector Integer Store Instructions

Name	Mnemonic	Syntax
Store Vector Element Integer Indexed	**svetbx** **svethx** **svetwx**	**v**S,**r**A,**r**B
Store Vector Element Indexed	**stvx**	**v**S,**r**A,**r**B
Store Vector Element Indexed LRU[1]	**stvxl**	**v**S,**r**A,**r**B

[1] On the MPC7450, lvxl, stvxl are interpreted to be transient. See Section 7.1.2.3, "Data Stream Touch Instructions."

2.6.4 Control Flow

AltiVec instructions can be freely intermixed with existing PowerPC instructions to form a complete program. AltiVec instructions provide a vector compare and select mechanism to implement conditional execution as the preferred mechanism to control data flow in AltiVec programs. In addition, AltiVec vector compare instructions can update the condition register thus providing the communication from AltiVec execution units to PowerPC branch instructions necessary to modify program flow based on vector data.

2.6.5 Vector Permutation and Formatting Instructions

Vector pack, unpack, merge, splat, permute, select, and shift instructions can be used to accelerate various vector math operations and vector formatting. Details of these instructions follow.

2.6.5.1 Vector Pack Instructions

Half-word vector pack instructions (**vpkuhum**, **vpkuhus**, **vpkshus**, **vpkshss**) truncate the sixteen half words from two concatenated source operands producing a single result of sixteen bytes (quad word) using either modulo (2^8), 8-bit signed-saturation, or 8-bit unsigned-saturation to perform the truncation. Similarly, word vector pack instructions (**vpkuwum**, **vpkuwus**, **vpkswus**, **vpksws**) truncate the eight words from two concatenated source operands producing a single result of eight half words using modulo (2^{16}), 16-bit signed-saturation, or 16-bit unsigned-saturation to perform the truncation.

Table 2-102 describes the vector pack instructions.

Table 2-102. Vector Pack Instructions

Name	Mnemonic	Syntax
Vector Pack Unsigned Integer [h,w] Unsigned Modulo	**vpkuhum** **vpkuwum**	**v**D, **v**A, **v**B
Vector Pack Unsigned Integer [h,w] Unsigned Saturate	**vpkuhus** **vpkuwus**	**v**D, **v**A, **v**B

Table 2-102. Vector Pack Instructions (continued)

Name	Mnemonic	Syntax
Vector Pack Signed Integer [h,w] Unsigned Saturate	**vpkshus** **vpkswus**	**v**D, **v**A, **v**B
Vector Pack Signed Integer [h,w] signed Saturate	**vpkshss** **vpkswss**	**v**D, **v**A, **v**B
Vector Pack Pixel	**vpkpx**	**v**D, **v**A, **v**B

2.6.5.2 Vector Unpack Instructions

Byte vector unpack instructions unpack the 8 low bytes (or 8 high bytes) of one source operand into 8 half words using sign extension to fill the most-significant bytes (MSBs). Half word vector unpack instructions unpack the 4 low half words (or 4 high half words) of one source operand into 4 words using sign extension to fill the MSBs.

Two special purpose forms of vector unpack are provided—the Vector Unpack Low Pixel (**vupklpx**) and the Vector Unpack High Pixel (**vupkhpx**) instructions for 1/5/5/5 αRGB pixels. The 1/5/5/5 pixel vector unpack, unpacks the four low 1/5/5/5 pixels (or four 1/5/5/5 high pixels) into four 32-bit (8/8/8/8) pixels. The 1-bit α element in each pixel is sign extended to 8 bits, and the 5-bit R, G, and B elements are each zero extended to 8 bits.

Table 2-103 describes the unpack instructions.

Table 2-103. Vector Unpack Instructions

Name	Mnemonic	Syntax
Vector Unpack High Signed Integer	**vupkhsb** **vupkhsh**	**v**D, **v**B
Vector Unpack High Pixel	**vupkhpx**	**v**D, **v**B
Vector Unpack Low Signed Integer	**vupklsb** **vupklsh**	**v**D, **v**B
Vector Unpack Low Pixel	**vupklpx**	**v**D, **v**B

2.6.5.3 Vector Merge Instructions

Byte vector merge instructions interleave the 8 low bytes or 8 high bytes from two source operands producing a result of 16 bytes. Similarly, half-word vector merge instructions interleave the 4 low half words (or 4 high half words) of two source operands producing a result of 8 half words, and word vector merge instructions interleave the 2 low words or 2 high words from two source operands producing a result of 4 words. The vector merge instruction has many uses. For example, it can be used to efficiently transpose SIMD vectors. Table 2-104 describes the merge instructions.

Table 2-104. Vector Merge Instructions

Name	Mnemonic	Syntax
Vector Merge High Integer	**vmrghb** **vmrghh** **vmrghw**	**v**D, **v**A, **v**B
Vector Merge Low Integer	**vmrglb** **vmrglh** **vmrglw**	**v**D, **v**A, **v**B

2.6.5.4 Vector Splat Instructions

When a program needs to perform arithmetic vector operations, the vector splat instructions can be used in preparation for performing arithmetic for which one source vector is to consist of elements that all have the same value. Vector splat instructions can be used to move data where it is required. For example to multiply all elements of a vector register (VR) by a constant, the vector splat instructions can be used to splat the scalar into the VR. Likewise, when storing a scalar into an arbitrary memory location, it must be splatted into a VR, and that VR must be specified as the source of the store. This guarantees that the data appears in all possible positions of that scalar size for the store. Table 2-105 describes the vector splat instruction.

Table 2-105. Vector Splat Instructions

Name	Mnemonic	Syntax
Vector Splat Integer	**vspltb** **vsplth** **vspltw**	**v**D, **v**B, UIMM
Vector Splat Immediate Signed Integer	**vspltisb** **vspltish** **vspltisw**	**v**D, SIMM

2.6.5.5 Vector Permute Instructions

Permute instructions allow any byte in any two source VRs to be directed to any byte in the destination vector. The fields in a third source operand specify from which field in the source operands the corresponding destination field is taken. The Vector Permute (**vperm**) instruction is a very powerful one that provides many useful functions. For example, it provides a way to perform table-lookups and data alignment operations. An example of how to use the **vperm** instruction in aligning data is described in “Quad-Word Data Alignment” in Chapter 3, “Operand Conventions,” of the *AltiVec Technology Programming Environments Manual*. Table 2-106 describes the vector permute instruction.

Table 2-106. Vector Permute Instruction

Name	Mnemonic	Syntax
Vector Permute	**vperm**	**v**D, **v**A,**v**B,**v**C

2.6.5.6 Vector Select Instruction

Data flow in the vector unit can be controlled without branching by using a vector compare and the Vector Select (**vsel**) instructions. In this use, the compare result vector is used directly as a mask operand to vector select instructions.The **vsel** instruction selects one field from one or the other of two source operands under control of its mask operand. Use of the TRUE/FALSE compare result vector with select in this manner produces a two instruction equivalent of conditional execution on a per-field basis. Table 2-107 describes the **vsel** instruction.

Table 2-107. Vector Select Instruction

Name	Mnemonic	Syntax
Vector Select	**vsel**	**v**D,**v**A,**v**B,**v**C

2.6.5.7 Vector Shift Instructions

The vector shift instructions shift the contents of one or of two VRs left or right by a specified number of bytes (**vslo**, **vsro**, **vsldoi**) or bits (**vsl**, **vsr**). Depending on the instruction, this shift count is specified either by low-order bits of a VR or by an immediate field in the instruction. In the former case the low-order 7 bits of the shift count register give the shift count in bits (0 ≤ count ≤ 127). Of these 7 bits, the high-order 4 bits give the number of complete bytes by which to shift and are used by **vslo** and **vsro**; the low-order 3 bits give the number of remaining bits by which to shift and are used by **vsl** and **vsr**.

Table 2-108 describes the vector shift instructions.

Table 2-108. Vector Shift Instructions

Name	Mnemonic	Syntax
Vector Shift Left	**vsl**	**v**D,**v**A,**v**B
Vector Shift Right	**vsr**	**v**D,**v**A,**v**B
Vector Shift Left Double by Octet Immediate	**vsldoi**	**v**D,**v**A,**v**B,SH
Vector Shift Left by Octet	**vslo**	**v**D,**v**A,**v**B
Vector Shift Right by Octet	**vsro**	**v**D,**v**A,**v**B

2.6.5.8 Vector Status and Control Register Instructions

Table 2-109 summarizes the instructions for reading from or writing to the AltiVec status and control register (VSCR), described in Section 7.1.1.5, “Vector Save/Restore Register (VRSAVE).”

Table 2-109. Move To/From VSCR Register Instructions

Name	Mnemonic	Syntax
Move to AltiVec Status and Control Register	**mtvscr**	**v**B
Move from AltiVec Status and Control Register	**mfvscr**	**v**B

2.7 AltiVec VEA Instructions

The PowerPC virtual environment architecture (VEA) describes the semantics of the memory model that can be assumed by software processes, and includes descriptions of the cache model, cache-control instructions, address aliasing, and other related issues. Implementations that conform to the VEA also adhere to the UISA, but may not necessarily adhere to the OEA. For further details, see Chapter 4, “Addressing Mode and Instruction Set Summary,” in the *Programming Environments Manual*.

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

2.7.1 AltiVec Vector Memory Control Instructions

Memory control instructions include the following types:

- Cache management instructions (user-level and supervisor-level)
- Translation lookaside buffer (TLB) management instructions

This section briefly summarizes the user-level cache management instructions defined by the AltiVec VEA. See Chapter 3, “L1, L2, and L3 Cache Operation,” for more information about supervisor-level cache, segment register manipulation, and TLB management instructions.

The AltiVec architecture specifies the data stream touch instructions **dst(t)**, **dstst(t)**, and it specifies two data stream stop (**dss(all)**) instructions. The MPC7450 implements all of them. The term **dst***x* used below refers to all of the stream touch instructions.

The instructions summarized in this section provide user-level programs the ability to manage on-chip caches, see Chapter 3, “L1, L2, and L3 Cache Operation,” for more information about cache topics.

Bandwidth between the processor and memory is managed explicitly by the programmer through the use of cache management instructions. These instructions provide a way for software to communicate to the cache hardware how it should prefetch and prioritize the writeback of data. The principal instruction for this purpose is a software directed cache prefetch instruction called data stream touch (**dst**). Other related instructions are provided for complete control of the software directed cache prefetch mechanism.

Table 2-110 summarizes the directed prefetch cache instructions defined by the AltiVec VEA. Note that these instructions are accessible to user-level programs.

Table 2-110. AltiVec User-Level Cache Instructions

Name	Mnemonic	Syntax	Implementation Notes
Data Stream Touch (non-transient)	**dst**	**r**A,**r**B,STRM	—
Data Stream Touch Transient	**dstt**	**r**A,**r**B,STRM	Used for last access
Data Stream Touch for Store	**dstst**	**r**A,**r**B,STRM	Not recommended for use in MPC7450
Data Stream Touch for Store Transient	**dststt**	**r**A,**r**B,STRM	Not recommended for use in MPC7450
Data Stream Stop (one stream)	**dss**	STRM	—
Data Stream Stop All	**dssall**	STRM	—

For detailed information for how to use these instruction, see Section 7.1.2.3, “Data Stream Touch Instructions.”

2.7.2 AltiVec Instructions with Specific Implementations for the MPC7450

The AltiVec architecture specifies Load Vector Indexed LRU (**lvxl**) and Store Vector Indexed LRU (**stvxl**) instructions. The architecture suggests that these instructions differ from regular AltiVec load and store instructions in that they leave cache entries in a least recently used (LRU) state instead of a most recently used (MRU) state. This supports efficient processing of data which is known to have little reuse and poor caching characteristics. The MPC7450 implements these instructions as suggested. They follow all the cache allocation and replacement policies described in Section 3.5, “L1 Cache Operation,” but they leave their addressed cache entries in the LRU state. In addition, all LRU instructions are also interpreted to be transient and are also treated as described in Section 7.1.2.2, “Transient Instructions and Caches.”

Chapter 3
L1, L2, and L3 Cache Operation

The MPC7450 microprocessor contains separate 32-Kbyte, eight-way set-associative level 1 (L1) instruction and data caches to allow the execution units and registers rapid access to instructions and data. In addition, the MPC7450 microprocessor features an integrated 256-Kbyte level 2 (L2) cache (512-Kbyte L2 for the MPC7447 and MPC7457, 1-Mbyte for the MPC7448) and address tags and status bits for a level 3 (L3) cache that supports either 1 or 2 Mbytes of cache. Note that the L3 cache is not supported by the MPC7448, MPC7447A, MPC7447, MPC7445, or MPC7441.

This chapter describes the organization of the on-chip L1 instruction and data caches, cache coherency protocols, cache control instructions, various cache operations, the organization and features of the L2 cache, and a description of the L3 cache controller. It describes the interaction between the caches, the load/store unit (LSU), the instruction unit, and the memory subsystem. This chapter also describes the replacement algorithms used for each of the caches and the L3 private memory feature of the MPC7450.

Note that in this chapter, the term ‘multiprocessor’ is used in the context of maintaining cache coherency. These multiprocessor devices could be actual processors or other devices that can access system memory, maintain their own caches, and function as bus masters requiring cache coherency.

AltiVec Technology and the Cache Implementation

The implementation of AltiVec technology in the MPC7450 has implications that affect the cache model. They are as follows:

- AltiVec transient instructions (**dstt**, **dststt**, **lvxl**, and **stvxl**), described in Section 7.1.2.2, “Transient Instructions and Caches”
- AltiVec LRU instructions (**lvxl**, **stvxl**), described in Section 3.5.7.3, “AltiVec LRU Instruction Support”
- External system bus transactions caused by caching-inhibited AltiVec loads and stores or write-through AltiVec stores, as described in Section 3.8.1, “MPC7450 Caches and System Bus Transactions”

The MPC7448 adds support for out-of-order issue of AltiVec instructions. Instructions can be issued out-of-order from the bottom two VIQ entries (VIQ1–VIQ0). An instruction in VIQ1 does not have to wait for an instruction in VIQ0 that is waiting for operand availability.

3.1 Overview

The MPC7450 L1 cache implementation has the following characteristics:

- Two separate 32-Kbyte instruction and data caches (Harvard architecture)
- Eight-way set-associative caches
- Caches have 32-byte cache blocks. A cache block is the block of memory that a coherency state describes—it corresponds to a cache line for the L1 data cache.
- Cache directories are physically addressed. The physical (real) address tag is stored in the cache directory.
- Caches implement a pseudo least-recently-used (PLRU) replacement algorithm within each way.
- Cache write-back or write-through operation is programmable on a per-page or per-block basis.
- The instruction cache can provide four instructions per clock cycle; the data cache can provide four words per clock cycle.
  - Two-cycle latency and single-cycle throughput for instruction or data cache accesses
- Caches can be disabled in software
- Caches can be locked in software
- Supports a four-state modified/exclusive/shared/invalid (MESI) coherency protocol
  - A single coherency status bit for each instruction cache block allows encoding for the following two possible states:
    - Invalid (INV)
    - Valid (VAL)
  - Two status bits (MESI[0–1]) for each data cache block allow encoding for coherency, as follows:
    - 00 = invalid (I)
    - 01 = shared (S)
    - 10 = exclusive (E)
    - 11 = modified (M)
- Separate copy of data cache tags for efficient snooping
- Both the L1 caches support parity generation and checking (enabled through bits in the ICTRL register) as follows:
  - Instruction cache—one parity bit per instruction
  - Data cache—one parity bit per byte of data
- No snooping of instruction cache except for **icbi** instruction
- The caches implement a pseudo least-recently-used (PLRU) replacement algorithm within each way.
- Data cache supports AltiVec LRU and transient instructions, as described in Section 1.3.2.2, “AltiVec Instruction Set”

- Critical double- and/or quad-word forwarding is performed as needed. Critical quad-word forwarding is used for AltiVec loads and instruction fetches. Other accesses use critical double-word forwarding.
- Each cache can be invalidated or locked by setting the appropriate bits in the hardware implementation-dependent register 0 (HID0), a special-purpose register (SPR) that is implementation-specific.

The MPC7450 supports a fully-coherent 64-Gbyte physical memory address space (when extended addressing is enabled with HID0[XAEN] = 1). Bus snooping is used to ensure the coherency of global memory with respect to the data cache.

On an L1 data cache miss, cache blocks are filled in one 32-byte beat from the L2 cache, L3 cache, or system bus, and the critical data is forwarded immediately to the requesting execution unit (and register file). Load misses are processed as described in Section 3.1.2.4, “LSU Load Miss, Castout, and Push Queues,” providing for hits under misses.

The instruction cache is also filled in one 32-byte beat from the L2 cache, L3 cache, or system bus, and the critical quad word is simultaneously forwarded to the instruction queue, thus minimizing stalls due to cache fill latency. Note that if the instruction fetch is from cache-inhibited memory and the bus is operating in 60x bus mode, the bus access is still a 32-byte transaction, even though only the required 16 bytes are transmitted to the instruction queue. However, in MPX bus mode, a cache-inhibited instruction fetch performs a 16-byte transaction on the bus. The instruction cache is also not blocked to internal accesses while a cancelled instruction cache miss is outstanding, providing for hits under misses.

The instruction cache provides a 128-bit interface to the instruction unit, so up to four instructions can be made available to the instruction unit in a single clock cycle on an L1 instruction cache hit. The instruction unit accesses the instruction cache frequently in order to sustain the high throughput provided by the 12-entry instruction queue.

Additionally, the on-chip L2 cache has the following features:

- Integrated 256-Kbyte, eight-way set-associative unified instruction and data cache for the MPC7450 (512-Kbyte for the MPC7447 and MPC7457, 1-Mbyte for the MPC7448)
- Maintains instructions, data, or both instructions and data (selectable through L2CR)
- Fully pipelined to provide 32 bytes per clock cycle to the L1 caches. In the MPC7448, pipelined to provide 32 bytes per two clock cycles to the L1 caches.
- Total latency of 9 processor cycles for L1 data cache miss that hits in the L2 (in the MPC7448, 11 cycles with ECC disabled, 12 cycles with ECC enabled)
- Uses one of two random replacement algorithms (selectable through L2CR)
- Cache write-back or write-through operation programmable on a per-page or per-block basis
- Organized as 32 bytes/block and two blocks (sectors) /line (a cache block is the block of memory that a coherency state describes).
- In the MPC7448, error correction and detection using a SECDED (single-error correction, double-error detection) protocol. Every 64 bits of data comes with 8 bits of error detection/correction, which can be programmed as ECC (error correction code) across the 64 bits of data, byte parity, or no error detection/correction.

- Supports parity generation and checking for both tags and data (enabled through L2CR). In the MPC7448, tag parity is enabled separately in the L2ERRDIS register, and data parity can be enabled through L2CR only when ECC is disabled.
- Two status bits (MESI[0–1]) for each L2 cache block allow encoding for coherency, as follows:
  - 00 = invalid (I)
  - 01 = shared (S)
  - 10 = exclusive (E)
  - 11 = modified (M)
- Prefetching of the second (unrequired) block through up to three L2 prefetch engines enabled through MSSCR0
- In the MPC7448, error injection modes provided for testing

Finally, the L3 cache controller on the MPC7450 has the following features:

- Provides critical double-word forwarding to the requesting unit
- On-chip tags support 1 Mbyte or 2 Mbytes of external SRAM that is eight-way set-associative
- Maintains instructions, data, or both instructions and data (selectable through L3CR)
- Cache write-back or write-through operation programmable on a per-page or per-block basis
- Organized as 64 bytes/line configured as two blocks (sectors) with separate status bits per line for 1-Mbyte configuration
- Organized as 128 bytes/line configured as four blocks (sectors) with separate status bits per line for 2-Mbyte configuration
- 1 Mbyte or 2 Mbytes of the L3 SRAM can be designated as private memory.
- Supports same four-state (MESI) coherency protocol as L1 and L2 caches
- Supports parity generation and checking for both tags and data (enabled through L3CR)
- Same choice of two random replacement algorithms used by L2 cache (selectable through L3CR)
- Configurable core-to-L3 frequency divisors
- 64-bit external L3 data bus sustains 64 bits per L3 clock cycle
- Supports MSUG2 dual data rate (DDR) synchronous burst SRAMs, PB2 pipelined synchronous burst SRAMs, and pipelined (register-register) late-write synchronous burst SRAMs

Note that the L3 cache and the L3 cache interface are not supported on the MPC7441, MPC7445, MPC7447, MPC7447A, or MPC7448.

3.1.1 Block Diagram

The instruction and data caches, L2 cache, and L3 cache controller are integrated in the MPC7450 as shown in Figure 3-1.

Both L1 caches are tightly coupled with the MPC7450 L2 cache, L3 cache controller, and the memory subsystem to allow efficient access to the L2 cache, L3 cache, or the system interface and other bus masters. The memory subsystem receives requests for memory operations from the LSU (on behalf of the instruction and data caches) and provides queues for loading and storing from the caches.

The system interface performs external bus operations per the 60x or MPX bus protocol. Depending on the transaction type, the critical 8 bytes (for double words) or 16 bytes (for quad words) are forwarded to the requesting unit. Note that for instruction fetches, the critical quad word is always forwarded. Also, the system interface accumulates 64-bit data beats from the bus into a 32-byte entity before loading it into the L1, L2, and L3 caches. The system interface also captures snoop addresses for the L1 data cache, the L2 and L3 caches, and the memory reservation (**lwarx** and **stwcx.**) operations.

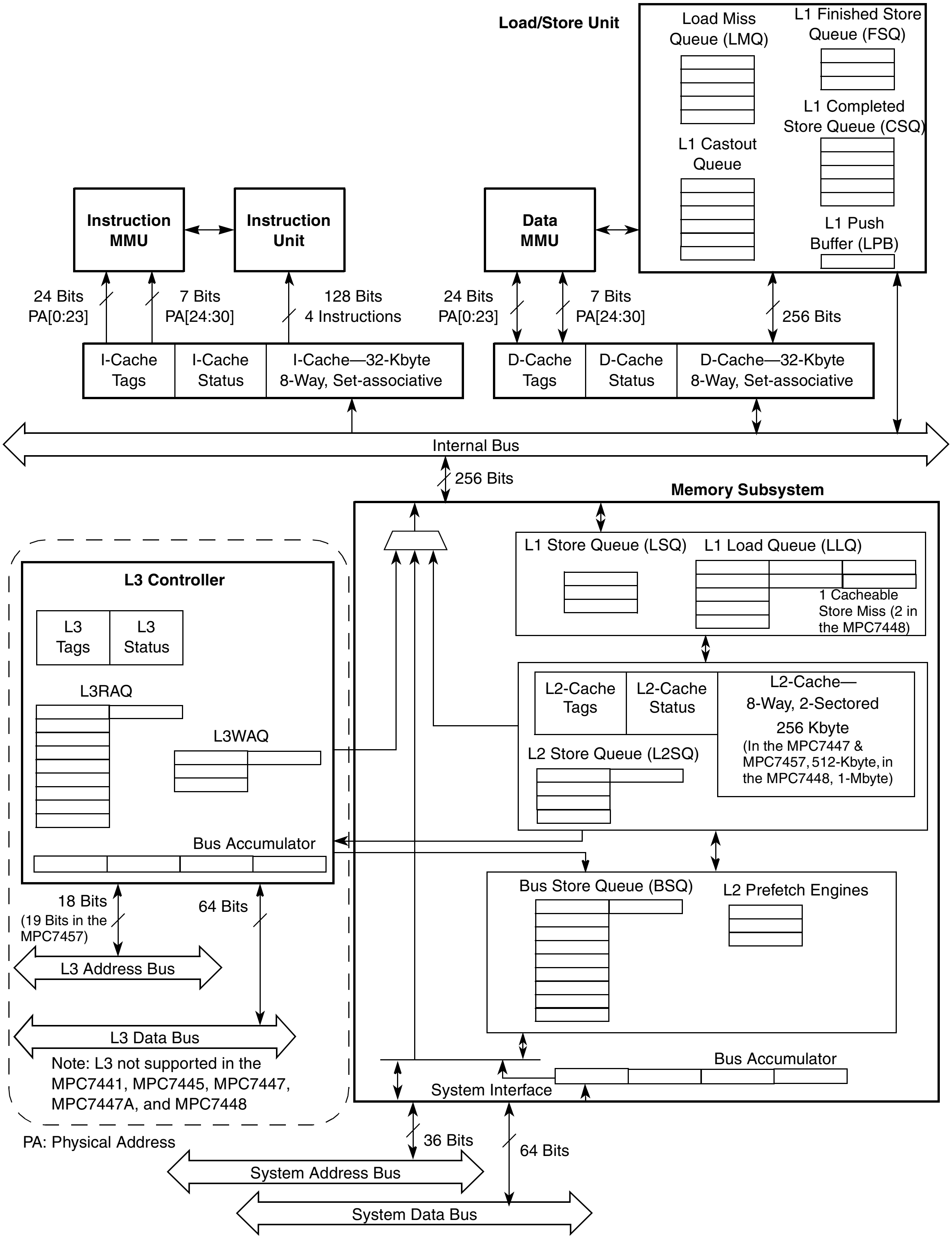


Figure 3-1. Cache/Memory Subsystem Integration

3.1.2 Load/Store Unit (LSU)

The data cache supplies data to the general-purpose registers (GPRs), floating-point registers (FPRs), and vector registers (VRs) by means of the load/store unit (LSU). The MPC7450 LSU is directly coupled to the data cache with a 32-byte interface (a cache line) to allow efficient movement of data to and from the GPRs, FPRs, and VRs. The LSU provides all the logic required to calculate effective addresses, handles data alignment to and from the data cache, and provides sequencing for load/store string and load/store multiple operations. Write operations to the data cache can be performed on a byte, half-word, word, double-word, or quad-word basis.

This section describes the LSU queues that support the L1 caches. See Section 3.3.3, “Load/Store Operations and Architecture Implications,” for more information on architectural coherency implications of load/store operations and the LSU on the MPC7450. Also, see Chapter 6, “Instruction Timing,” for more information on other aspects of the LSU and instruction scheduling considerations.

The vector touch engine (VTE) generates cache line fetch requests based on the contents of the **dst**, **dsts**, **dss**, and **dssall** instructions that are part of the AltiVec specification. These instructions are not disabled by the AltiVec enable bit in the MSR. See Chapter 7, “AltiVec Technology Implementation,” for more information on the VTE.

3.1.2.1 Cacheable Loads and LSU

When free of data dependencies cacheable loads execute in the LSU in a speculative manner with a maximum throughput of one per cycle and a 3-cycle latency for integer and vector loads. Note that floating-point loads have a 4-cycle latency through the LSU. Data returned from the cache is held in a rename buffer until the completion logic commits the value to the processor state.

3.1.2.2 LSU Store Queues

Stores cannot be executed speculatively. Stores must be held in the 3-entry finished store queue (FSQ), as shown in Figure 3-1, until the completion logic signals that the store instruction is to be committed. When the store is committed, it moves to the 5-entry committed store queue (CSQ). A store remains in the CSQ until the data cache is updated if the access is cacheable. If a store is cache-inhibited, the operation moves through the CSQ on to the rest of the memory subsystem.

To reduce the latency of loads dependent on stores, the MPC7450 supports data forwarding from any entry in the CSQ before the data is actually written to the cache. The addresses of subsequent loads are compared to all entries in the CSQ and, on a hit, use the data from the newest matching entry. If a load aliases to both a CSQ entry and an FSQ entry, the LSU pipeline stalls. The load needs the newest data from the FSQ and the data is not available until it is completed and moves to the CSQ. Note that no forwarding occurs from a **stwcx.** operation but forwarding does occur from store operations caused by **dcbz** instructions.

3.1.2.3 Store Gathering/Merging

To increase external bandwidth to frame buffers and I/O devices, the MPC7450 performs store gathering of unguarded write-through stores or cache-inhibited stores. Two of these store operations are gathered in the CSQ if the following requirements are met:

- Entry CSQ0 is currently accessing the memory subsystem (that is, it missed in the data cache).
- The stores are bytes, half words, words, double words or quad words (and are the same size).
- The stores are adjacent or overlapping in address (words in the same double word, double words in the same quad word, or quad words in the same cache line).
- The stores are adjacent in the CSQ.
- Both stores are aligned.
- The system bus is operating in MPX bus mode, or the stores are words or smaller.

The same store-gathering mechanism is used to gather cacheable write-back stores. In this case, these stores can be gathered anywhere within the same cache line if they have not yet accessed the cache. Also, these stores do not need to be of the same size.

Not all stores are gathered. In particular, when there is a series of stores, the first store often appears to the memory subsystem as ungathered.

Store gathering and store merging are enabled through HID0[SGE]. Note that in addition to the clearing of SGE, the **eieio** instruction may also be used to keep stores from being gathered. If an **eieio** instruction is detected in the store queues, store gathering is not performed. If HID1[SYNCBE] = 1, the **eieio** instruction also causes a system bus broadcast operation, which may be used to prevent external devices, such as a bus bridge chip, from gathering stores. See Section 3.3.3.3, “Load Ordering with Respect to Other Loads,” for more information on the effects of **eieio**

If multiple cacheable stores are gathered such that the result is one 32-byte store, the processor issues a single line kill block transfer instead of the store.

3.1.2.4 LSU Load Miss, Castout, and Push Queues

The LSU requests cache blocks that miss in the L1 data cache from the next levels of memory. In the case of a cache miss for a load, the load is placed in the 5-entry load miss queue (LMQ) until it can be serviced to allow for subsequent loads to continue to propagate through the LSU.

The LSU also maintains a 6-entry L1 castout queue (LCQ) as a place-holder for data cache castouts caused by the PLRU replacement algorithm until they can be serviced. Note that castouts are only selected (by the replacement algorithm) when the new cache line is ready to be loaded into the L1. Because all L1 data cache misses can potentially require a castout, misses do not access the L2, L3, or system bus until a slot is available in the LCQ for the potential castout operation.

Finally, the LSU also maintains a 1-entry push buffer (LPB) for holding a cache push operation caused by a snoop hit of modified data in the L1 data cache until it can complete. Note that all entries in the LCQ and LPB are snooped when other masters are accessing the MPC7450 bus.

3.1.3 Memory Subsystem Blocks

As shown in Figure 3-1, the memory subsystem interfaces to the L1 instruction and data caches and the LSU with a 256-bit internal bus. The four major logic blocks are described in the following subsections. Conceptually, the general flow for transactions through the memory subsystem can be considered to be from the L1 service queues, to the L2 cache, to the L3 interface, to the bus service queues, noting that data from the bus can flow directly from the bus accumulator at the external system interface to the 256-bit internal bus (loading the L2 and L3 caches in parallel). Exceptions to this are noted in the following subsections.

Note that transactions on the external bus performed by alternate masters are snooped by all relevant entities in the MPC7450. Thus the L1 data cache, the LSU queues, memory subsystem queues, and the L2 and L3 caches are all checked for a snoop hit. When a snoop hit occurs and a push is required, the MPC7450 retries the bus transaction and performs the push operation or performs data intervention (if the bus is operating in MPX bus mode and MSSCR0[EIDIS] = 0).

3.1.3.1 L1 Service Queues

Separate from the LMQ and the two store queues of the LSU, the memory subsystem block maintains two additional queues for handling L1 misses. The L1 load queue (LLQ) of the memory subsystem contains a total of eight entries (nine in the MPC7448). They are as follows:

- Five for load misses, including those generated by **dcbt**, **dcbtst**, **dst**, **dsts**, and **eciwx** instructions
- Two for instruction fetches
- One for a cacheable store request that is marked as write-back (W = 0), which requires a read-with-intent-to-modify load transaction on the bus, or for loads generated by **dcba** and **dcbz**. The MPC7448 has support for a second cacheable store request.

For efficiency, these accesses are simultaneously sent to the L2 and L3 caches from the LLQ and reside in the LLQ until the data has been loaded. If the access requires a system interface transaction (based on the L2 and L3 responses), the LLQ causes that bus transaction to occur. If the access is non-transient and misses in all three caches, all three caches (if enabled) are loaded with the missed data when it is read from the bus.

Also, separate from the L1 store queues of the LSU, the memory subsystem has an L1 store queue (LSQ) that maintains three entries waiting to be written to the L2 cache (if enabled). The three entries are dedicated as follows:

- One for
  - Stores, including caching-inhibited and write-through stores,
  - Memory management instructions, and
  - **sync**, cache control, and memory synchronization instructions
- One for castouts
- One for snoop pushes

All accesses that go through the LSQ also propagate to the L3 cache and the system bus, except for L1 castouts that are caused by a replacement operation due to a reload, and result in a hit in the L2.

Thus, note that castouts caused by the **dcbf** instruction do propagate to the L3 cache and the system bus.

3.1.3.2 L2 Cache Block

The integrated L2 cache on the MPC7450 is a unified (possibly containing instructions and data), 256-Kbyte on-chip cache. For the MPC7447 and MPC7457, the L2 cache is 512 Kbytes. On the MPC7448, the L2 cache is 1 Mbyte. It is eight-way set-associative and organized with 32-byte blocks and two blocks per line as shown in Figure 3-17. Thus each line shares the same tag, but the MESI bits are independently maintained for each block.

When the L2 and corresponding L1 cache are enabled, load and store entries from the LLQ and LSQ propagate to the L2 cache, provided caching is allowed (the I bit of WIMG for that particular access is cleared). The L2 services accesses from the LLQ and LSQ with a 3-cycle total latency (in the MPC7448, 5 cycles with ECC disabled, 6 cycles with ECC enabled) and a maximum throughput of one L2 access per clock cycle (one access every other cycle in the MPC7448). In the MPC7448 has a 2-cycle throughput due to the increased size of the L2 cache.

As described in Section 3.1.3.1, “L1 Service Queues,” LLQ accesses are simultaneously sent to the L2 and L3 caches. LSQ accesses serviced by the L2 that need service by the L3 propagate to the L2 store queue (L2SQ) for service by the L3 cache. The L2SQ has a total of 5 entries as follows:

- Four entries for L2 castouts (or stores)
- One entry for pushes and interventions caused by snoop hits

For more detailed information about the functions of the L2 cache, see Section 3.6, “L2 Cache.”

3.1.3.3 System Interface Block

As described in Section 3.1.3.1, “L1 Service Queues,” the LLQ can cause bus transactions to occur. In addition, the system interface block of the memory subsystem maintains the following two entities that can cause external bus transactions:

- Bus store queue (BSQ)—After the L2 and L3 caches respond to an access and the access generates a castout (or write-through store) or a push operation, it is sent to the BSQ for service by the system interface. The BSQ maintains up to nine outstanding castout operations and one push operation.
- L2 prefetch engines—When only one block of an L2 cache line is valid (due to an L2 reload caused by a read miss in the L1, L2, and L3 caches), the L2 prefetch engines can initiate an external bus transaction to fill the second block of that L2 cache line. Up to 3 separate outstanding L2 prefetches can be enabled. See Section 3.6.3.2, “L2 Prefetch Engines and MSSCR0,” for more detailed information about the L2 prefetch engines. Note that these prefetch engines only fetch from the system bus and do not fetch from the L3 cache.

Also, the system interface block maintains a bus accumulator that collects four double words (instructions or data) from the system interface for forwarding to the internal bus on reads.

3.1.4 L3 Cache Controller Block

The L3 cache controller maintains the tags and status for the 1- or 2-Mbyte L3 cache. The L3 cache is also a unified (possibly containing instructions and data) cache that is 8-way set-associative and organized with 32-byte blocks and two blocks per line (1 Mbyte) or four blocks per line (2 Mbyte). Each line shares the same tag, but the MESI bits are independently maintained for each block. Note that the L3 cache is not supported by the MPC7441, MPC7445, MPC7447, MPC7447A, or MPC7448.

The L3 cache controller also has queues that serve as staging areas for pending SRAM read and write transactions. There is an L3RAQ that has a total of ten entries that are dedicated as follows:

- Nine entries for pending SRAM reads, including loads and castouts
- One entry for pending snoop pushes

Note that if the L3RAQ is full, the LLQ may stall.

Also, there is an L3WAQ that has a total of 4 entries that are dedicated as follows:

- Three entries for pending SRAM writes, including L2 castouts
- One entry for L3 reloads

Note that if the L3WAQ is full, the L2SQ may stall and L3 reloads may be dropped.

In the same way that LSQ entries, after they having been serviced by the L2 cache, propagate to the L2 store queue (L2SQ) for service by the L3 cache, the BSQ serves as a staging area for data being transferred between the L3 cache and the system interface. Also, the L3 cache controller block maintains a bus accumulator that collects four double words (instructions or data) from the L3 interface for forwarding to the memory subsystem block. Note that the L3 cache can also be configured to be used as private memory. For more detailed information about the functions of the L3 cache controller, see Section 3.7, “L3 Cache Interface.”

3.2 L1 Cache Organizations

The L1 instruction and data caches of the MPC7450 are both organized as 128 sets of eight blocks with 32 bytes in each cache line. The following subsections describe the differences in the organization of the instruction and data caches. For information on L2 and L3 cache operation, see Section 3.6, “L2 Cache,” and Section 3.7, “L3 Cache Interface.”

3.2.1 L1 Data Cache Organization

The L1 data cache is organized as shown in Figure 3-2.

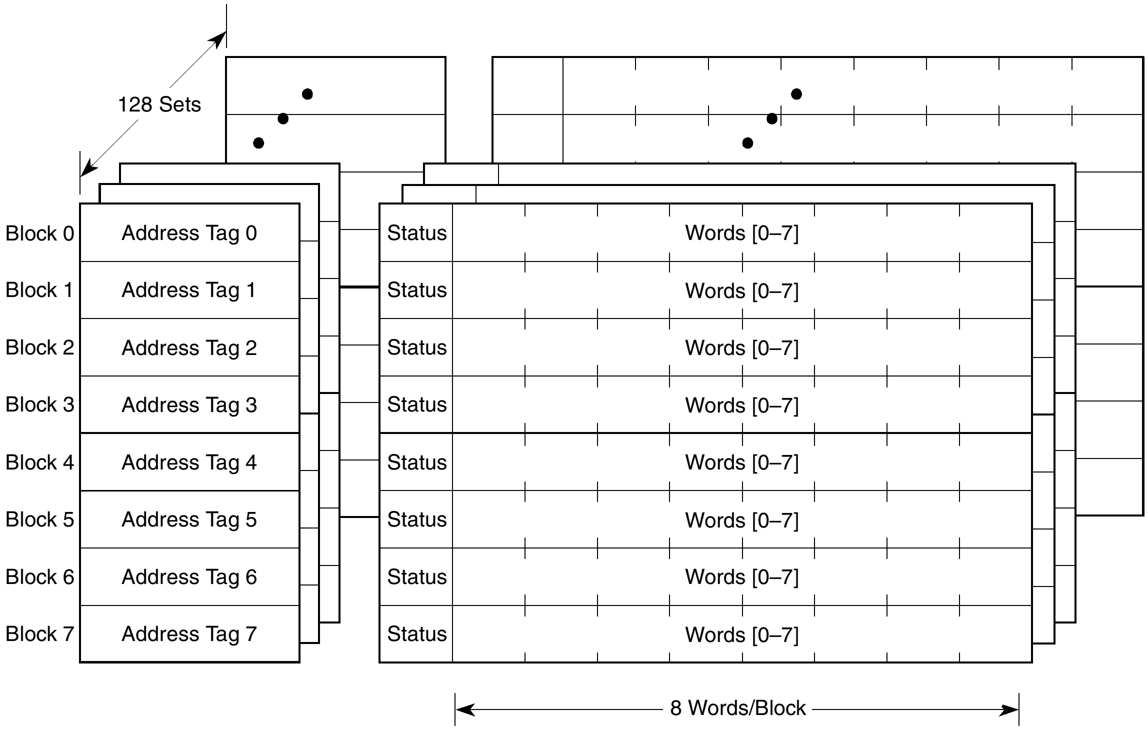


Figure 3-2. L1 Data Cache Organization

Each block consists of 32 bytes of data, three status bits, and an address tag. Note that in the PowerPC architecture, the term ‘cache block,’ or simply ‘block,’ when used in the context of cache implementations, refers to the unit of memory at which coherency is maintained. For the MPC7450 L1 data cache, this is the 32-byte cache line. This value may be different for other implementations using the PowerPC architecture. Also, although it is not shown in Figure 3-2, the data cache has one parity bit/byte (four parity bits/word).

Each cache block contains eight contiguous words from memory that are loaded from an eight-word boundary (that is, bits PA[31:35] of the physical addresses are zero); as a result, cache blocks are aligned with page boundaries. Address bits PA[24:30] provide the index to select a cache set. The tags consist of physical address bits PA[0:23]. Address translation occurs in parallel with set selection (from PA[24:30]). Lower address bits PA[31:35] locate a byte within the selected block. All of these address ranges are shown for 36-bit physical addressing (enabled when HID0[XAEN] = 1). When 32-bit addressing is used (HID0[XAEN] = 0), all of these physical address bits are shifted down by 4, and the tags consist of physical address bits PA[0:19].

The data cache tags are dual-ported and non-blocking for efficient load/store and snooping operations. Thus the data cache can be accessed internally while a load for a miss is pending (allowing hits under misses). When the load miss is actually updating the cache, subsequent loads are blocked for two cycles and stores are blocked for one cycle (but the data for the load miss can be forwarded to the execution unit simultaneously). The LMQ allows misses under misses to occur.

There are three status bits associated with each cache block. These bits are used to implement the modified/exclusive/shared/invalid (MESI) cache coherency protocol. The coherency protocols are described in Section 3.3, “Memory and Cache Coherency.”

3.2.2 L1 Instruction Cache Organization

The L1 instruction cache is organized as shown in Figure 3-3.

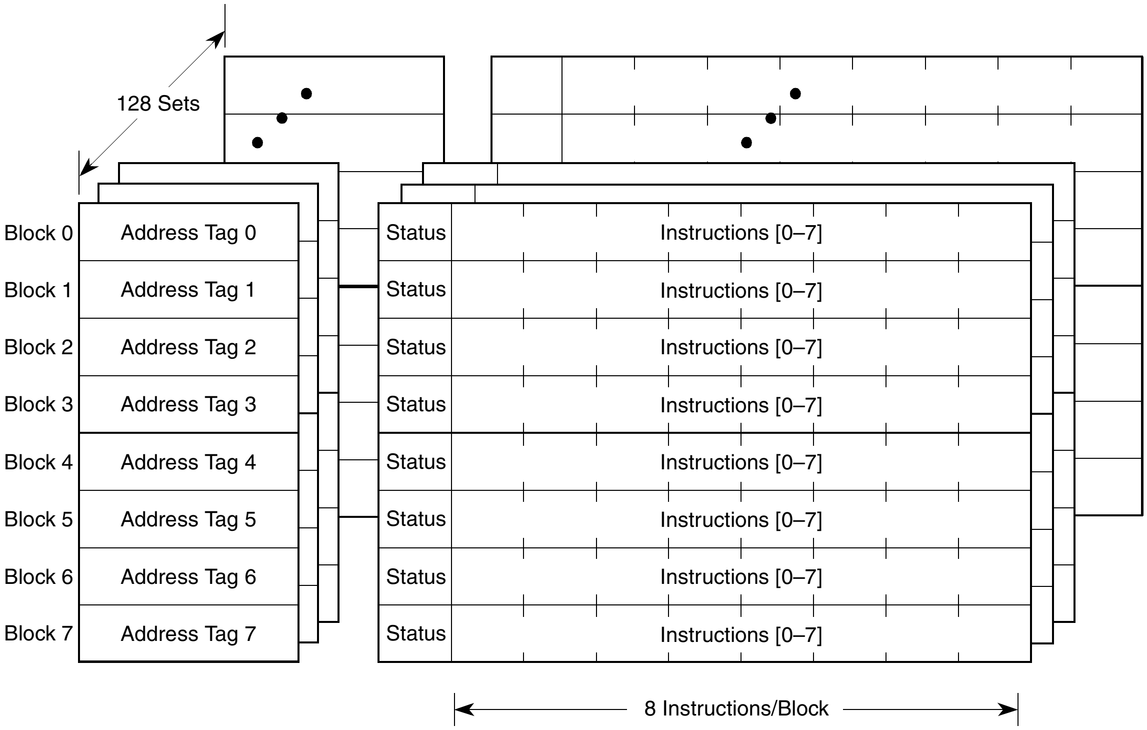


Figure 3-3. L1 Instruction Cache Organization

Each block consists of 8 instructions, a single status bit, and an address tag. As with the data cache, each instruction cache block is loaded from an eight-word boundary (that is, bits PA[31:35] of the physical addresses are zero); as a result, cache blocks are aligned with page boundaries. Also, address bits PA[24:30] provide the index to select a set, and bits PA[31:33] select an instruction within a block. The tags consist of physical address bits PA[0:23]. Address translation occurs in parallel with set selection (from PA[24:30]). All of these address ranges are shown for 36-bit physical addressing (enabled when HID0[XAEN] = 1). When 32-bit addressing is used (HID0[XAEN] = 0), all of these physical address bits are shifted down by 4, and the tags consist of physical address bits PA[0:19].

The instruction cache is also non-blocking in that it can be accessed internally while a fill for a miss is pending (allowing hits under misses). In addition, subsequent misses can also be sent to the memory subsystem before the original miss is serviced (allowing misses under misses). When a miss is actually updating the cache, subsequent accesses are blocked for one cycle (but the instruction that missed can be forwarded to the instruction unit simultaneously).

The instruction cache differs from the data cache in that it does not implement a multiple state cache coherency protocol. A single status bit indicates whether a cache block is valid or invalid. The instruction cache is not snooped, so if a processor modifies a memory location that may be contained in the instruction cache, software must ensure that such memory updates are visible to the instruction fetching mechanism. This can be achieved with the following instruction sequence (using either **dcbst** or **dcbf**):

```
dcbst (or dcbf)|update memory
sync           |wait for update
icbi           |remove (invalidate) copy in instruction cache
sync           |ensure that ICBI invalidate at the instruction cache has completed
isync          |remove copy in own instruction buffer
```

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

Although it is not shown in Figure 3-3, the instruction cache has one parity bit/word.

3.3 Memory and Cache Coherency

The primary objective of a coherent memory system is to provide the same image of memory to all devices using the system. Coherency allows synchronization and cooperative use of shared resources. Otherwise, multiple copies of a memory location, some containing stale values, could exist in a system resulting in errors when the stale values are used. Each potential bus master must follow rules for managing the state of its cache. This section describes the coherency mechanisms of the PowerPC architecture and the cache coherency protocols that the MPC7450 caches support.

Unless specifically noted, the discussion of coherency in this section applies to the L1 data cache and the L2 and L3 caches. The instruction cache is not snooped. Instruction cache coherency must be maintained by software. However, the MPC7450 does support a fast instruction cache invalidate capability as described in Section 3.4.1.5, "L1 Instruction and Data Cache Flash Invalidation." Also, the flushing of self-modifying code from the data cache (and L2 and L3) is described in Section 3.4.4.8, "Instruction Cache Block Invalidate (icbi)."

3.3.1 Memory/Cache Access Attributes (WIMG Bits)

Some memory characteristics can be set on either a memory management block or page basis by using the WIMG bits in the BAT registers or page table entries (PTE), respectively. These bits allow both uniprocessor and multiprocessor system designs to exploit numerous system-level performance optimizations. The WIMG attributes control the following functionalities:

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

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 loads and instruction fetches from being performed until they are guaranteed to be required by the sequential execution model.

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.

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 non-global).

3.3.1.1 Coherency Paradoxes and WIMG

Care must be taken with respect to the use of the WIMG bits if coherent memory support is desired. Careless specification of these bits may create situations that present coherency paradoxes to the processor. These coherency paradoxes can occur within a single processor or across several processors. It is important to note that, in the presence of a paradox, the operating system software is responsible for correctness.

In particular, a coherency paradox can occur when the state of these bits is changed without appropriate precautions (such as flushing the pages that correspond to the changed bits from the caches of all processors in the system) or when the address translations of aliased real addresses specify different values for certain WIMG bit values. The MPC7450 supports aliasing for WIMG = 100x and WIMG = 000x; however, the MPC7450 does not support aliasing WIMG = 101x and WIMG = 001x. Specifically, this means that for a given physical address, the MPC7450 only supports simultaneous memory/cache access attributes for that physical address of write-through, caching-allowed, memory-coherency-not-required (WIMG = 100x) and write-back, caching-allowed, memory-coherency-not-required (WIMG = 000x).

For real addressing mode (that is, for accesses performed with address translation disabled—MSR[IR] = 0 or MSR[DR] = 0 for instruction or data access, respectively), the WIMG bits are automatically generated as 0b0011 (all memory is write-back, caching-allowed, memory-coherency-required, and guarded).

3.3.1.2 Out-of-Order Accesses to Guarded Memory

On the MPC7450, instructions are not fetched from guarded memory when instruction translation is enabled (MSR[IR] = 1). If an attempt is made to fetch instructions from guarded memory when MSR[IR] = 1, an ISI exception is taken.

The MPC7450 only fetches instructions out-of-order with respect to other instructions fetches from guarded memory when MSR[IR] = 0 and one of the following conditions applies:

- The instruction is in the instruction cache.
- The instruction resides in the same physical page as an instruction that is required by the execution model.
- The instruction resides in the next sequential physical page as an instruction that is required by the execution model.

Note that the MPC7450 can have two instruction fetches outstanding at any time.

The MPC7450 does not perform stores until they are required by the sequential execution model, independent of the setting of the G bit. The only effect of the G bit on stores is that the MPC7450 guarantees that stores to guarded (G = 1) and caching-inhibited (I = 1) memory are not store-gathered. (See Section 3.1.2.3, “Store Gathering/Merging,” for more information on store gathering.)

However, setting the G bit prevents a load from accessing the system interface until it is guaranteed to be required by the sequential execution model. Loads from guarded memory may be accessed out-of-order with respect to other loads from guarded memory if one of the following applies:

- The target location is valid in the data cache.
- The load is guaranteed to be executed. In this case, the entire cache block containing the referenced data may be loaded into the cache.

Note that instruction fetches and loads may also be prevented from accessing the system interface until they are guaranteed to be required by the sequential execution model by setting the speculative access disable bit, HID0[SPD]. Also note that setting HID0[SPD] does not prevent loads from bypassing stores. See Section 3.3.3.5, “Enforcing Store Ordering with Respect to Loads,” for more information.

For the MPC7450, a guarded load is not allowed to access the system interface until that load is at the bottom of the completion buffer. This means that all prior load accesses to the system interface must have already returned data to the processor before the subsequent guarded load is allowed to access the system address bus. This prevents the MPC7450 from pipelining a guarded load with any other type of load on the system interface. Note that this has a large negative effect on load miss bandwidth performance. For this reason, it is not recommended to have guarded loads in code streams that require high system bandwidth utilization.

3.3.2 Coherency Support

The MPC7450 provides full hardware support for PowerPC cache coherency and ordering instructions (**dcbz**, **dcbi**, **dcbf**, **sync**, **icbi**, and **eieio**) and full hardware implementation of the TLB management instructions (**tlbie**, and **tlbsync**). Snooping, described in Section 3.8.4, “Snooping of External Transactions,” is integral to the memory subsystem design and operation. The MPC7450 is self-snooping and can $\overline{\text{ARTRY}}$ its own **stwcx.** broadcasts.

Each 32-byte cache block in the data cache contains two status bits. The MPC7450 uses these bits to support the coherency protocols and to direct reload operations. The L1 data cache status bits and the conditions that cause them to be set or cleared are defined in Table 3-1. Note that analogous status bits are also used in the L2 and L3 caches.

Table 3-1. Data Cache Status Bits

MESI [0–1]	Name	Meaning	Set Conditions	Clear Conditions
11	Modified (M)	The cache block is modified with respect to the external system interface	• Store miss reload from bus, L2 or L3 cache • Write-back store hit on ¬S	Snoop hit
10	Exclusive (E)	The cache block is valid	Reload from bus, L2 or L3 cache	• **dcbi**, **dcbf**, and **dcbst** hit • Write-back store hit to S (see Section 3.5.5, "Store Hit to a Data Cache Block Marked Shared") • Snoop clean hit • Snoop invalidate hit
01	Shared (S)	The cache block is shared with other processors and is read-only	• Load miss reload from bus with $\overline{\text{SHD}}$ response • Load miss reload from L2 cache with L2 cache status = S • Load miss reload from L3 cache with L3 cache status = S	None
00	Invalid (I)	—	—	—

Every data cache block state is defined by its MESI status bits. Note that in a multiprocessor system, a cache line can exist in the exclusive state in at most one L1 data cache at any one time.

3.3.2.1 Coherency Between L1, L2, and L3 Caches

The MPC7450 allows for the L1 data, L2, and L3 caches to have different coherency status for the same cache block. A cache block in the L2 and/or L3 cache is allowed to be shared when the same block in the L1 is exclusive or modified. Additionally, an L2 block can be shared when the corresponding L3 block is exclusive or modified (or vice versa). The true coherency state of a cache block within the MPC7450 is determined by analyzing all three levels of the cache hierarchy.

3.3.2.1.1 Cache Closer to Core with Modified Data

A cache block can be in the shared, exclusive, or modified state in the L2 or L3, while a cache closer to the processor core has the block in the modified state. In this case the cache closer to the core may have newer data. So by definition, if a cache block is in the shared, exclusive, or modified state in the L1, L2, or L3, it has the newest data if no cache closer to the processor core has the block in the modified state.

If a cache block is in the modified state in the L2 or L3 and that block is modified in a cache closer to the processor core, the L2 and L3 may castout out-of-date data to memory. In this case, the newest data still exists in the cache closer to the processor core.

3.3.2.1.2 Transient Data and Different Coherency States

In the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448 the allowance of different cache states between the L1 and L2 caches eliminates the need to allocate or update the state in the L2 when a transient (**dststt** or **stvxl**) store occurs to a block that is marked as shared in the L2. In this case, the LLQ treats the L2 block as invalid for stores.

3.3.2.2 Snoop Response

Table 3-2 describes the snoop responses used by the MPC7450 and defines the symbols used in Section 3.3.2.5, "MESI State Transitions." See Chapter 8, "Signal Descriptions," and Chapter 9, "System Interface Operation," for detailed signal timing and bus protocol information.

Table 3-2. Snoop Response Summary

Snoop Response	State Transition Diagram Symbol	Description
No response	— (No symbol)	The processor does not contain any memory at the snooped address or the coherency protocol does not require a response. The snoop has been fully serviced and no internal pipeline collisions occurred that would require a busy response.
$\overline{\text{SHD}}$ asserted	S	The processor contains data from the snooped address or a reservation on the snooped address.
$\overline{\text{ARTRY}}$ asserted	A	The processor cannot service the snoop due to an internal pipeline collision (busy). The same address tenure must be rerun at a later time.
$\overline{\text{ARTRY}}$ followed by $\overline{\text{BR}}$ asserted	AS	The processor contains a modified copy of data from the snooped address and is prepared to perform a window-of-opportunity (W) snoop push. The same address tenure must be rerun at a later time.
$\overline{\text{HIT}}$ asserted for one cycle	H1 (MPX bus mode only)	The processor contains a modified copy of data from the snooped address and is prepared to perform cache-to-cache or window-of-opportunity (C or W) intervention.

3.3.2.3 Intervention

Table 3-3 briefly describes the intervention types used by the MPC7450. See Chapter 9, "System Interface Operation," for signaling protocol information for each intervention type.

Table 3-3. Snoop Intervention Summary

Intervention Type	State Transition Diagram Symbol	Description
No intervention	— (No symbol)	The processor does not contain any memory at the snooped address or the coherency protocol does not require intervention.
Window-of-opportunity	W	Window-of-opportunity snoop push for hits on modified data. The processor performs a write-with-kill, snoop-push transaction in the next address tenure. The MPC7450 asserts $\overline{\text{BR}}$ in the window of opportunity to initiate the snoop push operation. The window of opportunity is defined as the second cycle after an $\overline{\text{AACK}}$ that has been $\overline{\text{ARTRY}}$ed. Only the intervening master can assert $\overline{\text{BR}}$ in the window of opportunity. When a master asserts $\overline{\text{BR}}$ in the window of opportunity, it uses it to perform a snoop push (write-with-kill) to the most previous snoop address (unless the master still has a write-with-kill pending due to a previous window-of-opportunity request that is not yet satisfied). The MPC7450 always presents a cache-block aligned address (that is, A[31:35] = 0b0_0000) for every window-of-opportunity snoop push.
Cache-to-cache	C (MPX bus mode only)	Cache-to-cache intervention for hits on modified data. The processor has queued up a data-only write transaction to provide data to the snooping master (cache-to-cache intervention). If another master asserts $\overline{\text{ARTRY}}$ coincident with the assertion of $\overline{\text{HIT}}$, the MPC7450 cancels the queued-up data-only transaction but does not attempt to perform a window-of-opportunity snoop push. The cache block state is already changed to the new state due to the snoop. Thus, the intervening processor (the one that asserted $\overline{\text{HIT}}$) does not contain the cache block in a state suitable for intervention when the retried snoop transaction is rerun on the bus. However, it can perform a window-of-opportunity snoop push when the retried snoop transaction is rerun.

3.3.2.4 Simplified Transaction Types

For the purposes of snooping bus transactions, the MPC7450 treats related (but distinct) transaction types as a single simplified transaction type. Table 3-4 defines the mapping of simplified transaction types to actual transaction types.

Table 3-4. Simplified Transaction Types

Simplified Transaction Type	Actual Transaction Type
Read	Read Read-atomic
RWITM	RWITM (read-with-intent-to-modify) RWITM-atomic RCLAIM (read-claim)
RWNITC	RWNITC (read-with-no-intent-to-cache)—Acts like a read transaction for snoop response purposes; acts like a clean transaction for MESI state change purposes.
Write	Write-with-flush Write-with-flush-atomic

Table 3-4. Simplified Transaction Types (continued)

Simplified Transaction Type	Actual Transaction Type
Flush	Flush
Clean	Clean
Kill	Kill Write-with-kill
Reskill (Used for reservation snooping only)	RWITM RWITM-atomic RCLAIM Write-with-flush Write-with-flush-atomic Kill Write-with-kill

In the following state transition diagrams, RWNITC is not explicitly shown. For state transitions (for example, modified to exclusive), the MPC7450 treats RWNITC like a clean operation. For intervention purposes (for example a W or C intervention as defined in Table 3-3), the MPC7450 treats RWNITC like a read operation.

3.3.2.5 MESI State Transitions

The state diagrams in this section use symbols on the transition lines for snoop response and intervention type. For example, H1-C denotes a $\overline{\text{HIT}}$-asserted snoop response and a cache-to-cache intervention type. See Table 3-2 and Table 3-3 for the symbols used in the state diagrams.

3.3.2.5.1 MESI Protocol in MPX Bus Mode with Data Intervention Enabled

The following state diagrams (Figure 3-4, Figure 3-5, Figure 3-6, Figure 3-7, and Figure 3-8) show the MESI state transitions when the MPC7450 is configured for MPX bus mode with modified data intervention enabled (MSSCR0[EIDIS] = 0).

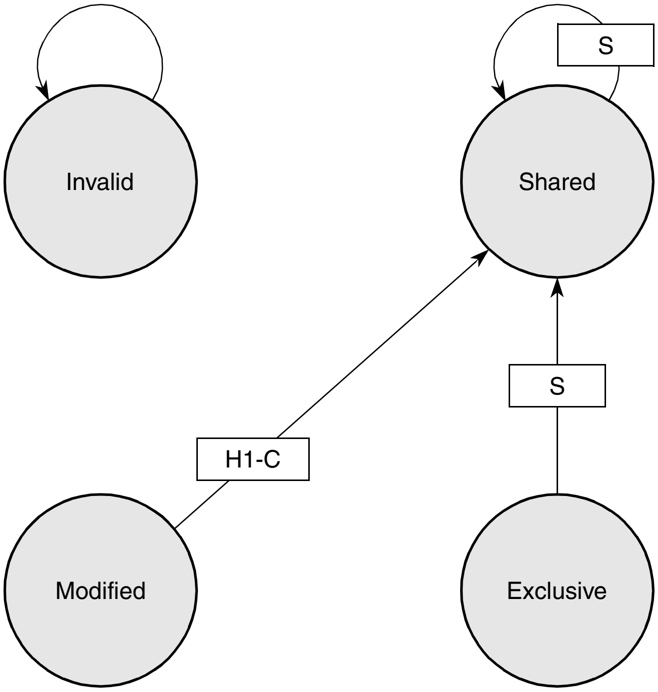


Figure 3-4. Read Transaction—MPX Bus Mode, MSSCR0[EIDIS] = 0

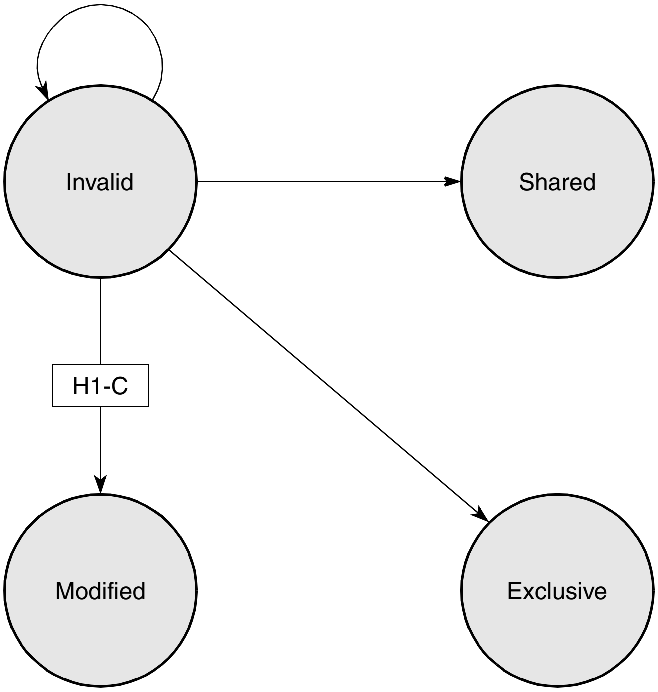


Figure 3-5. RWITM and Flush Transactions—MPX Bus Mode, MSSCR0[EIDIS] = 0

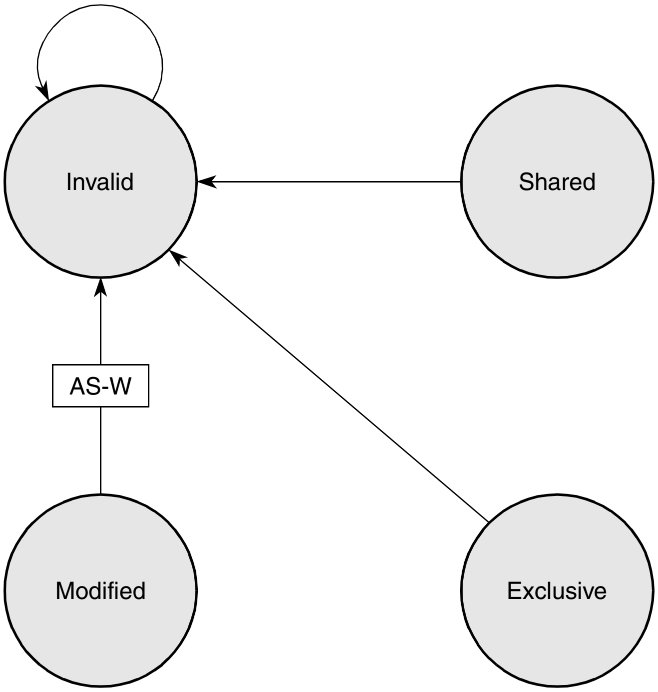


Figure 3-6. Write Transaction—MPX Bus Mode, MSSCR0[EIDIS] = 0

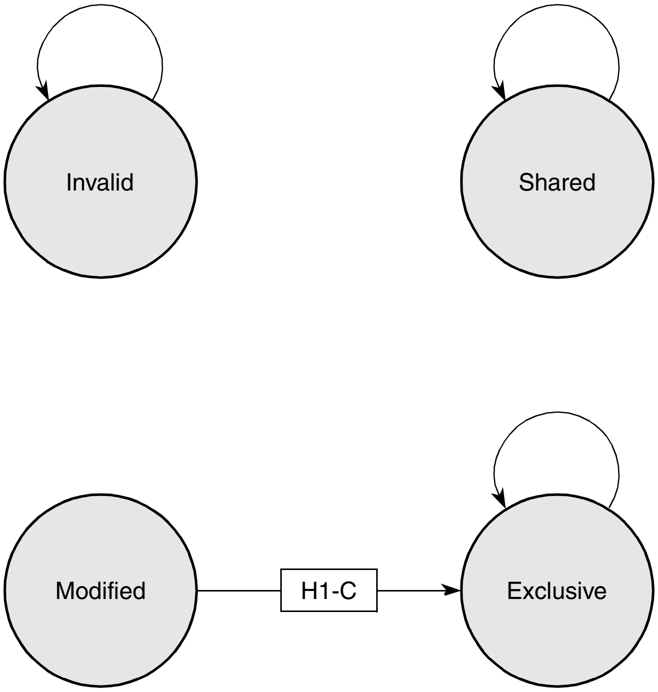
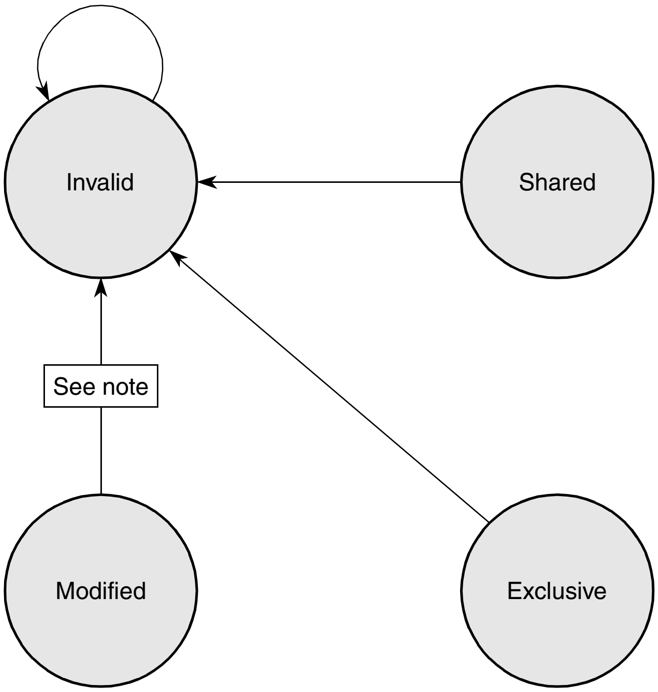


Figure 3-7. Clean Transaction—MPX Bus Mode, MSSCR0[EIDIS] = 0



Note: If another master asserts $\overline{\text{ARTRY}}$, the MPC7450 performs a window-of-opportunity style push. Otherwise, there is no intervention.

Figure 3-8. Kill Transaction—MPX Bus Mode, MSSCR0[EIDIS] = 0

3.3.2.5.2 MESI Protocol in 60x Bus Mode and MPX Bus Mode (with Intervention Disabled)

The following state diagrams (Figure 3-9, Figure 3-10, Figure 3-11, and Figure 3-12) show the MESI state transitions when the MPC7450 is configured for 60x bus mode and for MPX bus mode when hit intervention is disabled (MSSCR0[EIDIS] = 1).

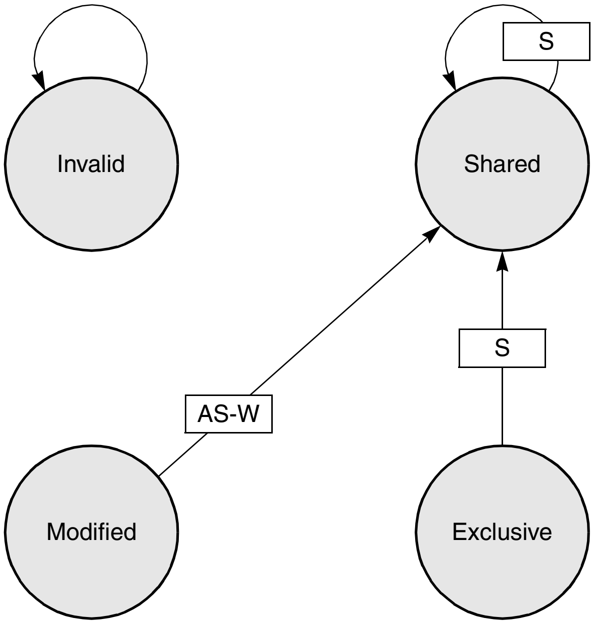


Figure 3-9. Read Transaction—60x and MPX Bus Modes, MSSCR0[EIDIS] = 1

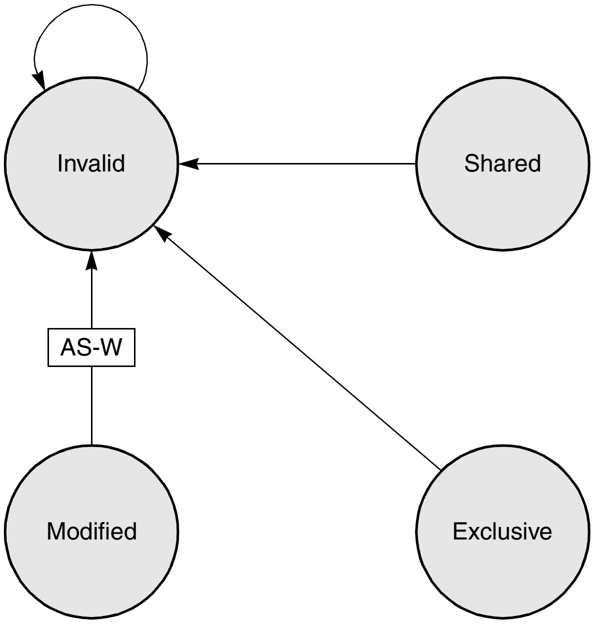


Figure 3-10. RWITM, Write, and Flush Transactions—60x and MPX Bus Modes, MSSCR0[EIDIS] = 1

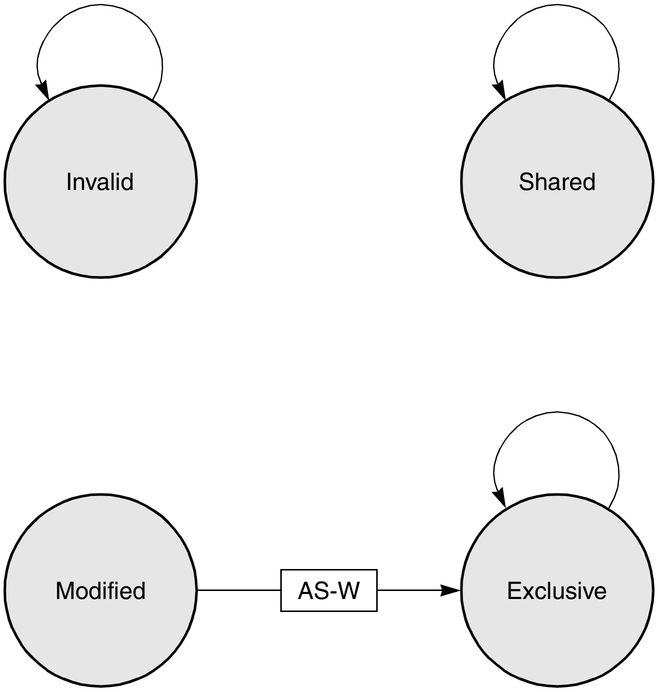
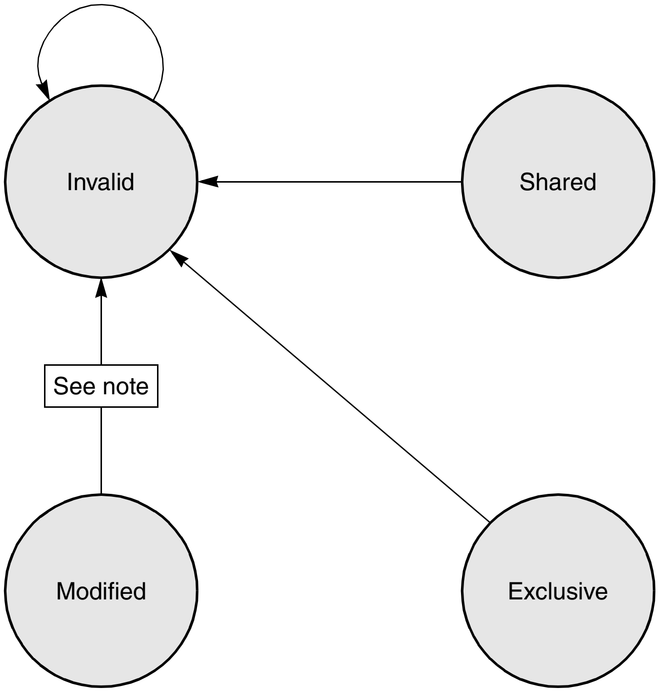


Figure 3-11. Clean Transaction—60x and MPX Bus Modes, MSSCR0[EIDIS] = 1



Note: If another master asserts $\overline{\text{ARTRY}}$, the MPC7450 performs a window-of-opportunity style push. Otherwise, there is no intervention.

Figure 3-12. Kill Transaction—60x and MPX Bus Modes, MSSCR0[EIDIS] = 1

3.3.2.6 Reservation Snooping

The MPC7450 snoops all transactions against the contents of the reservation address register independent of the cache snooping. The following state diagrams (Figure 3-13, Figure 3-14, and Figure 3-15) show the response to those snoops.

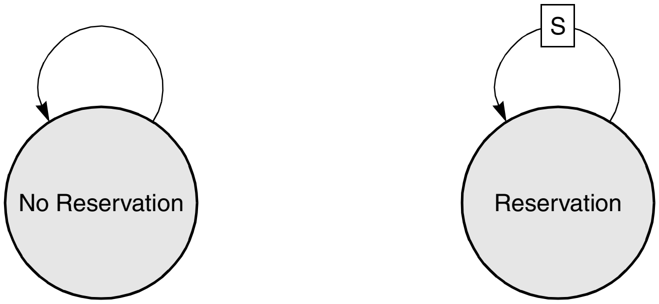


Figure 3-13. Read Transaction Snoop Hit on the Reservation Address Register

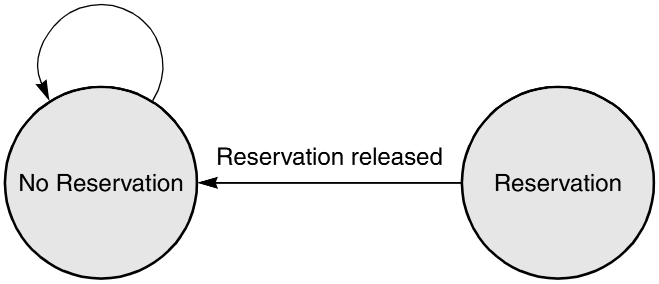


Figure 3-14. Reskill Transaction Snoop Hit on the Reservation Address Register

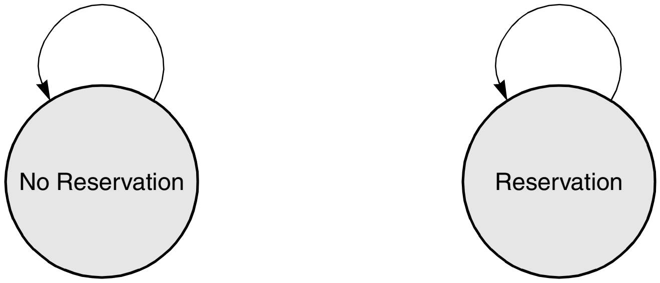


Figure 3-15. Other Transaction Snoop Hit on the Reservation Address Register

3.3.3 Load/Store Operations and Architecture Implications

Load and store operations are assumed to be weakly ordered on the MPC7450. The load/store unit (LSU) can perform load operations that occur later in the program ahead of store operations, even when the data cache is disabled (see Section 3.3.3.2, “Sequential Consistency of Memory Accesses”).

The MPC7450 does not provide support for direct-store segments. Operations attempting to access a direct-store segment cause a DSI exception. For additional information about DSI exceptions, refer to Section 4.6.3, “DSI Exception (0x00300).”

3.3.3.1 Performed Loads and Store

The PowerPC architecture defines a performed load operation as one that has the addressed memory location bound to the target register of the load instruction. The architecture defines a performed store operation as one where the stored value is the value that any other processor will receive when executing a load operation (that is, of course, until it is changed again). With respect to the MPC7450, caching-allowed (WIMG = x0xx) loads and caching-allowed, write-back (WIMG = 00xx) stores are performed when they have arbitrated to address the cache block in the L1 data cache, the L2 cache, the L3 cache, or the system interface. Note that loads are considered performed at the L1 data cache, L2 cache, or L3 cache only if the respective cache contains a valid copy of that address. Write-back stores are considered performed at the L1 data cache, L2 cache, or L3 cache only if the respective cache contains a valid, non-shared copy of that address. Caching-inhibited (WIMG = x1xx) loads and stores, and write-through (WIMG = 10xx) stores are considered performed when they have been successfully presented to the external system bus. A set of rules for load and store ordering using the WIMG bits in the BAT registers or page table entries (PTE) in the MPC7450 is listed in Table 3-5.

Table 3-5. Load and Store Ordering with WIMG Bit Settings

W	I	M	G	Order[1], [2]
n	1	*n*	1	Stores are ordered with respect to other stores. Loads are ordered with respect to other loads. A store followed by a load requires an **eieio**.instruction in between the store and load.
1	0	*n*	1	Stores are ordered with respect to other stores. Loads are ordered with respect to other loads. A store followed by a load requires a **sync**.instruction in between the store and load.
1	*n*	*n*	0	Stores are ordered with respect to other stores. A load followed by a load requires a **sync**.instruction in between the loads. A store followed by a load requires a **sync**.instruction in between the store and load.
0	0	1	*n*	A store followed by a store requires an **eieio**.instruction in between the stores. A load followed by a load requires a **sync**.instruction in between the loads. A store followed by a load requires a **sync**.instruction in between the store and load.

Table 3-5. Load and Store Ordering with WIMG Bit Settings

W	I	M	G	Order[1], [2]
0	0	0	*n*	A store followed by a store requires an **eieio**.instruction in between the stores. A load followed by a load requires a **sync**.instruction in between the loads. A store followed by a load requires a **sync**.instruction in between the store and load.
0	1	*n*	0	A store followed by a store requires an **eieio**.instruction in between the stores. A load followed by a load requires a **sync**.instruction in between the loads. A store followed by a load requires a **sync**.instruction in between the store and load.

[1] Any load followed by any store is always ordered for the MPC7450.

[2] A **sync** instruction will cover the synchronization cases that require an **eieio** instruction. However, an **eieio** instruction will not cover all the synchronization cases that require a **sync** instruction.

3.3.3.2 Sequential Consistency of Memory Accesses

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

The MPC7450 achieves sequential consistency by operating a single data pipeline to the cache/MMU. All memory accesses are presented to the MMU in exact program order and therefore exceptions are determined in order. Loads are allowed to bypass stores after exception checking has been performed for the store, but data dependency checking is handled in the load/store unit so that a load does not bypass a store with an address match. Newer caching-allowed loads can bypass older caching-allowed loads only if the two loads are to different 32-byte address granules. Newer caching-allowed write-back stores can bypass older caching-allowed write-back stores if they do not store to overlapping bytes of data.

Note that although memory accesses that miss in the L1 cache are forwarded to the load/store unit load queue for future arbitration for the L2 cache (and possibly the L3 cache and external bus), all potential synchronous exceptions have been resolved before the L1 cache access. In addition, although subsequent memory accesses can address the L1 cache, full coherency checking between the L1 cache and the load/store unit load and store queues is provided to avoid dependency conflicts.

3.3.3.3 Load Ordering with Respect to Other Loads

The PowerPC architecture guarantees that the following loads are not re-ordered with respect to other similar loads:

- Caching-inhibited (I = 1) and guarded (G = 1) loads

The MPC7450 guarantees that the following loads are not re-ordered with respect to other similar loads:

- Caching-inhibited (I = 1) loads when HID0[SPD] = 1

Note that when address translation is disabled (real addressing mode), the default WIMG bits cause the I bit to be cleared (accesses are assumed to be caching-allowed), and thus the load accesses are weakly ordered with respect to each other. Refer to Section 5.2, “Real Addressing Mode,” for a description of the WIMG bits when address translation is disabled.

3.3.3.4 Store Ordering with Respect to Other Stores

The PowerPC architecture also guarantees that the following stores are not re-ordered with respect to other similar stores:

- Caching-inhibited (I = 1) stores

Additionally, the MPC7450 also guarantees that the following stores are not re-ordered with respect to other similar stores:

- Write-through (W = 1) stores

Otherwise, stores on the MPC7450 are weakly ordered with respect to other stores.

3.3.3.5 Enforcing Store Ordering with Respect to Loads

The PowerPC architecture specifies that an **eieio** instruction must be used to ensure sequential ordering of loads with stores.

The MPC7450 guarantees that any load followed by any store is performed in order (with respect to each other). The reverse, however, is not guaranteed. An **eieio** instruction must be inserted between a store followed by a load to ensure sequential ordering between that store and that load. Also note that setting HID0[SPD] does not prevent loads from bypassing stores.

If store gathering is enabled (through HID0[SGE]), the **eieio** instruction may also be used to keep stores from being gathered. If an **eieio** instruction is detected in the store queues, store gathering is not performed. If HID1[SYNCBE] = 1, the **eieio** instruction also causes a system bus broadcast operation, which may be used to prevent external devices, such as a bus bridge chip, from gathering stores. See Section 3.1.2.3, “Store Gathering/Merging,” for more information on store gathering.

3.3.3.6 Atomic Memory References

The PowerPC architecture defines the Load Word and Reserve Indexed (**lwarx**) and the Store Word Conditional Indexed (**stwcx.**) instructions to provide an atomic update function for a single, aligned word of memory. These instructions can be used to develop a rich set of multiprocessor synchronization primitives. Note that atomic memory references constructed using **lwarx**/**stwcx.** instructions depend on the presence of a coherent memory system for correct operation. These instructions should not be expected to provide atomic access to noncoherent memory. For detailed information on these instructions, refer to Chapter 2, “Programming Model,” in this book and Chapter 8, “Instruction Set,” in the *Programming Environments Manual*.

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

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

Executing an **lwarx** or **stwcx.** instruction to areas marked write-through or cache-inhibited causes a DSI exception. Additionally, executing an **lwarx** or **stwcx.** instruction when the L1 data cache is disabled or it is enabled and locked causes a DSI exception.

If the page table entry is marked caching-allowed (WIMG = x0xx) and an **lwarx** access misses in the cache, the MPC7450 performs a cache block fill. All bus operations that are a direct result of either an **lwarx** instruction or an **stwcx.** instruction are placed on the bus with a special encoding. Note that this does not force all **lwarx** instructions to generate bus transactions, but rather provides a means for identifying when an **lwarx** instruction does generate a bus transaction.

The MPC7450 snoops its own RWITM-atomic transactions to check the state of the reservation bit. If the reservation is set, the RWITM-atomic transaction succeeds. Otherwise, the MPC7450 internally retries it (as if it had asserted $\overline{\text{ARTRY}}$) and the transaction is re-sent as a read transaction.

3.4 L1 Cache Control

The MPC7450 L1 caches are controlled by programming specific bits in the HID0, ICTRL, and LDSTCR special-purpose registers and by issuing dedicated cache control instructions. Section 3.4.1, “Cache Control Parameters in HID0,” describes the HID0 cache control bits, Section 3.4.2, “Data Cache Way Locking Setting in LDSTCR,” describes the data cache way locking feature and Section 3.4.3, “Cache Control Parameters in ICTRL,” describes the L1 cache parity checking features and the instruction cache way locking. Note that the ICTC register also affects the instruction cache operation and it is described in Section 10.3, “Instruction Cache Throttling.”

Also, Section 2.2.5.1, “Hardware Implementation-Dependent Register 0 (HID0),” Section 2.2.5.3, “Memory Subsystem Control Register (MSSCR0),” and Section 2.2.5.5.22, “Load/Store Control Register (LDSTCR),” provide detailed information on the bit settings for these registers.

Finally, Section 3.4.4, “Cache Control Instructions,” describes the cache control instructions.

See Section 3.6.3, “L2 Cache Control,” for information on the L2 cache control functions and Section 3.7, “L3 Cache Interface,” for more information on the L3 cache.

3.4.1 Cache Control Parameters in HID0

The HID0 special-purpose register contains several bits that invalidate, disable, and lock the instruction and data caches. The following sections describe these L1 cache control facilities.

3.4.1.1 Enabling and Disabling the Data Cache

The data cache is enabled or disabled with the data cache enable bit, HID0[DCE]. HID0[DCE] is cleared on power-up, disabling the data cache. Snooping is not performed when the data cache is disabled.

When the data cache is in the disabled state (HID0[DCE] = 0), the cache tag status bits are ignored, and all data accesses are propagated to the system bus as single- or double-beat cache-inhibited ($\overline{\text{CI}}$ asserted) transactions, depending on the size of the access. Thus, they are ignored by the L2 and L3 caches, independent of the state of the L2 and L3. Note that disabling the data cache does not affect the translation logic; translation for data accesses is controlled by MSR[DR].

The setting of the DCE bit must be preceded by a **sync** instruction to prevent the cache from being enabled or disabled in the middle of a data access. In addition, the cache must be globally flushed before it is disabled to prevent coherency problems when it is re-enabled. See Section 3.5.8, “L1 Cache Invalidation and Flushing,” for more information on the flushing of the data cache.

The **dcbz** instruction causes an alignment exception when the access is to a cache-inhibited or write-through area of memory. Thus a **dcbz** causes an alignment exception for the cases when the data cache is disabled (HID0[DCE] = 0), or when the data cache is completely locked (LDSTCR[DCWL] = 0xFF or HID0[DLOCK] = 1). The touch load (**dcbt** and **dcbtst**) instructions are no-ops when the data cache is disabled; however, address translation is still performed for these instructions. Other cache instructions (**dcbf**, **dcbst**, and **dcbi**) do not affect the data cache when it is disabled.

Note that if the L1 data cache is disabled, the L2 and the L3 caches may be enabled, but they ignore all data accesses. The L2 cache is enabled or disabled with L2CR[L2E], and the L3 cache is enabled or disabled with L3CR[L3E].

3.4.1.2 Data Cache Locking with DLOCK

The entire contents of the data cache can be locked by setting the data cache lock bit, HID0[DLOCK]. No new tags are allocated for a locked data cache. Snoop hits, store hits (to mark the line modified), and **dcbf**, **dcbi**, and **dcbst** instructions are the only operations that can cause a tag state change in a locked data cache. If all ways of the data cache are locked, all stores are sent to the memory subsystem as cacheable but write-through (as if W = 1). Accesses caused by the **dcbz** instruction when the data cache is completely locked take an alignment exception as described in Section 3.4.1.1, “Enabling and Disabling the Data Cache.” However, accesses caused by the **dcba** instruction when the data cache is completely locked are treated as a no-op.

The setting of the DLOCK bit must be preceded by a **dssall/sync** instruction pair and followed by a **sync** instruction to prevent the data cache from being locked during a data access. Also, the data cache should be already enabled when setting DLOCK.

The MPC7450 treats a load hit to a locked data cache the same as a load hit to an unlocked data cache. That is, the data cache services the load with the requested data. However, a load that misses in a locked data cache is passed to the LMQ and propagates to the L2, L3 cache or system bus as a caching-allowed, 32-byte burst read. In this case, the data is forwarded to the requesting execution unit when it returns, but it is not loaded into the data cache.

The MPC7450 treats snoop hits to a locked data cache the same as snoop hits to an unlocked data cache. However, any cache block invalidated by a snoop hit remains invalid and is not reallocated until the cache is unlocked.

One to eight ways of the data cache can be locked by setting bits in LDSTCR. See Section 3.4.2, “Data Cache Way Locking Setting in LDSTCR,” for more information on way locking of the data cache.

3.4.1.3 Enabling and Disabling the Instruction Cache

The instruction cache may be enabled or disabled through the use of the instruction cache enable bit, HID0[ICE]. HID0[ICE] is cleared on power-up, disabling the instruction cache. The setting of the ICE bit must be preceded by an **isync** instruction to prevent the cache from being enabled or disabled in the middle of an instruction fetch. Furthermore, the setting of the ICE bit must be followed by an **isync** instruction in order for the setting to take effect. The **icbi** instruction is not affected by disabling the instruction cache. For further details on synchronization see Section 2.4.2.4.1, “Context Synchronization.”

When the instruction cache is in the disabled state (HID[ICE] = 0), the cache tag status bits are ignored, and all instruction fetches are forwarded to the L2 and L3 caches and the memory subsystem with the cacheability attribute determined by the WIMG bits. When the instructions are returned, they are forwarded to the instruction unit, but are not loaded into the instruction cache. Note that the $\overline{\text{CI}}$ signal always reflects the state of the caching-inhibited memory/cache access attribute (the I bit) for instruction accesses independent of the state of HID0[ICE]. Also note that disabling the instruction cache does not affect the translation logic; translation for instruction accesses is controlled by MSR[IR].

3.4.1.4 Instruction Cache Locking with ILOCK

The contents of the instruction cache can be locked by setting the instruction cache lock bit, HID0[ILOCK]. A completely locked instruction cache has no new tag allocations. **icbi** instructions are the only operations that can cause a tag state change in a locked instruction cache. The setting of the ILOCK bit must be preceded by an **isync** instruction to prevent the instruction cache from being locked during an instruction fetch.

An instruction fetch that hits in a locked instruction cache is serviced by the cache. An instruction fetch that misses in a completely locked instruction cache is propagated to the L2, L3, and system bus as a 32-byte burst read. When the instructions are returned, they are forwarded to the instruction unit but are not loaded into the instruction cache.

Note that the $\overline{\text{CI}}$ signal always reflects the state of the caching-inhibited memory/cache access attribute (the I bit) for instruction accesses independent of the state of HID0[ILOCK]. See Section 3.4.3.1, “Instruction Cache Way Locking,” for information on the locking of one to 8 ways of the instruction cache.

3.4.1.5 L1 Instruction and Data Cache Flash Invalidation

The HID0[ICFI] and HID0[DCFI] bits of the MPC7450 cause a flash invalidation of the instruction and data caches, respectively. Each cache can be flash invalidated independently. Note that HID0[ICFI] and HID0[DCFI] must not both be set with the same **mtspr** instruction, due to the synchronization requirements described in Section 2.4.2.4.1, “Context Synchronization.”

A reset operation does not invalidate the caches. Therefore, software must flash invalidate the instruction cache with the same **mtspr** to HID0 instruction that enables the instruction cache, and it must flash invalidate the data cache with the same **mtspr** to HID0 instruction that enables the data cache. When either HID0[ICFI] or HID0[DCFI] is set by software, the corresponding cache invalidate bit is cleared automatically in the following clock cycle. Note that there is no broadcast of a flash invalidate operation. An **isync** must precede the setting of the HID0[ICFI] in order for the setting to take effect.

Individual instruction cache blocks can be invalidated using the **icbi** instruction and individual data cache blocks can be invalidated using the **dcbi** instruction. See Section 3.4.4.8, “Instruction Cache Block Invalidate (icbi),” and Section 3.4.4.7, “Data Cache Block Invalidate (dcbi),” for more information about the **icbi** and **dcbi** instructions, respectively.

3.4.2 Data Cache Way Locking Setting in LDSTCR

The 8-bit DCWL parameter in LDSTCR controls the locking of from one to 8 ways of the data cache. Each bit in DCWL corresponds to a way of the data cache. Setting a bit in DCWL locks the corresponding way in the cache. The MPC7450 treats a load hit to a locked way in the data cache the same as a load hit to an unlocked data cache. That is, the data cache services the load with the requested data. Also, snoop hits and store hits to locked way in the data cache also operate the same as a hit to an unlocked cache. However, locked ways are never selected for replacement.

Setting all 8 bits is equivalent to setting the HID0[DLOCK] bit. See Section 3.4.1.2, “Data Cache Locking with DLOCK,” for more information. See Section 3.5.7.4, “Cache Locking and PLRU,” for more information on PLRU precautions with way locking.

3.4.3 Cache Control Parameters in ICTRL

The ICTRL controls instruction and data cache parity checking and error reporting and enables instruction cache way locking

3.4.3.1 Instruction Cache Way Locking

Similar to the DCWL parameter in LDSTCR, the 8-bit ICWL parameter in ICTRL controls the locking of from one to 8 ways of the instruction cache. Each bit in ICWL corresponds to a way of the instruction cache. Setting a bit in ICWL locks the corresponding way in the cache. The MPC7450 treats a hit to a locked way in the instruction cache the same as a hit to an unlocked instruction cache. That is, the cache services the fetch with the requested instructions. However, on a miss, locked ways are never selected for replacement.

Setting all 8 bits in ICWL is equivalent to setting the HID0[ILOCK] bit. See Section 3.4.1.4, “Instruction Cache Locking with ILOCK,” for more information. See Section 3.5.7.4, “Cache Locking and PLRU,” for more information on PLRU precautions with way locking.

3.4.3.2 Enabling Instruction Cache Parity Checking

Instruction cache parity checking is enabled with ICTRL[EICP]. When this bit is set, the parity of all instructions fetched from the L1 cache is checked. See Section 3.4.3.3, “Instruction and Data Cache Parity Error Reporting,” for information on the reporting of L1 cache parity errors.

3.4.3.3 Instruction and Data Cache Parity Error Reporting

Instruction and data cache parity errors are reported through the machine check exception mechanism if ICTRL[EICE] and ICTRL[EDCE] are set, respectively. In order for an instruction cache parity error to be reported, ICTRL[EICP] must also be set. Note that data parity checking is always enabled. When ICTRL[EICE] and ICTRL[EDCE] are cleared, instruction and data cache parity errors are masked. Note that when parity checking and reporting is enabled, parity errors can be reported (causing a machine check) for speculative fetches that result in a parity error, even if the access is never required.

3.4.4 Cache Control Instructions

The PowerPC architecture defines instructions for controlling both the instruction and data caches (when they exist). The cache control instructions: **dcbt**, **dcbtst**, **dcbz**, **dcbst**, **dcbf**, **dcba**, **dcbi**, and **icbi**—are intended for the management of the L1 caches. The MPC7450 interprets the cache control instructions as if they pertain only to its own L1 caches. These instructions are not intended for managing other caches in the system (except to the extent necessary to maintain coherency).

The MPC7450 snoops all global ($\overline{\text{GBL}}$ asserted) cache control instruction broadcasts. The **dcbst**, **dcbf**, and **dcbi** instructions cause a broadcast on the system bus (when M = 1) to maintain coherency. When M = 0, the broadcast of those instructions (and **icbi**, **tlbie**, and **tlbsync**) is controlled by the HID1[ABE] parameter. Therefore, HID1[ABE] must be set in multiprocessor systems.

The MPC7450 treats any cache control instruction directed to a direct-store segment (SR[T] = 1) as a no-op.

3.4.4.1 Data Cache Block Touch (dcbt)

The Data Cache Block Touch (**dcbt**) instruction provides potential system performance improvement through the use of a software-initiated prefetch hint. Note that implementations that support the PowerPC architecture are not required to take any action based on the execution of these instructions, but they may choose to prefetch the cache block corresponding to the effective address into their cache.

If the effective address of a **dcbt** instruction is directed to a direct-store segment (SR[T] = 1)x, or if HID0[NOPTI] = 1, the MPC7450 treats the instruction as a no-op without translation.

If the effective address of a **dcbt** instruction is not directed to a direct-store segment [T = 0] and HID0[NOPTI] = 0, the effective address is computed, translated, and checked for protection violations as defined in the PowerPC architecture. The **dcbt** instruction is treated as a load to the addressed byte with respect to address translation and protection. Note, however that a table search operation is never initiated for a **dcbt** instruction.

Additionally, the MPC7450 treats the **dcbt** instruction as a no-op if any of the following occur:

- A valid address translation is not found in the BAT or TLB
- Load accesses are not permitted to the addressed page (protection violation)
- The BAT or PTE is marked caching-inhibited (I = 1)
- The BAT or PTE is marked guarded (G = 1) and the **dcbt** instruction is not at the bottom of the completion queue
- The data cache is locked or disabled

If none of the conditions for a no-op are met, the MPC7450 checks if the addressed cache block is in the L1 data cache. If the cache block is not in the L1 data cache, the MPC7450 checks if the addressed cache block is in the L2 or L3 caches. If the cache block is not in the L2 or L3 cache, the MPC7450 initiates a burst read (with no intent to modify) on the system bus.

The data brought into the cache as a result of this instruction is validated in the same manner that a load instruction would be (that is, it is marked as exclusive or shared). Note that the successful execution of the **dcbt** instruction affects the state of the TLB and cache LRU bits as defined by the PLRU algorithm (see Section 3.5.7, “L1 Cache Block Replacement Selection”).

3.4.4.2 Data Cache Block Touch for Store (dcbtst)

The Data Cache Block Touch for Store (**dcbtst**) instruction behaves similarly to the **dcbt** instruction except that it attempts to gain ownership of the line by sending a request on the system bus if the data is not found in the L1, L2, or L3 caches in the exclusive or exclusive-modified state. Additionally, there are the following differences from **dcbt**:

- If the target address of a **dcbtst** instruction is marked write-through (W = 1), the instruction is treated as a no-op.
- If the **dcbtst** hits in the L1 data cache, the state of the block is not changed.
- If the **dcbtst** misses in the L1 data cache, but hits in the L2 or L3 cache as exclusive modified, the data is brought into the L1 data cache and is marked as exclusive.
- If the **dcbtst** misses in the L1 data cache, but hits in the L2 or L3 cache as shared, it is treated as a miss.
- If the **dcbtst** misses in both the L1 data cache and the L2 and L3 caches, the cache block fill request is signaled on the bus as a read (60x-bus mode) or as a read-claim (MPX bus mode) and the data is marked exclusive when it is brought into the L1 data cache from the system bus if the system response is not $\overline{\text{SHD}}$.

From a programming point of view, it can be advantageous to **dcbtst** instructions on the e600 core if multiple line misses may occur that would otherwise be caused by store instructions. This is because the MPC7450 supports only one outstanding store miss (from CSQ0), but **dcbtst** line misses are handled in the five-entry LMQ (so up to five **dcbtst** misses could be handled simultaneously).

If **dcbtst** (or **dstst**) is being used to prefetch a 32-byte coherency granule that will eventually be fully consumed by 32-byte’s worth of stores (that is, two back-to-back AltiVec **stvx** instructions), the inclusion of touch-for-store may reduce performance if the system is bandwidth-limited. This is because a touch-for-store must perform both a 32-byte coherency operation on the address bus (two or more bus cycles) and a 32-byte data transfer (four or more bus cycles). On the other hand, caching-allowed, write-back stores that merge to 32-bytes only require a 32-byte coherency operation (two or more bus cycles) because of the store-merging mechanism. In this scenario, using a **dcbz** to initialize the line sometime before the stores occur may also improve performance. See Section 3.1.2.3, “Store Gathering/Merging,” for more information.

3.4.4.3 Data Cache Block Zero (dcbz)

The effective address (EA) is computed, translated, and checked for protection violations as defined in the PowerPC architecture. The **dcbz** instruction is treated as a store to the addressed byte with respect to address translation, protection, and pipelining.

If the data is not found in the L1, L2, or L3 caches as exclusive or exclusive-modified, the physical address is broadcast on the system bus prior to the zero line fill if M = 1. Note the following:

- If the address hits in the L1 as exclusive or exclusive modified, zeros are written to the cache and the tag is marked as exclusive modified.
- If the address hits in the L1 as shared or misses in the L1, a lookup is performed in the L2 and L3 caches.
- If the address hits in the L1 as shared and M = 0, the lookup in the L2 and L3 caches is ignored, zeroes are written into the L1 cache, and the L1 tag is marked exclusive modified.
- If M = 1 and the L2 or L3 cache hits as exclusive or exclusive modified, zeros are written into the L1 and the L1 tag is marked exclusive modified.

Note that L1 cache misses for **dcbz** instructions follow the same line replacement algorithm as load misses to the L1 cache.

Executing a **dcbz** instruction can cause the following exceptions (noted in order of priority):

- Executing a **dcbz** instruction to a disabled or locked data cache generates an alignment exception.
- Executing a **dcbz** instruction to an EA with caching-inhibited or write-through attributes also generates an alignment exception.
- BAT and TLB protection violations for a **dcbz** instruction generate DSI exceptions.
- A **dcbz** instruction can also cause a data TLB miss on store exception if HID0[STEN] = 1 and either no translation is found in the BAT or TLB, or the change bit in a matching TLB entry is cleared.

3.4.4.4 Data Cache Block Store (dcbst)

The effective address is computed, translated, and checked for protection violations as defined in the PowerPC architecture. This instruction is treated as a load with respect to address translation and memory protection.

If the address hits in the cache and the cache block is in the modified state, the modified block is written back to memory and the cache block is placed in the invalid state in the L1. If the address hits in the data cache and the cache block is in any state other than modified, an address-only broadcast (clean) is performed and the cache block is placed in the invalid state in the data cache.If the address additionally hits in the L2 or L3 cache, the line is written back to memory and placed in the exclusive state in the L2 or L3 that hit.

The function of this instruction is independent of the WIMG bit settings of the block or PTE containing the effective address. However, if the address is marked memory-coherency- required (M = 1), the execution of **dcbst** causes an address broadcast on the system bus (if HID1[ABE] = 1). If HID1[ABE] = 0, execution of **dcbst** only causes an address broadcast on the system bus if the data is modified. Execution of a **dcbst** instruction occurs whether or not the L1, L2, or L2 caches are disabled or locked. However, it has no effect on a disabled L1, L2, or L3 cache.

A BAT or TLB protection violation for a **dcbst** generates a DSI exception. Additionally, a **dcbst** instruction can also cause a data TLB miss on load exception if HID0[STEN] = 1 and no translation is found in the BAT or TLB.

3.4.4.5 Data Cache Block Flush (dcbf)

The effective address is computed, translated, and checked for protection violations as defined in the PowerPC architecture. This instruction is treated as a load with respect to address translation and memory protection.

Note the following:

- If the address hits in the L1, L2, or L3 cache, and the block is in the exclusive modified state, the modified block is written back to memory and the cache block is invalidated.
- If the address hits in the L1, L2, or L3 cache, and the cache block is in the exclusive unmodified or shared state, the cache block is invalidated.
- If the address misses in the L1, L2, or L3 cache, no action is taken.

The function of this instruction is independent of the WIMG bit settings of the block or PTE containing the effective address. However, if the address is marked memory-coherency- required, the execution of **dcbf** broadcasts an address-only FLUSH transaction on the system bus if HID1[ABE] = 1. Execution of a **dcbf** instruction occurs whether or not the L1, L2, or L3 caches are disabled or locked. However, it has no effect on a disabled L1, L2, or L3 cache.

A BAT or TLB protection violation for **dcbf** generates a DSI exception. Additionally, a **dcbf** instruction can also cause a data TLB miss on load exception if HID0[STEN] = 1 and no translation is found in the BAT or TLB. See Section 3.5.8, “L1 Cache Invalidation and Flushing,” for more information.

3.4.4.6 Data Cache Block Allocate (dcba)

The MPC7450 implements the Data Cache Block Allocate (**dcba**) instruction. This is currently an optional instruction in the PowerPC virtual environment architecture (VEA); however, it may become required in future versions of the architecture. The **dcba** instruction provides potential system performance improvement through the use of a software-initiated pre-store hit. This allows software to establish a block in the data cache in anticipation of a store into that block, without loading the block from memory.

The MPC7450 executes the **dcba** instruction the same as a **dcbz** instruction, with one exception. In cases when **dcbz** causes an exception, a **dcba** will no-op unless the exception is DSI for a data breakpoint match or to generate a software table search operation (with HID0[STEN] = 1). Note that the **dcba** instruction has no effect when the L1 cache is disabled or locked.

3.4.4.7 Data Cache Block Invalidate (dcbi)

When a **dcbi** instruction is executed, the effective address is computed, translated, and checked for protection violations as defined in the PowerPC architecture. This instruction is treated as a store with respect to address translation and memory protection.

This instruction is treated the same as a **dcbf** in the caches. The only difference between **dcbi** and **dcbf** on the MPC7450 is that the **dcbi** instruction is privileged.

A BAT or TLB protection violation for a **dcbi** translation generates a DSI exception.

3.4.4.8 Instruction Cache Block Invalidate (icbi)

The **icbi** instruction invalidates a matching entry in the instruction cache. During execution, the effective address for the instruction is translated through the data MMU and broadcasts on the system bus using the memory-coherency attribute from translation if HID1[ABE] = 1. This instruction is treated as a load with respect to address translation and memory protection.

The MPC7450 always sends the **icbi** to the instruction cache for cache block address comparison and invalidation. The **icbi** instruction invalidates a matching cache entry regardless of whether the instruction cache is disabled or locked. The L2 and L3 caches are not affected by the **icbi** instruction.

An **icbi** instruction should always be followed by a **sync** and an **isync** instruction. This ensures that the effects of the **icbi** are seen by the instruction fetches following the **icbi** itself. For self-modifying code, the following sequence should be used to synchronize the instruction stream:

1. **dcbst** or **dcbf** (push new code from L1 data cache, L2, and L3 cache out to memory)
2. **sync** (wait for the **dcbst** or **dcbf** to complete)
3. **icbi** (invalidate the old instruction cache entry in this processor and, by broadcasting the **icbi** to the bus, invalidate the entry in all snooping processors)
4. **sync** (wait for the **icbi** to complete its bus operation)
5. **isync** (re-sync this processor’s instruction fetch)

The second **sync** instruction ensures completion of all prior **icbi** instructions. Note that the second **sync** instruction is not shown in Section 5.1.5.2, “Instruction Cache Instructions,” in the *Programming Environments Manual*. This **sync** is required on the MPC7450.

Since the **sync** instruction strongly serializes the MPC7450’s memory subsystem, performance of code containing several **icbi** instructions can be improved by batching the **icbi** instructions together such that only one **sync** instruction is used to synchronize all the **icbi** instructions in the batch.

3.5 L1 Cache Operation

This section describes the MPC7450 cache operations performed by the L1 instruction and data caches.

3.5.1 Cache Miss and Reload Operations

This section describes the actions taken by the L1 caches on misses for cacheable accesses. Also, it describes what happens on cache misses for cache-inhibited accesses as well as disabled and locked L1 cache conditions.

3.5.1.1 Data Cache Fills

The MPC7450 data cache blocks are filled (sometimes referred to as a cache reload) from the L2 or L3 cache or the memory subsystem when cache misses occur for cacheable accesses, as described in Section 3.1.2, “Load/Store Unit (LSU),” and Section 3.1.3, “Memory Subsystem Blocks.”

When the data cache is disabled (HID0[DCE] = 0]), the MPC7450 treats all data accesses as cache-inhibited (as if the memory coherency bit I = 1). Thus, even if the access would have hit in the cache, it proceeds to the memory subsystem as cache-inhibited. When the data is returned, it is forwarded to the requesting execution unit, but it is not loaded into any of the caches.

From 0 to 8 ways of the data cache can be locked, as described in Section 2.2.5.5.22, “Load/Store Control Register (LDSTCR),” and all 8 ways can also be locked by setting HID0[DLOCK]. When at least one way is unlocked, misses are treated normally and they allocate in one of the unlocked ways on a reload. If all 8 ways are locked, load misses proceed to the memory subsystem as normal cacheable accesses. In this case, the data is forwarded to the requesting execution unit when it returns, but it is not loaded into the data cache. If all 8 ways are locked, stores are sent to the memory subsystem as cacheable but write-through (as if W = 1).

The accesses caused by the following instructions cause the MPC7450 to take a DSI exception when the data cache is disabled or completely locked:

- **lwarx** or **stwcx.**
- **dcbz**

Note that cache-inhibited stores do not access any of the caches. See Section 3.5.3, “Store Miss Merging,” for more information on the handling of cacheable store misses. Also, see Section 3.6.4.1, “L2 Cache Miss and Reload Operations,” and Section 3.7.7.1, “L3 Cache Miss and Reload Operations,” for more information on L2 and L3 cache fills, respectively.

3.5.1.2 Instruction Cache Fills

The instruction cache provides a 128-bit interface to the instruction unit, so four instructions can be made available to the instruction unit in a single clock cycle on an L1 instruction cache hit. On a miss, the MPC7450 instruction cache blocks are loaded in one 32-byte beat from the L2 cache; the instruction cache is nonblocking, providing for hits under misses.

The instruction cache operates similarly to the data cache when all eight ways are locked. When the instruction cache is disabled (HID0[ICE = 0]), the instruction accesses bypass the instruction cache. However, unlike the data cache, these accesses are forwarded to the memory subsystem as cacheable and proceed to the L2 and L3 caches. When the instructions are returned, they are forwarded to the instruction unit but are not loaded into the instruction cache.

The instruction unit fetches a total of eight instructions at a time directly from the memory subsystem for the following cases of cacheable instruction fetches:

- The instruction cache is disabled.
- The instruction cache is enabled, all 8 ways are locked, and the access misses in the L1 cache.

Note that the MPC7450 bursts out of reset in MPX or 60x bus mode.

The MPC7450 always uses burst transactions for instruction fetches. If the instruction cache is disabled (HID0[ICE]=0), the MPC7450 will do a four-beat burst for instruction fetches and discard the last two beats. If the instruction cache is enabled (HID0[ICE]=1), the MPC7450 will do a four-beat burst for instruction fetches and use all four beats. Externally, at the next I-fetch, the address will increment by 16 bytes if the instruction cache is disabled or the address will increment by 32 bytes if the instruction cache is enabled. For more details about disabling the instruction and data cache see Section 2.2.5.1, "Hardware Implementation-Dependent Register 0 (HID0)," Section 9.3.2.4.3, "Write-Through (WT), Cache Inhibit (CI), and Global (GBL) Signals," and Section 9.6.2.3.1, "60x Transfer Size (TSIZ[0:2]) and Transfer Burst (TBST) Signals."

Note that although the L1 instruction cache is physically addressed, the branch target instruction cache (BTIC) is virtually addressed. However, it is automatically flushed when the instruction cache is invalidated, when an exception occurs, or when a **tlbie**, **icbi**, **rfi**, or **isync** instruction is executed. Because the BTIC is automatically flushed any time the address mappings might change, aliases do not occur in the BTIC. See Section 6.3.1, "General Instruction Flow," for more information on the BTIC.

3.5.2 Cache Allocation on Misses

This section describes the allocation of cache lines for both instruction and data cache misses. See Section 3.5.7, "L1 Cache Block Replacement Selection," for more information on L1 cache block replacement. See Section 3.6.4.2, "L2 Cache Allocation," and Section 3.7.7.2, "L3 Cache Allocation," for more information on the allocation and replacement algorithms used by the L2 and L3 caches of the MPC7450, respectively.

3.5.2.1 Instruction Access Allocation in L1 Cache

Instruction cache misses cause a new line to be allocated into the instruction cache on a pseudo LRU basis, provided the cache is not completely locked or disabled.

3.5.2.2 Data Access Allocation in L1 Cache

Data load or write-back store accesses that miss in the L1 data cache function similarly to L1 instruction cache misses. They cause a new line to be allocated on a pseudo LRU basis, provided the cache is not completely locked or disabled.

Note that modified data in the replacement line of any of the caches can cause a castout to occur. In all of these cases, the castout is not initiated until the new data is ready to be loaded. Note that one data access can cause multiple castout operations to be initiated (from the various MPC7450 caches).

3.5.3 Store Miss Merging

Write-back stores that miss in the L1 data cache cause a data cache fill operation to occur using the load queues of the LSU. The store data is preserved internally, and when the remainder of the cache line has been loaded from the memory subsystem, the store data is merged in to the appropriate bytes of the cache line as it is loaded into the data cache. See Section 3.1.2.3, "Store Gathering/Merging," for more information on store merging and Section 3.6.4.3, "Store Data Merging and L2," for more information on store misses and the L2 cache.

3.5.4 Load/Store Miss Handling (MPC7448-Specific)

The MPC7448 adds support for a second cacheable store miss, such that five loads/touches and/or two cacheable store misses (or dcbz/dcba fetches) can be in progress. Note that due to the constraint imposed by the size of the L1 castout queue (six entries), the maximum number of outstanding L1 data cache misses on the MPC7448 remains six, but the ability to support an additional cacheable store miss prevents a stall from occurring if two successive stores miss in the L1. The MPC7448 can also have two instruction fetches (L1 instruction cache misses) and three L2 alternate sector hardware prefetches active in the memory subsystem.

3.5.5 Store Hit to a Data Cache Block Marked Shared

When a write-back store hits in the L1 data cache and the block is shared, the target block is invalidated in the data cache. The current data from the target block is then treated as a store miss.

3.5.6 Data Cache Block Push Operation

When an L1 cache block in the MPC7450 is snooped (by another bus master) and the data hits and is modified, the cache block must be written to memory and made available to the snooping device. The push operation propagates out to the L2 and L3 caches, as well as the system bus. The cache block that hits is said to be pushed out onto the system bus.

3.5.7 L1 Cache Block Replacement Selection

Both the instruction and data cache use a pseudo least-recently-used (PLRU) replacement algorithm described in this section when a new block needs to be placed in the cache. Note that data cache replacement selection is performed at reload time, not when a miss occurs. Instruction cache replacement selection occurs when an instruction cache miss is first recognized.

3.5.7.1 PLRU Replacement

Each L1 cache is organized as eight blocks (ways) per set by 128 sets. There is a identifying bit for each way in the cache, L[0–7]. The PLRU algorithm is used to select the replacement target. There are seven PLRU bits, B[0–6] for each set in the cache.

This algorithm does not prioritize replacing invalid entries over valid ones; a way is selected for replacement according to the PLRU bit encodings shown in Table 3-6.

Table 3-6. L1 PLRU Replacement Way Selection

If the PLRU bits are:						Then the way selected for replacement is:
B0	0	B1	0	B3	0	L0
	0		0		1	L1
	0		1	B4	0	L2
	0		1		1	L3
	1	B2	0	B5	0	L4
	1		0		1	L5
	1		1	B6	0	L6
	1		1		1	L7

The PLRU algorithm is shown graphically in Figure 3-16.

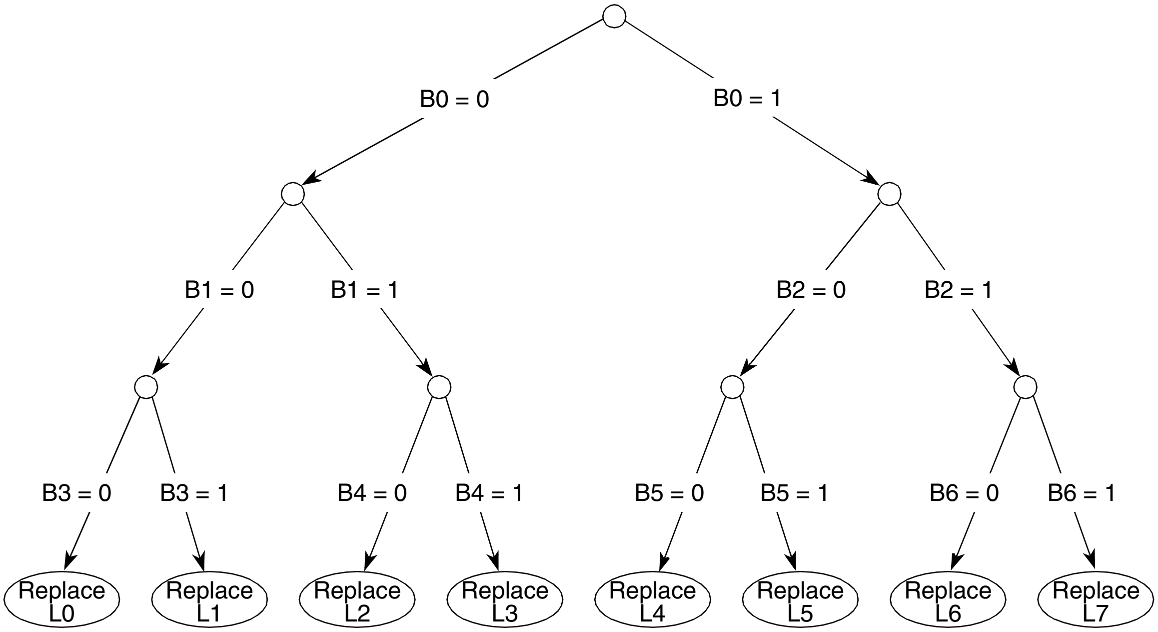


Figure 3-16. PLRU Replacement Algorithm

During power-up or hard reset, the valid bits of the L1 caches are not necessarily cleared and they must be explicitly cleared by setting the respective flash invalidate bits (HID0[DCFI] or HID0[ICFI]) before each cache is enabled. Subsequently, the PLRU bits are cleared to point to way L0 of each set.

3.5.7.2 PLRU Bit Updates

Except for snoop accesses, each time a cache block is accessed, it is tagged as the most recently used way of the set (unless accessed by the AltiVec LRU instructions; refer to Section 7.1.2.1, “LRU Instructions”). For every hit in the cache or when a new block is reloaded, the PLRU bits for the set are updated using the rules specified in Table 3-7. Note that only three PLRU bits are updated for any given access.

Table 3-7. PLRU Bit Update Rules

If the current access is to:	Then the PLRU bits in the set are changed to the following[1]:						
	B0	B1	B2	B3	B4	B5	B6
L0	1	1	x	1	x	x	x
L1	1	1	x	0	x	x	x
L2	1	0	x	x	1	x	x
L3	1	0	x	x	0	x	x
L4	0	x	1	x	x	1	x
L5	0	x	1	x	x	0	x
L6	0	x	0	x	x	x	1
L7	0	x	0	x	x	x	0

[1] x = Does not change.

3.5.7.3 AltiVec LRU Instruction Support

The data cache fully supports the AltiVec LRU instructions (**lvxl**, **stvxl**). If one of these instructions causes a hit in the data cache, the PLRU bits are updated such that the way which hit is marked as least-recently-used by using the PLRU update rules shown in Table 3-8. If no other hit to the cache index occurs, this way is selected for replacement upon the next data cache reload. Similarly, if an **lvxl** or **stvxl** instruction misses in the cache, the PLRU bits are updated, as shown in Table 3-8, when that cache block reloads the data cache. Note that the instruction cache is not subject to any AltiVec LRU accesses.

Table 3-8. PLRU Bit Update Rules for AltiVec LRU Instructions

If the current AltiVec LRU access is to:	Then the PLRU bits in the set are changed to the following[1]:						
	B0	B1	B2	B3	B4	B5	B6
L0	0	0	x	0	x	x	x
L1	0	0	x	1	x	x	x
L2	0	1	x	x	0	x	x
L3	0	1	x	x	1	x	x
L4	1	x	0	x	x	0	x
L5	1	x	0	x	x	1	x
L6	1	x	1	x	x	x	0
L7	1	x	1	x	x	x	1

[1] x = Does not change.

Note that an AltiVec LRU access simply inverts the update value of the three PLRU bits when compared to the normal (most-recently-used) update rules.

3.5.7.4 Cache Locking and PLRU

Care should be taken when locking between 1 and 8 ways in either of the L1 caches. For the best performance, there should be an equal number of locked ways on each side of each decision point of the binary tree shown in Figure 3-16, or all ways should be locked. Otherwise, the PLRU replacement algorithm will be biased to replace certain ways.

3.5.8 L1 Cache Invalidation and Flushing

When software guarantees that memory is not shared, the data cache can be invalidated by executing a series of loads followed by **dcbf** (or **dcbi**) instructions or by setting HID0[DCFI]. The instruction cache can be invalidated by setting HID0[ICFI].

When coherency is required to be maintained and data is shared among caches in a system, and the cache is going to be disabled or reconfigured, all the modified data in the data cache can be flushed by executing the following instructions in this order:

1. Way n:
   - a. Start with a base offset of zero. Perform a load followed by a **dcbf** instruction to that same address.
   - b. Increment the base offset by 32 bytes and perform the load/**dcbf** pair to the new address.
   - c. Repeat step b126 more times so that each load/**dcbf** pair addresses a different cache line in a way (progressing through all 128 combinations of PA[24:30]), assuming 36-bit physical addressing).
2. Way n + 1: Repeat the process shown in step 1 for the next way in the cache. This is started by incrementing the base offset used for the last set in way n by 32 bytes. Now PA[20:23] is incremented by one. Then repeat the remainder of step 1.
3. Way n + 2 to way n + 7: Repeat the process described in step 2 six more times (effectively progressing through all 8 combinations of PA[20:23]).

The **dcbf** instructions described above are not required if the loads in the sequence can be guaranteed to replace (flush) all the modified data in the cache and the loads can be from known addresses that will not be modified. This can be accomplished by loading from a memory range that will not be modified.

Exceptions and other events that can access the L1 cache should be disabled during this time so that the PLRU algorithm can function undisturbed. However, if it is impossible to disable exceptions and other events that can affect the PLRU, the sequence shown above can be modified as follows:

- Lock all ways in the data cache except way n. Then perform the process in step 1 above.
- Lock all ways in the data cache except way n + 1 and perform step 2, continuing with step 3 by unlocking way n + 2 through way n + 7 and performing the load/**dcbf** pairs for each unlocked way, one way at a time.

To minimize the time required to flush all the caches in the MPC7450, the L1 data cache can be flushed before flushing either the L2 or L3 caches, thus eliminating the flushing of the same line multiple times if it is modified in both the L1 and the L2 and/or L3 caches. Note that if cache flushing is performed without using the **dcbf** instruction and the L2 and/or L3 are flushed before the L1, the L2 and L3 should be disabled before flushing the L1 cache. This avoids loading of modified data into the L2 and L3. See 3.6.3.1.5, "Flushing of L1, L2, and L3 Caches," and Section 3.7.3.7, "L3 Cache Flushing," for more information on flushing the L2 and L3 caches, respectively.

3.5.9 L1 Cache Operation Summary

Table 3-10 summarizes all L1 cache activities caused by internal conditions. Table 3-9 defines some of the abbreviations used in Table 3-10. Note that the WIM bits are passed on to the memory subsystem unless explicitly shown as overridden in the MSS request type column of Table 3-10. See Section 3.8.4.2, "L1 Cache State Transitions and Bus Operations Due to Snoops," for a detailed description of L1 cache state transitions caused by external bus snooping.

Table 3-9. Definitions for L1 Cache-State Summary

Term	Definition
Load	One of the following instructions: **lbz**, **lbzx**, **lbzu**, **lbzux**, **lhz**, **lhzx**, **lhzu**, **lhzux**, **lha**, **lhax**, **lhau**, **lhaux**, **lwz**, **lwzx**, **lwzu**, **lwzux**, **lhbrx**, **lwbrx**, **lmw**, **lswi**, **lswx**, **lvebx**, **lvehx**, **lvewx**, **lvx**, **lvxl**, **lvsl**,and **lvsr**. A load reads cache memory and returns a data value of between 1 and 16 bytes. If the data is not in the L1 cache, the access causes a request to lower cache/memory to reload the L1 cache with the 32-byte cache line containing the requested data. If the 8/16 bytes of data (depending on size) containing the requested data are available before the rest of the cache line, this critical double-word is forwarded to the requesting execution unit before the line is reloaded. Note that misaligned loads and load string or load multiple may cause multiple memory accesses.
Store	One of the following instructions: **stb**, **stbx**, **stbu**, **stbux**, **sth**, **sthx**, **sthu**, **sthux**, **stw**, **stwx**, **stwu**, **stwux**, **sthbrx**, **stwbrx**, **stmw**, **stswi**, **stswx**, **stvebx**, **stvehx**, **stvewx**, **stvx**, **stvxl**. Stores cause an update of cache and/or memory of 1–16 bytes of data, depending on the WIM settings. Stores may cause a reload similar to loads above. Stores do not cause forwarding of data. Note that misaligned stores and store string or store multiple may cause multiple memory accesses.
Touch	One of the following instructions: **dcbt** or **dst**. Touches may cause a reload similar to loads above. Touches do not cause forwarding of data. Note that data stream touch (**dst**) may cause multiple memory accesses.
Store Touch	One of the following instructions: **dcbtst** or **dstst**. Store touches may cause a reload similar to loads above. Store touches do not cause forwarding of data. Note that data stream touch for store (**dstst**) may cause multiple memory accesses
dss	Data stream stop. It causes the tagged stream to stop prefetching. It is not sent to the MSS, and has no effect on prefetch requests already sent to the MSS.
lwarx	The same as loads above, but also causes the setting of the reservation bit in the processor.
stwcx.	The same as stores above, but the store is not performed unless the reservation is set, and the reservation is cleared once the store passes the coherency point.
dcbst, **dcbf**	Push modified data from any processor out to memory, and change valid lines to invalid.
dcbz, **dcba**	Claims ownership of a line without reloading the data and zeroes out the line.
L1 Deallocate	Caused by the allocation of a line in the L1 for a reload or **dcbz**. A deallocation casts out modified data and invalidates the line.
MSS request and MSS response	Memory subsystem request and memory subsystem response.
same	The state is unchanged.

Table 3-9. Definitions for L1 Cache-State Summary (continued)

Term	Definition
x	Do not care.
n/a	Does not apply.

Table 3-10. L1 Cache-State Transitions and MSS Requests

Internal Operation	WIM Setting	Initial L1 State	MSS Request	MSS Response	Final L1 State	Comments
Load	I = 0	I	Load	S	S	Load miss. Deallocate a line in the cache and reload the missing one from the MSS.
				E	E	
		S/E/M	none	n/a	same	Load hit—return data from L1.
	I = 1	n/a	Load	n/a	n/a	Cache-inhibited load
dcbt/dst	I = 0	I	Touch	S	S	Touch miss. Deallocate a line in the cache and reload the missing one from the MSS
				E	E	
		S/E/M	none	n/a	same	No-op
	I = 1	n/a	none	n/a	n/a	No-op
dcbtst/dsts	I = 0	I/S	Store Touch	S	S	Store touch miss. Reload the missing/shared one from the MSS If missing; deallocate a line.
				E	E	
		E/M	none	n/a	same	No-op
	I = 1	n/a	none	n/a	n/a	No-op
dss	x	n/a	none	n/a	n/a	Stops a **dst** or **dsts**.
lwarx	I = 0	I	LWARX	S	S	Same as load, but atomic bit is set on MSS access and reservation is set.
				E	E	
		S/E/M	none	n/a	same	Same as load hit, but set reservation
	I = 1	n/a	none	n/a	n/a	Cache-inhibited **lwarx** causes DSI exception.
	W = 1	n/a	none	n/a	n/a	Write-through **lwarx** causes DSI exception.

Table 3-10. L1 Cache-State Transitions and MSS Requests (continued)

Internal Operation	WIM Setting	Initial L1 State	MSS Request	MSS Response	Final L1 State	Comments
Store	W = 0 I = 0	I	Store	n/a	M	If L1 = I, deallocate a cache line and reload data from MSS. If L1 = S, invalidate and allocate line before initiating RWITM. Store data is merged with reload data.
	M = 0 W = 0 I = 0	S	none	n/a	M	
	M = 1 W = 0 I = 0	S	Store	n/a	M	
	For the MPC7448: M = X W = 0 I = 0	S	Store	n/a	M	
	W = 0 I = 0	E/M	none	n/a	M	Merge store data into L1.
	I = 1	n/a	Store	n/a	n/a	Initiate a store request to the MSS without changing cache state.
	W = 1	I/S/E/M	Store	n/a	same	If line is valid, merge store data into L1. Initiate a store request to MSS.
stwcx.	W = 0 I = 0	I	STWCX	E	M	Same as stores, but do not store data if reservation is not set. Reset reservation when past coherency point. Return whether successful. Note: a **stwcx.** which loses its reservation while pending in the MSS is converted into a load and possibly returns a shared response.
				S	S	
	M = 0 W = 0 I = 0	S	none	n/a	M	
	M = 1 W = 0 I = 0	S	STWCX	E	M	
				S	S	
	W = 0 I = 0	E/M	none	n/a	M	
	I = 1	n/a	none	n/a	n/a	Cache-inhibited **stwcx.** causes a DSI exception.
	W = 1	n/a	none	n/a	n/a	Write-through **stwcx.** causes DSI exception.
dcbst	x	I/S/E	DCBST	n/a	I	Push any modified data out to memory. Change cache line to invalid if it was valid.
		M	Write w/Clean	n/a	I	
dcbf	x	I/S/E	DCBF	n/a	I	Push any modified data out to memory and leave cache line invalid.
		M	Castout (W = 1)	n/a	I	

Table 3-10. L1 Cache-State Transitions and MSS Requests (continued)

Internal Operation	WIM Setting	Initial L1 State	MSS Request	MSS Response	Final L1 State	Comments
dcbz	I = 0 W = 0	I	DCBZ	n/a	M	Zero out data
	M = 0 I = 0 W = 0	S	none	n/a	M	
	M = 1 I = 0 W = 0	S	DCBZ	n/a	M	Claim ownership of line without reloading data. Write all 0’s to cache.
	For the MPC7448: M = X I = 0 W = 0	S	DCBZ	n/a	M	
	I = 0 W = 0	E/M	none	n/a	M	Zero out data.
	W = 1	I/S/E/M	none	n/a	same	Write-through or cache-inhibited **dcbz** causes alignment exception.
	I = 1	n/a	none	n/a	n/a	
dcba	I = 0 W = 0	I	DCBZ	n/a	M	Same as **dcbz** for I = 0,W = 0.
	M = 0 I = 0 W = 0	S	none	n/a	M	—
	M = 1 I = 0 W = 0	S	DCBZ	n/a	M	Claim ownership of line without reloading data. Write all 0’s to cache.
	For the MPC7448: M = X I = 0 W = 0	S	DCBZ	n/a	M	
	I = 0 W = 0	E/M	none	n/a	M	—
	W = 1	n/a	none	n/a	n/a	Write-through or cache-inhibited **dcba** is a no-op.
	I = 1	n/a	none	n/a	n/a	
L1 Deallocate	x	I/S/E	none	n/a	I	Deallocate is caused by an operation to another address (e.g. load) requiring an allocation of a cache line. Cast out modified data and invalidate line.
	x	M	Castout (W = 0)	n/a	I	
icbi	x	n/a	ICBI	n/a	n/a	No effect on D cache.

Table 3-10. L1 Cache-State Transitions and MSS Requests (continued)

Internal Operation	WIM Setting	Initial L1 State	MSS Request	MSS Response	Final L1 State	Comments
tlbie	x	n/a	TLBIE	n/a	n/a	No effect on L1 caches.
tlbsync	x	n/a	TLBSYNC	n/a	n/a	
sync	x	n/a	SYNC	n/a	n/a	
eieio	x	n/a	EIEIO	n/a	n/a	
eciwx	x	n/a	ECIWX	n/a	n/a	
ecowx	x	n/a	ECOWX	n/a	n/a	

3.6 L2 Cache

This section provides information about the on-chip L2 cache on the MPC7450. It describes the L2 cache organization, the L2 features and how they are controlled, L2 cache operation, and provides a summary of all actions of the L2 and L3 caused by internal operations in a summary table. See Section 3.8.4.3, “L2 and L3 Operations Caused by External Snoops,” for more information about the L2 cache and bus snooping.

3.6.1 L2 Cache Organization

The integrated L2 cache is organized as shown in Figure 3-17.

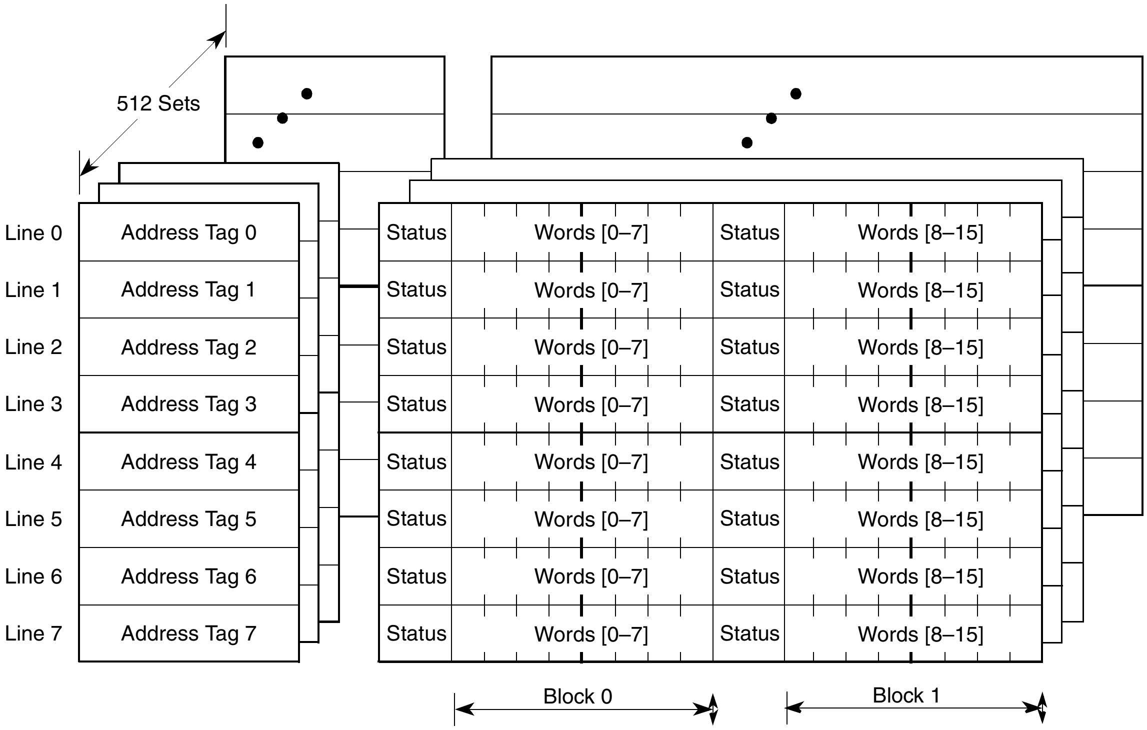


Figure 3-17. L2 Cache Organization for MPC7450

The L2 cache organization for the MPC7447 and MPC7457 is shown in Figure 3-18.

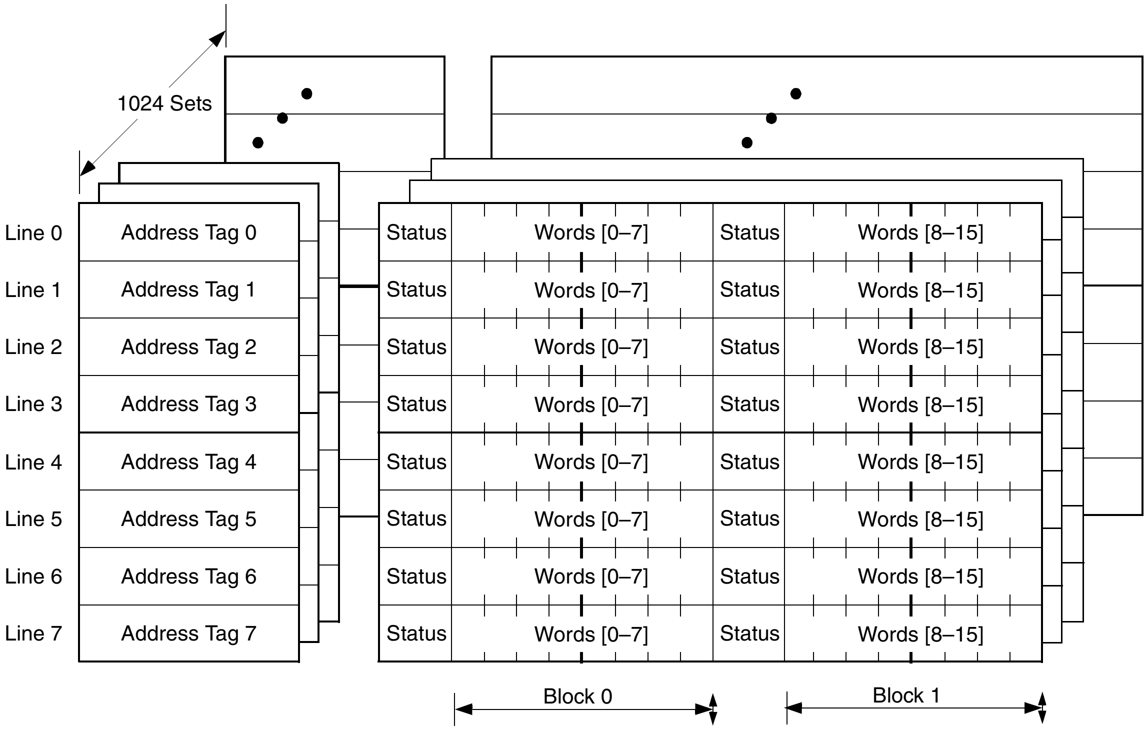


Figure 3-18. L2 Cache Organization for the MPC7447 and MPC7457

The L2 cache organization for the MPC7448 is shown in Figure 3-19.

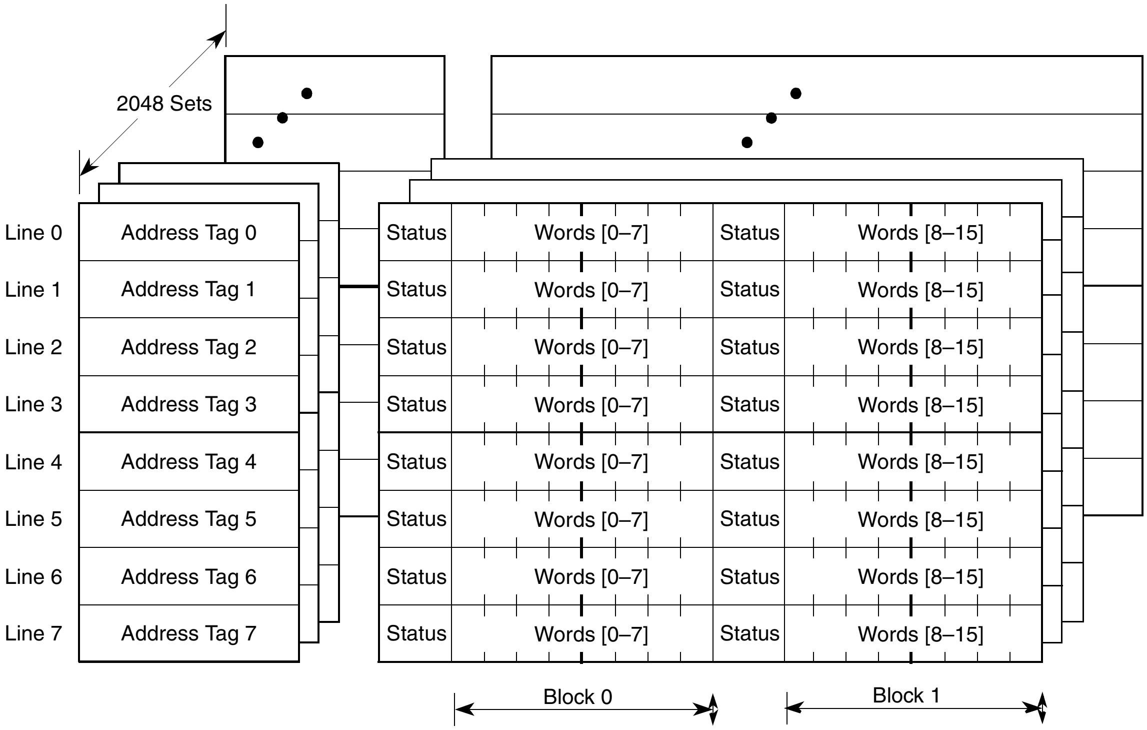


Figure 3-19. L2 Cache Organization for the MPC7448

Each line consists of 64 bytes of data, organized as two blocks (also called sectors) that are selected by one address bit. Although all 16 words in a cache line share the same address tag, each block maintains the two separate status bits for the 8 words of the cache block, the unit of memory at which coherency is maintained. Thus, each cache line can contain 16 contiguous words from memory that are read or written as 8-word operations. Note that the line replacement information for the L2 cache is maintained on a line basis.

The L2 cache tags are fully pipelined and non-blocking for efficient operation. Thus, the L2 cache can be accessed internally while a load for a miss is pending (allowing hits under misses). A reload for a cache miss is treated as a normal access and blocks other accesses for only a single cycle.

Similar to the L1 data cache, there are two status bits associated with each cache block of the L2 cache. These bits are used to implement the modified/exclusive/shared/invalid (MESI) cache coherency protocol. The coherency protocols are described in Section 3.3, “Memory and Cache Coherency.”

3.6.2 L2 Cache and Memory Coherency

The MPC7450 models for memory and cache coherency described in Section 3.3, “Memory and Cache Coherency,” for the L1 caches all apply for the L2 cache. Specifically, the WIMG bit model, the MESI cache coherency protocol, and the architectural implications of the ordering of loads and stores are as described in that section.

3.6.3 L2 Cache Control

The parameters for the L2 cache are controlled by L2CR, MSSCR0, and MSSSR0.

3.6.3.1 L2CR Parameters

The L2CR enables the L2 cache, enables parity checking on the L2, provides for instruction-only and data-only modes, provides hardware flushing for the L2, and selects between two available replacement algorithms for the L2 cache. L2CR is a supervisor-level read/write, implementation-specific register that is accessed as SPR 1017. The contents of L2CR are cleared during power-on reset. Refer to Section 2.2.5.5.1, “L2 Cache Control Register (L2CR),” for the bit descriptions of L2CR.

3.6.3.1.1 Enabling the L2 Cache and L2 Initialization

When the L2 cache is disabled, all accesses bypass the L2. Before the L2 cache is enabled, all L2 cache configurations must be set appropriately in L2CR and the L2 tags must be invalidated in the following sequence:

1. Verify that L2CR[L2E] = 0.
2. Invalidate the entire L2 cache by setting L2CR[L2I]. See Section 3.6.3.1.4, “L2 Cache Invalidation.”
3. Poll L2CR[L2I] until it is cleared.
4. Set remaining desired bits in L2CR and then set L2CR[L2E].

The L2 cache is disabled out of reset, so L2CR[L2E] = 0. Note that out of reset, the sequence above must obviously be preceded by the assertion and negation of $\overline{\text{HRESET}}$ per the timing requirements in the hardware specifications.

Setting L2CR[L2E] enables operation of the L2 cache, including snooping of the L2. Note that the **dcbf**, **dcbst**, and **dcbi** instructions have no effect on the L2 cache when it is disabled.

3.6.3.1.2 Enabling L2 Parity Checking

The L2 cache maintains one parity bit per byte of data and an additional parity bit for each tag (one tag parity bit per line).

In the MPC7450, L2 cache parity checking is enabled by setting L2CR[L2PE]. When L2CR[L2PE] = 1, L2 tag and data parity bits are independently generated and checked. When a parity error occurs for either the L2 address or data buses, a machine check exception is generated if MSR[ME] = 1. Otherwise, if MSR[ME] = 0, a checkstop occurs. Note that as a result of a machine check exception caused by an L2 or L3 parity error, SRR1[11] is set, and enabled L2 tag and data parity errors are reported in the L2TAG and L2DAT bits of MSSSR0. See Section 3.6.3.3, “L2 Parity Error Reporting,” and Section 2.2.5.4, “Memory Subsystem Status Register (MSSSR0),” for detailed information on the MSSSR0 bits.

With the addition of L2 data ECC support in the MPC7448, setting L2CR[L2PE] enables only data parity checking. Tag parity checking is enabled separately in the error disable register (L2ERRDIS). See the L2CR field descriptions in Table 2-12 for detailed information on changes to L2 parity checking in the MPC7448 as reflected in the L2CR[L2PE] bit. When an L2 data or tag parity error occurs, a machine check exception is generated if MSR[ME] = 1. Otherwise, if MSR[ME] = 0, a checkstop occurs. Enabled L2 tag parity errors are reported in the TPARERR bit of the error detect register (L2ERRDET) as well as in MSSSR0. See Section 3.6.3.4, “L2 Data ECC (MPC7448-Specific),” for information on ECC support in the MPC7448.

NOTE

> If ECC is enabled, setting L2CR[L2PE] has no effect; ECC checking will continue.

3.6.3.1.3 L2 Instruction-Only and Data-Only Modes

The L2CR also maintains the L2IO and L2DO bits for limiting the types of new accesses that are allocated into the L2. When L2CR[L2IO] is set, only instruction accesses that miss in the L2 allocate new entries in the L2. Data accesses that hit (loads and stores) operate normally (except for the case of store hits to blocks marked shared that actually function as misses). When L2CR[L2DO] is set, only data accesses that miss in the L2 allocate new entries in the L2. Instruction accesses that are already resident in the L2 (allocated before L2DO was set) provide instructions normally.

If both L2IO and L2DO are set, the L2 is effectively locked, and no new entries are allocated.

3.6.3.1.4 L2 Cache Invalidation

The L2 cache can be globally invalidated by setting L2CR[L2I]. This causes all valid bits in the L2 cache to be cleared. The L2CR[L2I] bit must not be set while the L2 cache is enabled (L2CR[LE] = 0). When the MPC7450 completes the invalidation, L2CR[L2I] is automatically cleared.

When software sets L2CR[L2I], the L2 cycles through all the tags and invalidates every entry in the cache without regard to the state of the line. The processor clears L2CR[L2I] upon completing the invalidation of the entire cache. Software can poll L2CR[L2I] to know when the invalidation is complete.

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

1. Execute a **dssall** instruction to cancel any pending data stream touch instructions.
2. Execute a **sync** instruction to finish any pending store operations in the load/store unit, disable the L2 cache by clearing L2CR[L2E], and execute an additional **sync** instruction after disabling the L2 cache to ensure that any pending operations in the L2 cache unit have completed.
3. Initiate the global invalidation operation by setting the L2CR[L2I] bit.
4. Monitor the L2CR[L2I] bit to determine when the global invalidation operation is completed (indicated by the clearing of L2CR[L2I]). The global invalidation requires approximately 8K core clock cycles to complete.
5. After detecting the clearing of L2CR[L2I], re-enable the L2 cache for normal operation by setting L2CR[L2E].

3.6.3.1.5 Flushing of L1, L2, and L3 Caches

The MPC7450 provides a hardware flushing mechanism for the L2 through the L2CR[L2HWF] bit. Note that prior to flushing the caches, L2 prefetching must be disabled (MSSCR0[L2PFE] = 0). When L2CR[L2HWF] is set, the L2 begins a flush by starting with the first cache index. Each modified block (sector) is cast out as it is flushed. After the first line in the first way is flushed (one block and then the other), the next way (same index) is flushed. When all ways for a given index have been flushed, the index is incremented and same process occurs for line 1, and so on.

During a hardware flush, the L2 services both read hits and bus snooping.

The hardware flush completes when all blocks in the L2 have a status of invalid. At this time, the processor automatically clears L2CR[L2HWF]. However, even though the hardware flush is considered complete, there may still be outstanding castouts queued in the L2SQ that need to be performed to the L3 and outstanding castouts in the BSQ waiting to be performed to the system interface.

Note that to guarantee that the L2 is completely invalid when flushing is complete, software must ensure that the L2 does not allocate new entries while the L2 is being flushed, by setting both L2CR[L2IO] and L2CR[L2DO] to lock the L2 cache.

The L2CR[L2I] invalidation is a subset of the L2CR[L2HWF] flushing mechanism. Note that some hardware resources are shared between the L2 and the L3 cache for supporting the hardware assisted flushing/invalidation features. This means that the MPC7450 cannot support simultaneous flushing/invalidation of both caches. Thus these must be done serially. The following sequence of steps is recommended for flushing the L1, L2, and L3 caches in the MPC7450:

1. Disable external interrupts (clear MSR[EE] to guarantee that the PLRU for the L1 is undisturbed by an interrupt handler).
2. Disable the L2 prefetching (clear MSSCR0[L2PFE]).
3. Flush the L1 data cache as described in Section 3.5.8, “L1 Cache Invalidation and Flushing.”
4. Set the L2CR[L2IO] and L2CR[L2DO] bits to completely lock the L2 cache.

5. Perform an **mtspr** L2CR to set L2HWF.
6. Poll the L2CR[L2HWF] bit using **mfspr** L2CR until L2CR[L2HWF] is cleared. When the bit is cleared, issue a **sync**. Although not necessary, the **sync** helps to clear the store queues in the memory subsystem before getting started with the L3 flushing. At this point the L2CR[L2IO] and L2CR[L2DO] bits can be cleared.
7. Set the L3CR[L3IO] and L3CR[L3DO] bits to completely lock the L3 cache.
8. Perform an **mtspr** L3CR to set L3HWF. See Section 3.7.3.7, “L3 Cache Flushing.”
9. Poll L3CR[L3HWF] using **mfspr** L3CR until it is cleared. When the bit is cleared, issue a **sync**. Although not necessary, the **sync** helps to clear the store queues in the memory subsystem. At this point the L3CR[L3IO] and L3CR[L3DO] bits can be cleared.

Also note that because the MPC7450 shares the invalidation and flushing logic internally, it is a programming error to set more than one of the following fields in the L2CR and L3CR at a time: L2I, L2HWF, L3I, or L3HWF. Setting more than one of these bits at any one time can cause one or both caches to not fully invalidate.

3.6.3.1.6 L2 Replacement Algorithm Selection

The L2 cache supports two pseudo-random modes of line replacement, selected by L2CR[L2REP]—3-bit counter mode, and pseudo-random number generator mode. See Section 3.6.4.4, “L2 Cache Line Replacement Algorithms,” for a detailed description of the two L2 replacement algorithms.

3.6.3.2 L2 Prefetch Engines and MSSCR0

Depending on the application, it may enhance performance to prefetch the second block of an L2 cache line on a cache line miss, even if no data in the second block is currently required. In this case, from one to three prefetch engines can be enabled to fill invalid blocks (that share a line with a valid block) in the L2 cache.

The L2 prefetch engines are enabled through the MSSCR0[L2PFE] field. Note that it is an error to enable the prefetch engines when the L2 cache is disabled. When prefetching is enabled, a prefetch is initiated when a load, instruction fetch, or write-back store misses in all the caches and the transaction must be performed to the external system interface for the required block. In this case, a prefetch is initiated to fill the second (unrequired) block, provided an enabled prefetch engine is available.

However, prefetches are not initiated if:

- The access is a data cache miss and the L2 cache is set up to cache instructions only (L2CR[L2IO] = 1) or
- The access is an instruction cache miss and the L2 cache is set up to cache data only (L2CR[L2DO] = 1).

Note that the L2 prefetches are also loaded into the L3 cache if it is enabled. Also note that prior to flushing the caches MSSCR0[L2PFE] must be cleared, see Section 3.6.3.1.5, “Flushing of L1, L2, and L3 Caches,” for further details on how to flush the caches.

3.6.3.3 L2 Parity Error Reporting

When L2 cache parity checking is enabled, L2 tag and data parity bits are independently generated and checked. Enabled L2 tag and data parity errors are reported in the MSSSR0[L2TAG] and MSSSR0[L2DAT] register fields. In the MPC7448, in addition to MSSSR0[L2TAG], the enabled L2 tag parity errors are also reported in L2ERRDET[TPARERR]. See Section 3.6.3.1.2, "Enabling L2 Parity Checking," for more information.

3.6.3.4 L2 Data ECC (MPC7448-Specific)

In the MPC7448, L2 error detection, reporting, and injection allow flexible handling of ECC and parity errors in the L2 data and tag arrays. When the MPC7448 detects an L2 error, the appropriate bit in the error detect register (L2ERRDET) is set. Error detection is disabled by setting the corresponding bit in the error disable register (L2ERRDIS). By default, tag parity and data ECC checking are enabled on the MPC7448. However, for a detected error to signal a 0x200 machine check, the user must enable error reporting in error interrupt enable register (L2ERRINTEN) for each error.

Whereas setting L2CR[L2PE] in the MPC7447A enables both tag and data parity checking, tag parity in the MPC7448 is controlled separately through the L2ERRDIS[TPARDIS] bit. Data parity in the MPC7448 can only be enabled through L2CR[L2PE] if ECC is disabled in the L2ERRDIS register.

3.6.3.4.1 Enabling or Disabling ECC

The L2 cache must be disabled and flushed before enabling or disabling ECC to ensure that no errors occur. ECC can be enabled or disabled with the L2ERRDIS register. Refer to Section 2.2.5.5.9, "L2 Error Disable Register (L2ERRDIS)—MPC7448-Specific," for details on enabling or disabling ECC in the L2ERRDIS register.

3.6.3.4.2 L2 Error Control and Capture

The error control and capture registers control the detection and reporting of tag parity and ECC errors in the MPC7448. Error reporting (by generating an interrupt) is enabled by setting the corresponding bit in the error interrupt enable register (L2ERRINTEN). Note that the error detect bit is set regardless of the state of the interrupt enable bit.

Single-bit errors are corrected as data is read from the L2 cache and the corrected data is forwarded to the appropriate destination. Note that the copy of the data stored in the L2 cache is not corrected. Because single-bit errors may accumulate in the L2 over time and become multiple-bit errors, software may choose to periodically flush the L2 (single-bit errors are corrected as they are flushed from the L2) or take other measures to correct data stored in the L2 once a certain number of single-bit errors are detected. The L2 error control register (L2ERRCTL) maintains the L2 cache threshold for the number of ECC single-bit errors to be detected before reporting an error condition and a count of the number of ECC single-bit errors that have already been detected. If the count reaches the threshold and single-bit error reporting is enabled (SBECCDIS = 0 and SBECCINTEN = 1), an error is reported.

Double-bit errors are detected but not corrected. In most instances, errors of three or more bits are detected; however, it is not guaranteed that multiple-bit errors will always be detected. For all multiple-bit errors, including double-bit errors, note that the erroneous data is still forwarded to its destination, and it is software's responsibility to take appropriate corrective action if an error is reported. The double word where an error resides is always known because error detection is performed on a double-word boundary.

The data, address, and attributes of the first detected error that causes an error to be reported are saved in the error capture registers (L2ERRADDR, L2ERRATTR, L2CAPTDATAHI, L2CAPTDATALO, and L2CAPTECC). These registers are updated every time an error is detected, but are frozen after the detection of an error that causes an error to be reported (L2ERRATTR[VALINFO] = 1). Subsequent errors set error bits in the error detection registers, but only the first error that was reported (by generating an interrupt) has the associated information captured, until software unlocks error capture by clearing L2ERRATTR[VALINFO] to 0. L2CAPTDATAHI and L2CAPTDATALO hold the high and low words of the L2 data that contains the detected error. The calculated ECC syndrome and datapath ECC of the failing double word are saved in the L2 error syndrome register (L2CAPTECC). The type of error detected (multiple L2, tag parity, multiple-bit ECC, single-bit ECC) is reflected in L2ERRDET. Tag parity, multiple-bit, and single-bit error detection can be enabled or disabled in L2ERRDIS.

L2ERRDET is implemented as a bit-reset type register. Reading from this register occurs normally; however, write operations can clear but not set bits. A bit is cleared whenever the register is written and the data in the corresponding bit location is a 1. For example, to clear bit 6 and not affect any other bits in the register, the value 0x0200_0000 is written to the L2ERRDET register.

3.6.3.4.3 ECC Error Reporting

In the MPC7448, when ECC is enabled and an ECC error is detected, the error is reported in both MSSSR0 and L2ERRDET. The ECC error is reported in the MSSSR0[L2DAT] field, as well as either L2ERRDET[SBECCERR] or L2ERRDET[MBECCERR], depending on whether the error is a single-bit or multiple-bit error.

3.6.3.4.4 L2 Error Injection

The L2 cache in the MPC7448 includes support for injecting errors into the L2 data, data ECC or tag. This may be used to test error recovery software by deterministically creating error scenarios.

The preferred method for using error injection is to map all blocks or pages cache-inhibited (WIMG = x1xx), except for one that will be marked cacheable (WIMG = x0xx) to be used as a temporary scratch buffer, set L2CTL[L2DO] to prevent allocation of instruction accesses, and invalidate the L2 by setting L2CTL[L2I] = 1. The following code sequence triggers an error, and then detects it (A is an address in the scratch page):

```
dcbz A          | allocates the line in the L2 in exclusive state
lwz A
```

Data or tag errors are injected into the line, per the error injection settings in L2ERRINJHI, L2ERRINJLO, and L2ERRINJCTL, at allocation. The final load detects and reports the error (if enabled) and allows software to examine the offending data, address, and attributes.

Note that error injection enable bits in L2ERRINJCTL must be cleared by software and the L2 must be invalidated (by setting L2CTL[L2I]) before resuming L2 normal operation.

3.6.3.5 Instruction Interactions with L2

The following instructions have effects on the L2 cache as listed:

- **dcbz** and **dcba** instructions that miss or hit as shared cause L2 allocation to reserve the line and a kill is sent to the L3 and external bus interface. When the kill completes, the L2 line is marked exclusive. **dcbz** instructions that hit as modified or exclusive cause no L2 state change.
- On the MPC7450, **dcba** differs from **dcbz** only in its exception generation. As such, it is identical to **dcbz** from an L2 perspective.
- Line pushes from the L1 data cache as the result of **dcbf**/**dcbst** instructions write through to the L3 and external bus interface. **dcbf** invalidates the L2 cache block in case of hit. A **dcbst** hit does not affect the block if it hits as either shared or exclusive; it is changed to exclusive if it hits as modified.
- **dcbf**/**dcbst** instructions that do not require a castout from the L1 data cache are issued to the L2 cache and perform an invalidate and/or castout from the L2 cache to the L3 as required. If they do not require a castout from the L2 cache, they are also issued to the L3.
- **dcbf** and **dcbi** instructions that address an area of memory marked with M = 1 cause a global transaction on the system bus if the line is modified or if HID1[ABE] is set.
- **icbi** instructions bypass the L2 cache and are forwarded to the L3.
- **sync** and **eieio** instructions bypass the L2 cache, and are forwarded to the L3 for further processing. Also, all **sync** and **eieio** instructions are broadcast on the system bus if HID1[SYNCBE] = 1.
- **eciwx**, **ecowx**, **tlbie**, and **tlbsync** instructions bypass the L2 cache, and are forwarded to the system interface for further processing.
- **dcbf**, **dcbst**, **dcbi**, **icbi**, **tlbie**, and **tlbsync** instructions are broadcast on the system bus if HID1[ABE] = 1.

3.6.4 L2 Cache Operation

This section describes the MPC7450 L2 cache operations.

All accesses to the L2 cache that are marked cache-inhibited bypass the L2 cache (even if they would have normally hit), and do not cause any L2 state changes. Note that all data accesses performed while the L1 data cache is disabled are considered cache-inhibited by the L2 cache and the rest of the memory subsystem. Therefore, all read accesses from the L2 cache are burst accesses (32-byte reads).

Single-beat writes occur to the L2 cache for the following:

- Write-through (W = 1) accesses that hit in the L2
- Stores that hit if all ways of the L1 cache are locked with LDSTCR[DCWL]
- Stores that hit if the L1 data cache is completely locked with HID0[DLOCK] = 1

For the MPC7448, single-beat writes only occur for stores ≥ 64 bits. Otherwise, the entire cache line is flushed, and the store goes to the bus.

In these cases, the writes also propagate to the L3 cache and the system interface. If the L2 cache state for the block is not modified, the cache is updated, but the status bits for the block are not changed.

In case of multiple pending requests to the L2 cache, the priorities are as shown in Table 3-11.

Table 3-11. L2 Cache Access Priorities

Priority	Type of Access
1	Snoop request
2	Reload into L2 or L1
3	L2 castout
4	Snoop push or data intervention
5	In the following order: a. Cacheable store miss in the L1 data cache b. Load miss in the L1 data cache c. Instruction miss in the L1 instruction cache
6	L1 castout

This section contains a detailed description of L2 and L3 actions caused by L1 requests. For more information on L2 and L3 actions caused by bus snooping, see Section 3.8.4.3, “L2 and L3 Operations Caused by External Snoops.”

3.6.4.1 L2 Cache Miss and Reload Operations

The MPC7450 L2 cache blocks are filled (sometimes referred to as a cache reload) from the L3 cache or the memory subsystem when cache misses occur for cacheable accesses, as described in Section 3.1.2, “Load/Store Unit (LSU),” and Section 3.1.3, “Memory Subsystem Blocks.”

As an L2 cache line is received from the bus (or L3) it is loaded into the L2 cache and marked according to the snoop response. If the reload requires a new line to be allocated in the L2 cache and the current line is modified, the modified line is castout from the L2 cache to the L3 cache at the time of the miss (not at the time of the reload).

Note that the L2 prefetch engines can be selected to fetch the second block of an L2 cache line, even if it is not required by the program. See Section 3.6.3.2, “L2 Prefetch Engines and MSSCR0,” for more information.

3.6.4.2 L2 Cache Allocation

Instruction cache misses in the L2 cache cause an L2 cache line to be allocated, provided the L2 cache is enabled and not marked as data-only (with the L2CR[L2DO] bit). Similarly, instruction cache misses in the L3 cache also cause an L3 cache line to be allocated, provided the L3 cache is enabled and not marked as data-only (with the L3CR[L3DO] bit).

Also, data accesses cause an L2 cache line to be allocated if the L2 misses and the L2 is enabled and not marked as instruction-only (with the L2CR[L2IO] bit). Also, data accesses cause an L3 cache line to be allocated if the L3 misses and the L3 is enabled and not marked as instruction-only (with the L3CR[L3IO] bit).

Write-back stores that miss in the L1 data cache but hit on an L2 cache block that is in the shared state are treated as store misses, causing a RWITM transaction to the L3 and the bus. In this case, the line is not deallocated, but it is reloaded as it is read from the L3 or the bus.

When the L1 data cache causes a castout and the L2 cache is enabled, the L2 cache does not allocate a new line for the castout if it misses. If the castout hits in the L2, the new castout data is written into the L2.

Transient accesses (caused by the **dstt**, **dststt**, **lvxl**, and **stvxl** instructions) are treated similarly to non-transient accesses, except that transient accesses do not cause entries to be allocated in either the L2 or L3 caches on a miss. However, when an L1 data cache miss occurs for a transient operation, and the L2 or L3 cache hits, the L2 and L3 cache states are updated appropriately.

3.6.4.3 Store Data Merging and L2

Write-through stores use byte enables in the L1 and L2 caches to merge the write data with the current cache contents (if it hits). If the L2 hits, the entire block is written to the L2 and the L2SQ (similar to a castout) for consumption by the L3 cache. If the L3 cache hits, the entire line is consumed in the L3. If the L2 misses and the write is for fewer than 32 bytes, the L3 block is flushed before the store is performed. In both cases, only the write data (and not the complete, merged L2 block) is written to the bus.

3.6.4.4 L2 Cache Line Replacement Algorithms

The two pseudo-random modes of line replacement for the L2 cache (selected by L2CR[L2REP]) are three-bit counter mode and pseudo-random number generator mode. The three-bit counter mode (when L2CR[L2REP] = 1) is based on a simple three-bit counter that is incremented on every clock cycle. When a miss occurs, the line in the way pointed to by the counter is chosen for replacement.

The pseudo-random number generator mode (when L2CR[L2REP] = 0) uses 16 latches that are clocked on every clock cycle as shown in Figure 3-20 with 3 XOR functions. The L2 cache uses the value in latches 4, 9, and 15 as the 3-bit value that selects the way for replacement.

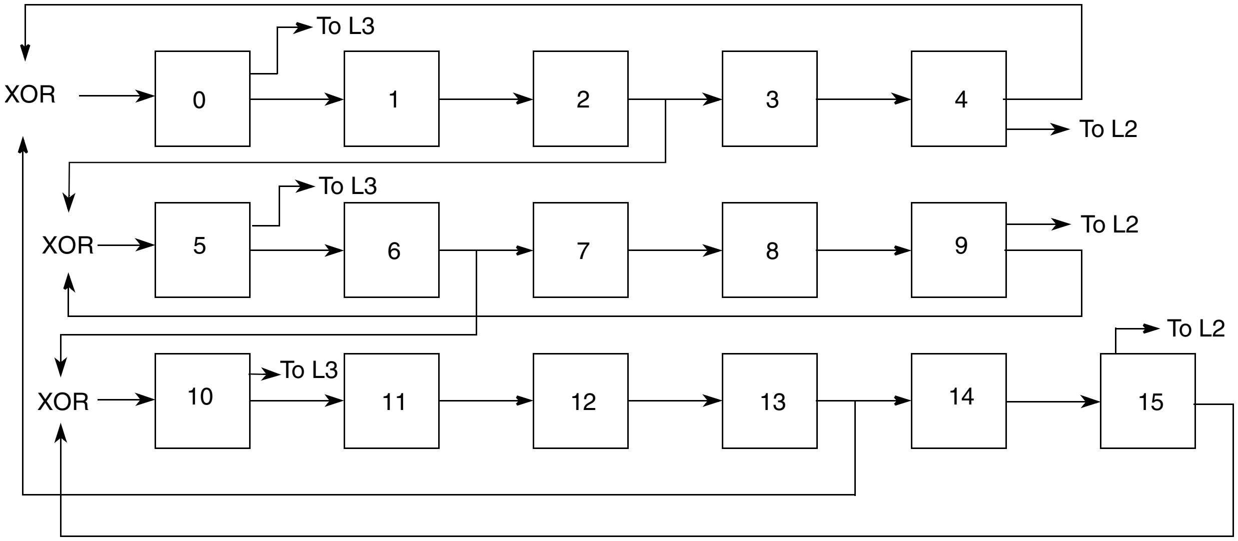


Figure 3-20. Random Number Generator for L2 (and L3) Replacement Selection

Due to the latency of the L2 cache look-up, there are 3 clock cycles between a read miss and the allocation of the replacement line. Thus, it would be possible that the same way can be chosen for replacement for two, or even three consecutive read misses with the algorithm as described above. In order to avoid this, the actual algorithm compares a selected replacement line with the three previous replacement lines. If the selected line matches with one of the three previous ones, a value of one, two, or three is automatically added to the value that selects the way for replacement.

Note that the L3 cache uses the same pseudo-random number generator logic for selecting replacement cache lines, but the L3 cache uses the values of three different latches for selecting the way for L3 replacement. See Section 3.7.7.4, “L3 Cache Replacement Selection,” for more information.

3.6.4.5 L2 and L3 Operations Caused by L1 Requests

This section contains a series of tables that define the actions of the L2 and L3 caches to service the L1 caches. See Section 3.8.4.3, “L2 and L3 Operations Caused by External Snoops,” for a description of L2 and L3 actions to service snoop requests.

Table 3-13 through Table 3-23 summarize all L2 and L3 cache activities and the internal conditions that cause them. Table 3-12 defines some of the abbreviations used in Table 3-13 through Table 3-23. Note the following:

- The WIM bits plus A (for atomic) are passed on to the memory subsystem unless they are overridden.
- The t (transient) indicator is also passed on to the memory subsystem.
- Any operation that requires an allocate in the L2 or L3 may fail to perform the allocate (whether due to a collision with a snoop, or due to the reload coming back faster than the allocate can arbitrate). In this case, the final state of the L2 or L3 cache will be the same as the initial state (I if invalid before, or S for store-hit_shared). For simplicity, the tables are written as if the allocate always succeeds.
- A **stwcx.** operation pending in the MSS that has not yet arbitrated or gone out on the bus may lose its reservation while it is pending. If this occurs, the RWITM atomic transaction on the bus is self-retried, and the operation is turned into a load operation. The MSS response to the L1 may be shared or exclusive depending on the bus response. For simplicity, these tables do not include that scenario, since it includes multiple transactions.

Table 3-12. Definitions for L2 and L3 Cache-State Summary

Term	Definition
L1 Snoop	The type of L1 snoop operation (if any) triggered by this MSS request.
MPX Bus Request	The MPX bus request (if any) triggered by this operation and its initial state. Any WIM setting in the MPX bus request type is a forced value (MMU WIM values are ignored).
Bus Response	The value of the shared snoop response (if applicable) to the MPX bus request.
Final L2 State	The MESI state of this address in the L2 cache after the operation completes. A represents the allocated state for retry conditions.
Final L3 State	The MESI state of this address in the L3 cache after the operation completes. A represents the allocated state for retry conditions.

Table 3-12. Definitions for L2 and L3 Cache-State Summary

Term	Definition
MSS Response to L1	If reloading the L1, whether the reload data is exclusive or shared.
SMC	Store miss complex. The series of queues that handle store misses.

Table 3-13. L2/L3 Cache State Transitions for Load, lwarx, Touch, and IFetches

WIM	Initial L2 State	Initial L3 State	MPX Bus Req	Bus Resp	Final L2 State	Final L3 State	MSS Resp to L1	Comments
I = 0 t = 0	I	I	Read W = 0	S	S	S	S	Forward the critical data to L1 (except touch). Reload L1, L2 and L3 from bus.
				E	E	E	E	
		S	none	n/a	S	same	S	Reload L1 and L2 from L3.
		E/M	none	n/a	E	same	E	
	S	I/S/E/M	none	n/a	same	same	S	Reload L1 from L2.
	E/M	I/S/E/M	none	n/a	same	same	E	
I = 0 t = 1	I	I	Read W = 0	S	same	same	S	Read or touch transient. Return reload data to L1, but don’t allocate or reload L2 or L3.
				E	same	same	E	
		S	none	n/a	same	same	S	
		E/M			same	same	E	
	S	I/S/E/M	none	n/a	same	same	S	
	E/M	I/S/E/M	none	n/a	same	same	E	
I = 1	n/a	n/a	Read W = 0	n/a	n/a	n/a	n/a	Bypass caches and perform cache-inhibited bus read.

Table 3-14. L2/L3 Cache State Transitions for Store Touch Operations

WIM	Initial L2 State	Initial L3 State	MPX Bus Req	Bus Resp	Final L2 State	Final L3 State	MSS Resp to L1	Comments
I = 0 t = 0	I/S	I/S	RClaim	S	S	S	S	Reload L1, L2 and L3 from bus.
				E	E	E	E	
		E/M	none	n/a	E	same	E	Reload L1 and L2 from L3.
	E/M	I/S/E/M	none	n/a	same	same	E	Reload L1 from L2.
I = 0 t = 1	I/S	I/S	RClaim	S	same	same	S	Store touch transient. Return reload data to L1, but don’t allocate or reload L2 or L3.
				E	same	same	E	
		E/M	none	n/a	same	same	E	
	E/M	I/S/E/M	none	n/a	same	same	E	

Table 3-15. L2/L3 Cache State Transitions for Store (and stwcx.) Operations

WIM	Initial L2 State	Initial L3 State	MPX Bus Req	Bus Resp	Final L2 State	Final L3 State	MSS Resp to L1	Comments
W = 0 I = 0 t = 0	I	I/S	RWITM	n/a	E	E	E	Reload L1, L2, and L3 from bus. Allocate in L3 over shared state.
		E/M	none	n/a	E	same	E	Reload L1 and L2 from L3.
	S	I/S	RWITM	n/a	E	E	E	Reload L1/L2/L3 from bus.
		E/M	none	n/a	E	same	E	Reload L1/L2 from L3.
	E/M	I/S/E/M	none	n/a	same	same	E	Reload L1 from L2.
W = 0 I = 0 t = 1	I	I	RWITM	n/a	same	same	E	Transient stores do not allocate/reload the L2/L3 caches.
		S	RWITM	n/a	same	same	E	If the L3 cache line gets flushed before the data comes back, then the line stays invalid.
		E/M	none	n/a	same	same	E	—
	S	I	RWITM	n/a	same	same	E	If the L2/L3 cache state gets flushed before reload, then the line stays invalid.
		S	RWITM	n/a	same	same	E	
		E/M	none	n/a	same	same	E	—
	E/M	I/S/E/M	none	n/a	same	same	E	—
W = 1	I	I/S/E	Write w/Flush	n/a	same	same/I	n/a	Flush L3 if <32 bytes of write-through data. If 32 bytes of write-through data, data is merged in L3 and tag state remains the same. Do write-through store on bus.
		M	Write w/Kill (W = 0, M = 0), Write w/Flush	n/a	same	same/I	n/a	
	S/E/M	I	Write w/Flush	n/a	same	I	n/a	Merge data into L2. Put L2 data into L2SQ. Do write-through store of unmerged data on bus.
		S/E/M	Write w/Flush	n/a	same	same	n/a	Merge data into L2. Put L2 data into L2SQ. Write data into L3. Do write-through store of unmerged data on bus.
I = 1	n/a	n/a	Write w/Flush	n/a	n/a	n/a	n/a	Bypass L2 and L3 caches and do cache-inhibited store on bus.

Table 3-16. L2/L3 Cache State Transitions for Castout Operations

WIM	Initial L2 State	Initial L3 State	MPX Bus Req	Bus Resp	Final L2 State	Final L3 State	MSS Resp to L1	Comments
W = 0 M = 0	I	I	Write w/Kill (W = 0, M = 0)	n/a	same	same	n/a	Cast out L1 data to bus.
		S/E/M	none	n/a	same	M	n/a	Cast out L1 data to L3.
	S/E/M	I/S/E/M	none	n/a	M	same	n/a	Cast out L1 data to L2.
W = 1	I/S/E/M	I/S/E/M	Write w/Kill (W = 1, M = 0)	n/a	I	I	n/a	Push data from L1 for **dcbf**.

Table 3-17. L2/L3 Cache State Transitions for L2 Castout Operations

WIM	Initial L2 State	Initial L3 State	MPX Bus Req	Bus Resp	Final L2 State	Final L3 State	MSS Resp to L1	Comments
x	M	I	Write w/Kill (W = 0, M = 0)	n/a	I	same	n/a	Cast out L2 data to bus.
		S/E/M	none	n/a	I	M	n/a	Cast out L2 data to L3.

Table 3-18. L2/L3 Cache State Transitions for L3 Castout Operations

WIM	Initial L2 State	Initial L3 State	MPX Bus Req	Bus Resp	Final L2 State	Final L3 State	MSS Resp to L1	Comments
x	n/a	M	Write w/Kill (W = 0 M = 0)	n/a	n/a	I	n/a	Cast out L3 data to bus.

Table 3-19. L2/L3 Cache State Transitions for dcbf Operations

WIM	Initial L2 State	Initial L3 State	MPX Bus Req	Bus Resp	Final L2 State	Final L3 State	MSS Resp to L1	Comments
x	I	I/S/E	Flush	n/a	same	I	n/a	Invalidate L3.
		M	Write w/Kill (W = 1, M = 0)	n/a	I	I	n/a	Push data from L3 to bus.
	S/E	I/S/E	Flush	n/a	I	I	n/a	Invalidate L2 and L3.
		M	Write w/Kill (W = 1, M = 0)	n/a	I	I	n/a	Push data from L3 to bus.
	M	I/S/E/M	Write w/Kill (W = 1, M = 0)	n/a	I	I	n/a	Push data from L2 to bus

Table 3-20. L2/L3 Cache State Transitions for dcbz Operations

WIM	Initial L2 State	Initial L3 State	MPX Bus Req	Bus Resp	Final L2 State	Final L3 State	MSS Resp to L1	Comments
M = 0	I/S/E/M	I/S/E/M	none	n/a	same	same	E	No need to claim ownership. Synthesize L2 hit.
M = 1	I	I/S	Kill	n/a	E	E	E	Claim ownership for line.
		E/M	none	n/a	E	same	E	
	S	I/S	Kill	n/a	E	E	E	
		E/M	none	n/a	E	same	E	
	E/M	I/S/E/M	none	n/a	same	same	E	—

Table 3-21. L2/L3 Cache State Transitions for dcbst Operations

WIM	Initial L2 State	Initial L3 State	MPX Bus Req	Bus Resp	Final L2 State	Final L3 State	MSS Resp L1	Comments
x	I/S/E	I/S/E	Clean	n/a	same	same	n/a	—
		M	Write w/Kill (W = 1)	n/a	same	E	n/a	Push data from L3 to bus.
	M	I	Write w/Kill (W = 1, M = 0)	n/a	E	same	n/a	Push data from L2 to bus.
		S/E/M	Write w/Kill (W = 1)	n/a	E	E	n/a	Push data from L2 to bus, capturing it in L3.

Table 3-22. L2/L3 Cache State Transitions for Write with Clean Operations

WIM	Initial L2 State	Initial L3 State	MPX Bus Req	Bus Resp	Final L2 State	Final L3 State	MSS Resp to L1	Comments
x	I	I	Write w/Kill (W = 1, M = 0)	n/a	same	same	n/a	Push data from L1 to bus.
		S/E/M	Write w/Kill (W = 1, M = 0)	n/a	same	E	n/a	Push data from L1 to bus, capturing it in L3.
	S/E/M	I	Write w/Kill (W = 1, M = 0)	n/a	E	same	n/a	Push data from L1 to bus, capturing it in L2.
		S/E/M	Write w/Kill (W = 1, M = 0)	n/a	E	E	n/a	Push data from L1 to bus, capturing it in L2 and L3.

Table 3-23. L2/L3 Cache State Transitions for Remaining Instructions

MSS Op	Initial L2 State	Initial L3 State	MPX Bus Req	Bus Resp	Final L2 State	Final L3 State	MSS Resp to L1	Comments
icbi	n/a	n/a	ICBI	n/a	n/a	n/a	n/a	No action in L2/L3 cache.
tlbie	n/a	n/a	TLBIE	n/a	n/a	n/a	n/a	No action in L2/L3 cache.
tlbsync	n/a	n/a	TLBSYNC	n/a	n/a	n/a	n/a	No action in L2/L3 cache
sync	n/a	n/a	SYNC	n/a	n/a	n/a	n/a	**sync** causes ordering of previous and subsequent loads/stores from the same processor.
eieio	n/a	n/a	EIEIO	n/a	n/a	n/a	n/a	**eieio** causes ordering of certain loads and stores.

Table 3-23. L2/L3 Cache State Transitions for Remaining Instructions (continued)

MSS Op	Initial L2 State	Initial L3 State	MPX Bus Req	Bus Resp	Final L2 State	Final L3 State	MSS Resp to L1	Comments
eciwx	n/a	n/a	xferdata	n/a	n/a	n/a	n/a	**eciwx** bypasses L2 and L3 and performs a graphics read operation on the bus.
ecowx	n/a	n/a	xferdata	n/a	n/a	n/a	n/a	**ecowx** bypasses L2 and L3 caches and performs a graphics write operation on the bus.

3.7 L3 Cache Interface

This section describes the MPC7450 microprocessor L3 cache interface, and its configuration and operation. It describes how the MPC7450 signals, defined in Chapter 8, “Signal Descriptions,” interact to perform address and data transfers to and from the L3 cache. Note that the L3 cache is not supported by the MPC7441, MPC7445, MPC7447, MPC7447A, or MPC7448.

3.7.1 L3 Cache Interface Overview

The MPC7450’s L3 cache interface is implemented with an on-chip, eight-way set-associative tag memory with 2K tags per set, and a dedicated interface with support for up to 2 Mbytes of external synchronous SRAMs.

The tags are sectored to support either two or four cache blocks per tag entry, depending on the L3 cache size. Each sector (32-byte cache block) in the L3 cache has two status bits that are used to implement the MESI cache coherency protocol. Accesses to the L3 cache can be designated as write-back or write-through and the L3 maintains cache coherency through snooping. The processor uses critical double-word forwarding on the L3 cache interface.

The L3 interface can be configured to use 1 Mbyte or 2 Mbytes (or 4 Mbytes for the MPC7457) of the SRAM area as a private memory space. Accesses to private memory do not propagate to the system bus. The MPC7450 can also be configured to use the first 1 Mbyte of SRAM as L3 cache and the second 1 Mbyte as private memory. In this case, accesses to the private memory space do not propagate to the L3 cache (or the external system bus). For the MPC7457, the L3 can be configured to use the first 2 Mbytes of SRAM as L3 cache and the second 2 Mbytes as private memory.

The L3 cache control register (L3CR) provides control of L3 cache configuration, private memory control, and interface timing. The L3 private memory control register (L3PM) configures the private memory address range.

The L3 cache interface provides two clock outputs that allow the clock inputs of the SRAMs to be driven at select frequency divisions of the processor core frequency.

3.7.2 L3 Cache Organization

The L3 cache tags address four blocks (128 bytes) with each tag entry (line) when 2 Mbytes of external SRAM is used; they address two blocks (64 bytes) with each tag entry when 1 Mbyte of external SRAM is used. Each block maintains distinct coherency status bits and coherency is maintained in the same way as in the L2 cache. Also similar to the L2 cache, L3 entries are replaced on a line basis. Thus the organization is similar to that of the L2 cache (shown in Figure 3-17) when the L3 is configured for 1 Mbyte of SRAM, except that there are 2,048 sets. Additionally, when configured for 2 Mbytes of SRAM, there are twice as many blocks per line.

3.7.3 L3 Cache Control Register (L3CR)

The L3 cache control register (L3CR) controls the L3 cache configuration, timing, and operation. The following sections describe the L3 cache control parameters in the L3CR.

The L3CR is a supervisor-level read/write, implementation-specific register that is accessed as SPR 1018. The contents of L3CR are cleared during power-on reset. See Section 2.2.5.5.15, “L3 Cache Control Register (L3CR),” for additional information about the configuration of the L3CR.

The private memory feature of the MPC7450 is enabled with the L3CR[PMEN] and the size is determined by L3CR[PMSIZ]. These fields are described further in Section 3.7.8, “L3 Private Memory Operation.”

3.7.3.1 Enabling the L3 Cache and L3 Initialization

The L3 cache is enabled or disabled by programming the L3CR[L3E] parameter. This parameter enables or disables the operation of the L3 cache (including snooping) starting with the next transaction that the L3 cache unit receives. When the L3 cache is disabled, the cache tag status bits are ignored and all accesses are propagated to the system bus.

Following a power-on or hard reset, the L3 cache and the L3 clocks are disabled initially. Before enabling the L3 cache, the L3 clock must first be configured through the L3CR[L3CLK] and L3CR[CLKEN] bits, and a period of time must elapse. Also before enabling the L3 cache, all other bits in the L3CR must be set appropriately, and the L3 cache must be globally invalidated.

The sequence for initializing the L3 cache is as follows:

Verify that L3CR[L3E] = 0.

1. Set the L3CR[L3CLK] bits to the desired clock divider setting. All other L3 cache configuration bits should be set to properly configure the L3 cache interface for the SRAM type, size, and interface timing required, except do not set L3E, L3I, L3PE, or L3CLKEN.
2. Set L3CR[5] (otherwise reserved bit) to 1.
3. Set L3CR[L3CLKEN] to 1.
4. Wait for the L3 cache clocks to stabilize (100 processor cycles). This can be timed by setting the decrementer for a time period equal to the correct number of L3 cache clocks, or by performing an L3 cache global invalidate.

5. Perform an L3 cache global invalidate. The global invalidate could be performed before enabling the L3 clocks, or in parallel with waiting for the L3 clocks to stabilize. Refer to Section 3.7.3.6, “L3 Cache Invalidation,” for more information about L3 cache global invalidation. Note that a global invalidate always takes much longer than it takes for the L3 clocks to stabilize.
6. Clear L3CR[L3CLKEN] to zero.
7. Perform a **sync** instruction and wait 100 processor cycles.
8. Set the L3E and L3CLKEN bits of L3CR.
9. Perform a **sync** instruction and wait 100 processor cycles.

After the L3 clocks stabilize, an L3 cache global invalidate has been performed, and the other L3 cache configuration bits have been set, enable the L3 cache for normal operation by setting the L3CR[L3E] bit to 1.

Before the L3 cache is disabled it must be flushed to prevent coherency problems. The cache management instructions **dcbf**, **dcbst**, and **dcbi** do not affect the L1 data, L2 or L3 caches when the caches are disabled.

3.7.3.2 L3 Cache Size

The L3CR[L3SIZ] bit configures the size of the L3 cache and it should be set according to the organization of the L3 data RAMs that are present. Table 3-24 lists the data RAM organizations for the two L3 cache sizes noting that a 64/72-bit data bus size is always used. Table 3-24 also indicates typical SRAM sizes that might be used to construct such a cache

Table 3-24. L3 Cache Sizes and Data RAM Organizations for the MPC7450

L3 Cache Size	L3 Data RAM Organization	Example SRAM Sizes That Might Be Used
1 Mbyte (L3CR[L3SIZ] = 0)	128K x 64/72	(2) 128K x 32/36
2 Mbyte (L3CR[L3SIZ] = 1)	256K x 64/72	(2) 256K x 32/36

Notes:
The MPC7450 supports only one bank of SRAMs.
For very high speed operation, no more than two SRAMs should be used.

3.7.3.3 L3 Cache SRAM Types

The L3CR[L3RT] bits configure the L3 interface for the type of synchronous SRAMs that are used. The MPC7450 supports:

- MSUG2 dual data rate SRAMs that provide data synchronous to the L3_ECHO_CLK input signals to the MPC7450 and on each clock edge
- Late-write SRAMs which are required by the MPC7450 to be of the pipelined (register-register) configurations
- Pipeline burst SRAMs, referred to as PB2-type SRAMs

Note that the burst feature built into standard burst SRAMs and late-write SRAMs is not used by the MPC7450.

3.7.3.4 L3 Cache Data-Only and Instruction-Only Modes

Similar to the L2 cache, the L3 cache can be configured so that subsequent instruction accesses are not allocated into the L3 cache. Also, it can be configured so that subsequent data accesses are not allocated into the L3 cache. These instruction-and data-only features can be used together to effectively lock the contents of the L3 cache.

3.7.3.4.1 L3 Instruction-Only and Data-Only Operation

The L3CR maintains the L3IO and L3DO bits for limiting the types of new accesses that are allocated into the L3. When L3CR[L3IO] is set, only instruction accesses that miss in the L3 allocate new entries in the L3. Data accesses that hit (loads and stores) operate normally (except for the case of store hits to blocks marked shared that actually function as misses). When L3CR[L3DO] is set, only data accesses that miss in the L3 allocate new entries in the L3. Instruction accesses that are already resident in the L3 (allocated before L3DO was set) provide instructions normally.

3.7.3.4.2 L3 Cache Locking Using L3CR[L3DO] and L3CR[L3IO]

The MPC7450 L3 cache can be locked by setting both the L3DO and L3IO bits of the L3CR. This prevents instruction cache misses from reloading the L3 cache and prevents data cache misses (or store hits that are marked as shared) from allocating entries in the L3 cache. Note that locking the L3 cache using this mechanism is completely independent of L1 data or instruction cache and L2 cache locking.

3.7.3.5 L3 Cache Parity Checking and Generation

The L3CR[L3PE] parameter enables parity checking for the L3 data RAM interface. Additionally setting L3CR[APE] enables parity checking for the L3 address bus; L3CR[L3PE] and L3CR[APE] must both be set to enable L3 address bus parity checking. When L3PE is cleared, all L3 parity checking is disabled.

Note that the L3 interface always generates and drives parity on the L3DP[0:7] signals for writes to the SRAM array. The parity assignments for the L3DP[0:7] signals are as shown in Table 3-25.

Table 3-25. L3 Data Parity Signal Assignments

L3DP[0:7] Signal	L3[L3APE], L3[L3PE] = 01	L3[L3APE], L3[L3PE] = 11
L3DP[0]	L3DATA[00:07]	L3DATA[00:07], L3ADDR[16:18]
L3DP[1]	L3DATA[08:15]	L3DATA[08:15], L3ADDR[14:15]
L3DP[2]	L3DATA[16:23]	L3DATA[16:23], L3ADDR[12:13]
L3DP[3]	L3DATA[24:31]	L3DATA[24:31], L3ADDR[10:11]
L3DP[4]	L3DATA[32:39]	L3DATA[32:39], L3ADDR[08:09]
L3DP[5]	L3DATA[40:47]	L3DATA[40:47], L3ADDR[05:07]
L3DP[6]	L3DATA[48:55]	L3DATA[48:55], L3ADDR[02:04]
L3DP[7]	L3DATA[56:63]	L3DATA[56:63], L3ADDR[00:01]

L3CR[L3PE] also enables parity checking of the on-chip L3 tags and status bits. When L3CR[L3SIZ] = 0 (1 Mbyte of L3 cache), the 19 bits of L3 tag and one set of 3 status bits (22 bits total) are checked by one internal parity bit. Additionally, a second set of 3 status bits (for the second block) is checked by a second parity bit. When the L3 is configured for 2 Mbytes of cache, (L3CR[L3SIZ] = 1), the status bits for the third and fourth block are checked by two additional parity bits. All of these internal parity bits are set so that the bits being checked, plus the parity bit, contain an odd number of 1's.

When a parity error occurs for either the L3 address or data buses, or the internal tags and status bits, a machine check exception is generated if MSR[ME] = 1. If MSR[ME] = 0, a checkstop occurs. In the case of a machine check exception caused by an L2 or L3 parity error, SRR1[11] is set and MSSSR0 is set appropriately, to indicate which parity error caused the exception. Note that the MSSSR0 bits are set for parity errors even if MSR[ME] = 0 and no exception occurs. See Section 2.2.5.4, "Memory Subsystem Status Register (MSSSR0)," for more information on MSSSR0.

3.7.3.6 L3 Cache Invalidation

The MPC7450 supports invalidation of the L3 cache through the L3CR[L3I] parameter. Setting L3I causes a global invalidation of the L3 cache. The MPC7450 performs an invalidation by automatically sequencing through the L3 cache tags and clearing all the status bits for each tag. The global invalidation function must be performed only while the L3 cache is disabled. L3I must never be set while the L3 cache is enabled.

The L3 cache tags must be explicitly invalidated by software after a power-on or hard reset by setting the L3I bit.

L3CR[L3I] is automatically cleared when an L3 global invalidate is complete. It should be monitored after an L3 global invalidate has been initiated to determine when the global L3 invalidation has completed.

The sequence for performing a global invalidation of the L3 cache is as follows:

1. Execute a **dssall** instruction to cancel any pending data stream touch instructions.
2. Execute a **sync** instruction to finish any pending store operations in the load/store unit, disable the L3 cache by clearing L3CR[L3E], and execute an additional **sync** instruction after disabling the L3 cache to ensure that any pending operations in the L3 cache unit have completed.
3. Initiate the global invalidation operation by setting the L3CR[L3I] bit.
4. Monitor the L3CR[L3I] bit to determine when the global invalidation operation is completed (indicated by the clearing of L3CR[L3I]). The global invalidation requires approximately 8K core clock cycles to complete.
5. After detecting the clearing of L3CR[L3I], re-enable the L3 cache for normal operation by following the L3 initialization procedure described in Section 3.7.3.1, "Enabling the L3 Cache and L3 Initialization."

3.7.3.7 L3 Cache Flushing

The MPC7450 provides a hardware flush mechanism for the L3 cache through L3CR[L3HWF]. This hardware flush method is the recommended method for flushing the L3 cache. When the processor detects a state transition from 0 to 1 in L3HWF, the MPC7450 initiates a hardware flush of the L3 cache.

The flush is performed by starting with the lowest cache index and flushing all cache entries with that index through all the ways of the cache one way at a time until all ways are flushed. Thus, the next index is selected and the same process is repeated for all ways with that index. For each index and way of the cache, the processor generates a castout operation to the system bus for all modified cache blocks. At the end of the hardware flush, all lines in the L3 cache tags are in the invalid state. During the flush, read hits and snoops are fully serviced by the L3 cache.

When the L3 cache tags have been fully flushed of all valid entries, the L3CR[L3HWF] bit is automatically cleared. Note that when L3HWF is cleared, it does not guarantee that all lines from the L3 have been written completely to the system interface. L3 castouts may still be queued up in the bus interface unit. A final **sync** instruction is required to guarantee that all data from the L3 cache has been written to the system address bus.

Note that if the L3 must be guaranteed to be completely invalid when flushing is complete, software must ensure that the L3 does not allocate new entries while the L3 is being flushed by locking the L3 cache by setting L3CR[L3IO] and L3CR[L3DO].

Section 3.6.3.1.5, “Flushing of L1, L2, and L3 Caches,” contains procedures for flushing all of these caches and describes the serial requirements for flushing and invalidation of the L2 and L3 caches, as much of this logic is shared.

3.7.3.8 L3 Cache Clock and Timing Controls

The L3CR[L3CLK] parameter specifies the operating frequency for the L3 data RAM interface. This is expressed as a clock divider ratio relative to the MPC7450 core clock frequency. When L3CR[L3CLKEN] = 0, the L3 data signals are not driven or latched and the L3 clock outputs (L3_CLK[0:1]) are turned off. After setting the L3 clock ratio, a period of at least 100 processor clock cycles must elapse before enabling the L3 interface. Note that L3CR[L3CLK] should only be changed after L3CR[L3CLKEN] has been cleared for at least 100 processor clocks.

The SRAMS use the L3_CLK[0:1] signals to synchronously sample the address, control and write data signals. If DDR SRAMs are used, they drive a skewed version of the L3_CLK signals into the L3_ECHO_CLK[0:3] inputs of the MPC7450. The L3_ECHO_CLK[0:3] inputs are synchronous to the SRAM outputs. If PB2 or late-write SRAM are used, a feedback loop on the L3_ECHO_CLK signals is employed for synchronization; see the hardware specifications for more information. As the MPC7450 latches read data relative to L3_ECHO_CLK signals, it is synchronized to the processor clock using a first-in-first-out structure (FIFO) to eliminate metastability. When a beat of data is latched by the L3 interface, it is stored in the receive FIFO so that additional beats can be received even if the processor has not yet sampled the data and forwarded it to the L3 accumulator.

The L3CR[L3NIRCA] specifies the timing of L3_CLK[0:1] relative to the L3 address, data, and control buses. When L3CR[L3NIRCA] = 1, L3_CLK[0:1] is driven earlier relative to the L3 address, data, and control buses only when using non-integer frequency divider ratios. Setting L3CR[L3NIRCA] may be useful in a system requiring extra hold time on the L3 output signals. Note that MSSCR0[L3TCEN] (L3 turnaround clock enable) and MSSCR0[L3TC] (L3 turnaround clock count) allow a delay to be added between L3 reads and writes to allow the read/write mode switch to settle. This may be useful for troubleshooting systems when additional dead cycles between read and write transactions are desirable. In most cases, these bits should be cleared. Note that MSSCR0[L3TC] affects the read-to-write and write-to-read turn around.

3.7.3.9 L3 Sample Point Configuration

The L3CR[L3CKSP], L3CR[L3CKSPEXT], L3CR[L3PSP] bits specify the L3 and processor clock cycles in which the MPC7450 samples data from the receive FIFO on a read and loads the data into the L3 bus accumulator. In order to calculate the correct values of L3CR[L3CKSP] and L3CR[L3PSP] for internal sampling, the expected delays of L3_ECHO_CLK[0:3] must be estimated. Since these settings determine when the processor will forward data from the FIFO of the L3 data signals, incorrect settings may cause unpredictable and unrepeatable results, including data corruption and system instability. All of the following must be taken into account:

- Signal delays of the board
- For DDR, any delays between the reception of an L3_CLK edge by the SRAM and the generation of the corresponding L3_ECHO_CLK edge
- Offset of the external L3_CLK[*n*] pins with respect to the internal L3 clock
- Internal delays associated with the L3_CLK[*n*] and L3_ECHO_CLK[*n*] pins
- Access time of the L3 SRAM
- Number of data beats that must be valid before sampling can occur

For details on the L3_CLK offset and internal delays of L3_CLK[*n*] and L3_ECHO_CLK[*n*], see the hardware specifications.

Finally, L3CR[SPO] affects the L3 interface signal timing by adding one L3 clock cycle of latency on read operations when it is set. The L3CR[SPO] bit is reserved for future SRAM devices that may require the additional latency.

3.7.3.9.1 Pipeline Burst and Late-Write SRAM

One beat of data is sampled from the L3 accumulator in each L3 clock cycle for PB2 and late-write SRAM, so the FIFO must not be sampled until after the first data beat is valid. A core-to-L3 clock ratio of 4:1 is shown in this example. Since the first beat of data is valid in the FIFO on the third core clock within the second L3 clock period, the minimum sample point setting is L3CKSP = 2 and L3PSP = 3. In many systems, it may be necessary to allow additional time for the data to be valid. In these instances, sampling can be delayed by adding one or more core clocks to the sample point settings. Because of the nature of these settings, it is strongly recommended to use conservative sample point settings. The earliest recommended sample point is at least one core cycle after the earliest possible sample (L3CKSP = 3 and L3PSP = 0) as shown in Figure 3-21.

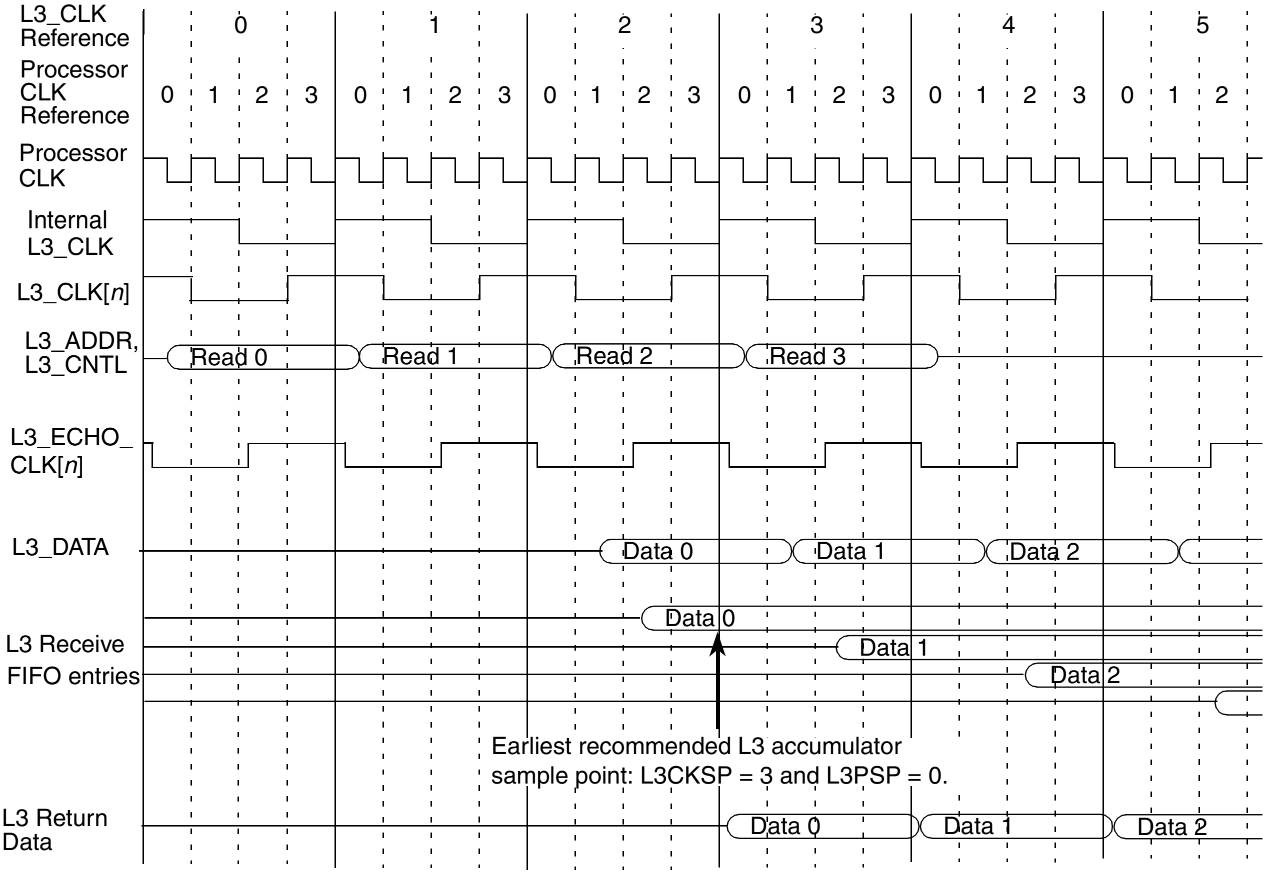


Figure 3-21. Example L3 Accumulator Sample Point Configuration for PB2 and Late-Write SRAM

3.7.3.9.2 MSUG2 DDR SRAM

Two beats of data are forwarded into the L3 accumulator in each L3 clock cycle for DDR SRAM. Because of this, the FIFO must not be sampled until after the second data beat is valid. A core-to-L3 clock ratio of 4:1 is shown in this example. Since the second beat of data is valid in the FIFO on the second core clock within the third L3 clock period, the minimum sample point setting is L3CKSP = 3 and L3PSP = 2. In many systems, it may be necessary to allow additional time for the data to be valid. In these instances, sampling can be delayed by adding one or more core clocks to the sample point settings. Because of the critical nature of these settings, it is strongly recommended to use conservative sample point settings The earliest recommended sample point is at least one core cycle after the earliest possible sample (L3CKSP = 3 and L3PSP = 3) as shown in Figure 3-22.

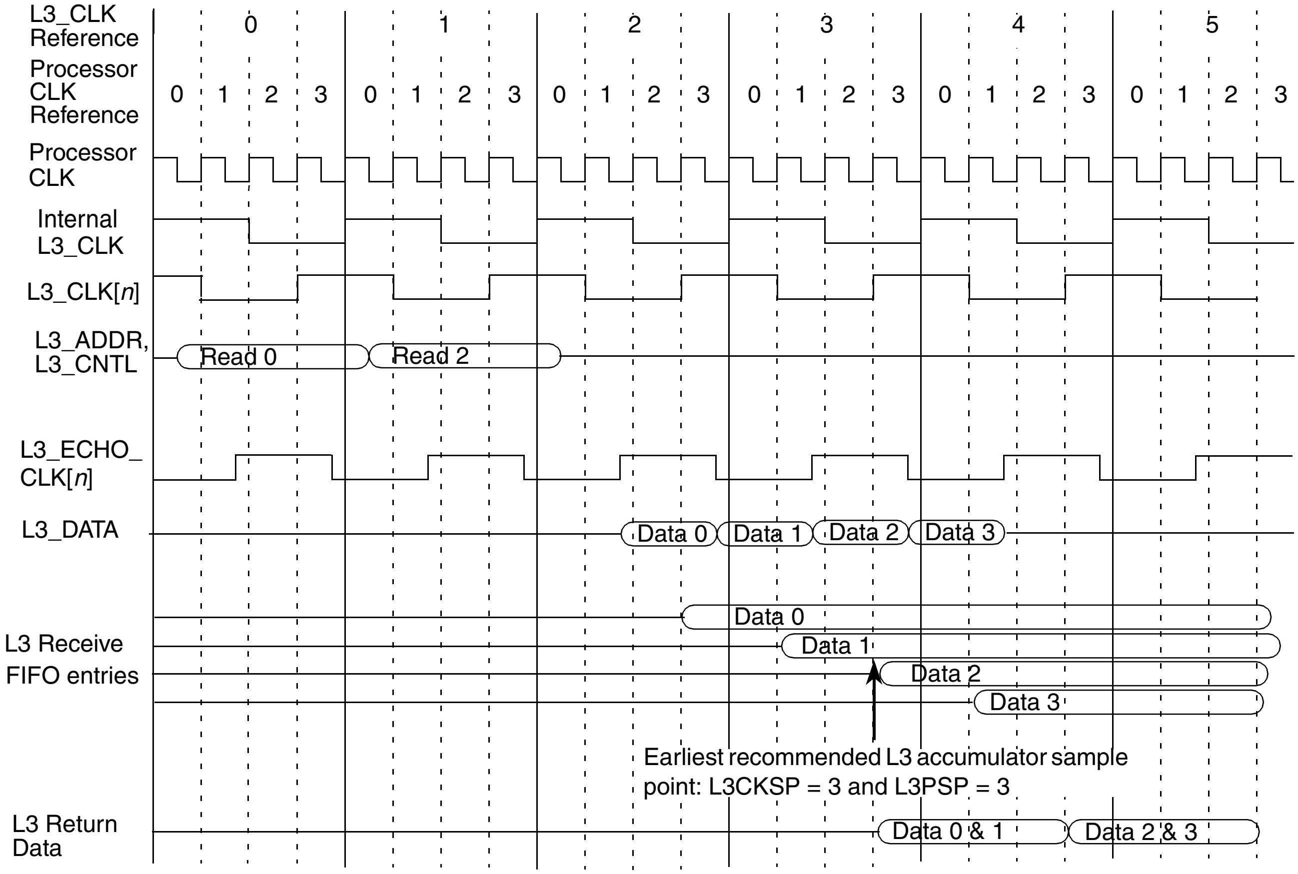


Figure 3-22. Example L3 Accumulator Sample Point Configuration for MSUG2 DDR SRAM

3.7.4 L3 Private Memory Address Register (L3PM)

The 16-bit L3PM[PMBA] parameter specifies the starting base address of the private memory of the L3 cache interface of the MPC7450 when it is enabled. The address is aligned to the appropriate block size. If the upper 16 bits of physical address (with extended addressing enabled (HID0[XAEN] = 1)) of a load, store or cache operation match the value in PMBA, the data is read or written from the external SRAMs. If extended addressing is disabled, the upper four bits of PMBA must be zero in order to be able to match the internal value of A0–A3 (which are zero). Note that transactions that hit in the private memory space are not visible on the external system bus.

Note also that either 1, 2, or 4 Mbytes of private memory can be specified. If 2 Mbytes of private memory are specified, only the upper 15 bits of the physical address are compared with [PMBA[0–14]. For 4 Mbytes of private memory, only the upper 14 bits of the physical address are compared with PMBA[0–13].

The L3PM is a supervisor-level read/write, implementation-specific register that is accessed as SPR 983. The contents of the L3PM are cleared during power-on reset. See Section 2.2.5.5.23, “L3 Private Memory Address Register (L3PM),” for information about programming the L3PM and see Section 3.7.8, “L3 Private Memory Operation,” for more information about enabling L3 private memory and the operation of this feature.

3.7.5 L3 Parity Error Reporting and MSSSR0

When L3 cache parity checking is enabled (L3CR[L3PE] = 1), L3 tag and data parity bits are independently generated and checked. Enabled L3 tag and data parity errors are reported in the L3TAG and L3DAT bits of MSSSR0. See Section 3.7.3.5, “L3 Cache Parity Checking and Generation,” and Section 2.2.5.4, “Memory Subsystem Status Register (MSSSR0),” for more information.

3.7.6 Instruction Interactions with L3

The following instructions have effects on the L3 cache as follows:

- **dcbz** and **dcba** instructions that miss or hit as shared cause L3 allocation to reserve the line and a kill is sent to the external bus interface. When the kill completes, the L3 line is marked exclusive. **dcbz** instructions that hit as modified or exclusive cause no L3 state change.
- On the MPC7450, **dcba** differs from **dcbz** only in its exception generation. As such, it is identical to **dcbz** from an L3 perspective.
- Line pushes from the L1 data cache as the result of **dcbf**/**dcbst** instructions write through to the external bus interface. **dcbf** invalidates the L3 cache block in case of hit. A **dcbst** hit does not affect the block if it hits as exclusive. If it hits as modified in the L3, then it is changed to exclusive. If it hits as shared in the L3 but it is modified in the L1 or L2, it is changed to exclusive.)
- **dcbf**/**dcbst** instructions that do not require a line push from the L1 data cache or L2 cache are issued to the L3 cache and perform an invalidate and/or castout from the L3 cache to the system bus as required. If they do not require a castout from the L3 cache, they are issued to the system bus as a flush (for **dcbf**) or clean (for **dcbst**).
- **dcbf** and **dcbi** instructions that address an area of memory marked with M = 1 cause a global transaction on the system bus if HID1[ABE] = 1.
- **icbi** instructions bypass the L3 cache and are forwarded to the system bus.
- **sync** and **eieio** instructions bypass the L3 cache, and are forwarded to the L3 for further processing. Also, all **sync** and **eieio** instructions are broadcast on the system bus if HID1[SYNCBE] = 1.
- **eciwx**, **ecowx**, **tlbie**, and **tlbsync** instructions bypass the L3 cache, and are forwarded to the system interface for further processing.
- **dcbf**, **dcbst**, **dcbi**, **icbi**, **tlbie**, and **tlbsync** instructions are broadcast on the system bus if HID1[ABE] = 1.

3.7.7 L3 Cache Operation

The MPC7450’s L3 cache is a combined instruction and data cache that receives memory requests from both L1 instruction and data caches and the L2 cache. The L1 requests are generally the result of instruction fetch misses, data load or store misses, L1 data cache castouts, write-through operations, or cache management instructions. Those requests are processed by the L2 cache and L3 cache in parallel. If the L2 cache misses, or requires further action from the memory subsystem, the L3 interface can service the request.

Each L1 miss request generates an address lookup in the L3 cache tags. If a hit occurs, the instructions or data are forwarded to the L2 cache and the appropriate L1 cache. A miss in the L3 cache tags causes the request to be forwarded to the system bus interface. The L3 cache also services snoop requests from the system bus.

See Section 3.6.4.5, “L2 and L3 Operations Caused by L1 Requests,” and Section 3.8.4.3, “L2 and L3 Operations Caused by External Snoops,” for more detailed information about the actions of the L3 caused by internal operations and snoops, respectively.

In case of multiple pending requests to the L3 cache, the priorities are as shown in Table 3-26.

Table 3-26. L3 Cache Access Priorities

Priority	Type of Access
1	Snoop request
2	Reload into L3
3	L3 castout
4	Snoop push or data intervention
5	L1 miss (data or instruction)

Note that a load, an instruction fetch or a cacheable store could gain access to the L2 cache based on the priorities shown in Table 3-11 but not gain access to the L3 cache based on the priorities of Table 3-26.

3.7.7.1 L3 Cache Miss and Reload Operations

Burst read requests from the L1 caches that miss in the L2 and L3 caches initiate a burst read operation from the system interface for the cache block that missed. If the L3 allocate requires a new tag entry and the current tag is modified, any modified sectors of the tag to be replaced are castout from the L3 cache to the system interface at the time of the miss. The cache block that is received from the bus is loaded into the L3 and forwarded to the L2 (and the appropriate L1 cache). L2 cache misses are also allocated into the L3.

3.7.7.2 L3 Cache Allocation

The L3 cache uses the same allocation principles as the L2 cache (as described in Section 3.6.4.2, “L2 Cache Allocation”). Thus, instruction cache misses in the L3 cache cause an L3 cache line to be allocated, provided the L3 cache is enabled and not marked as data-only (with the L3CR[L3DO] bit). Also, data accesses cause an L3 cache line to be allocated if the L3 misses and the L3 is enabled and not marked as instruction-only (with the L3CR[L3IO] bit).

Write-back stores that miss in the L1 data cache or L2 cache but hit on an L3 cache block that is in the shared state are treated as store misses, causing a RWITM transaction to the bus. In this case, the line is not deallocated, but it is reloaded as it is read from the bus.

L3 cache entries are not allocated for writes that miss in the L3. When the L1 data cache causes a castout, the L2 cache does not allocate a new line for the castout if it misses. If the L3 cache is disabled, then a block replaced from the L1 data cache or L2 cache is cast out to the system interface if the cache block is marked modified.

Transient accesses (caused by the **dstt**, **dststt**, **lvxl**, and **stvxl** instructions) are treated similarly to non-transient accesses, except that transient accesses do not cause entries to be allocated in either the L2 or L3 caches on a miss. However, when an L1 data cache miss occurs for a transient operation, and the L2 or L3 cache hits, the L2 and L3 cache states are updated appropriately.

3.7.7.3 $\overline{\text{CI}}$ and $\overline{\text{WT}}$ Accesses and L3

All requests to the L3 cache that are marked caching-inhibited bypass the L3 cache (even if they would have normally hit), and do not cause any L3 tag state changes.

Write-through stores that hit in the L2 cause the cache block from the L2 to be written to the L3 cache. If the block hits in the L3, the updates occur and the original store data is passed to the system bus.

If the write-through store misses in the L2 but hits in the L3, the block is flushed from the L3 as a castout if the line had been modified in the L3. If the write-through store misses in the L3, a new line is not allocated in the L3 and only the original store data is passed on to the system bus.

3.7.7.4 L3 Cache Replacement Selection

The L3 cache uses the same two pseudo-random modes of line replacement used by the L2 cache. For the L3 cache L3CR[L3REP] selects either the three-bit counter mode or the pseudo-random number generator mode. The three-bit counter mode (when L3CR[L3REP] = 1) is based on a simple three-bit counter that is incremented on every clock cycle. When a miss occurs, the line in the way pointed to by the counter is chosen for replacement.

The pseudo-random number generator mode (when L3CR[L3REP] = 0) uses the same 16 latches used by the L2 cache described in Section 3.6.4.4, “L2 Cache Line Replacement Algorithms.” These latches are clocked on every clock cycle as shown in Figure 3-20 with 3 XOR functions. However, while the L2 cache uses the value in latches 4, 9, and 15 as the 3-bit value that selects the way for replacement, the L3 cache uses the value in latches 0, 5, and 10 as the 3-bit value for way selection.

3.7.8 L3 Private Memory Operation

The private memory feature allows the MPC7450 to have access to a low latency, high bandwidth private memory space. The private memory space is not snooped and therefore is not coherent with other processors in a system. The private memory space can contain instructions and data and its contents can be cached in the L1 instruction and data caches and the L2 cache, provided that accesses are marked as caching-allowed. Note that instructions in the L3 private memory space should not be marked as caching-inhibited, as caching-inhibited accesses completely bypass the L3 interface. If extended addressing is disabled, the upper four bits of PMBA must be zero in order to be able to match the internal value of A0–A3 (which are zero).

The private memory feature of the MPC7450 is enabled with the L3CR[PMEN] bit and the size is determined by L3CR[PMSIZ]. The L3 private memory logic can be configured such that all of the L3 cache space is used as private memory, or half of the space can be used as L3 cache, and half can be used as private memory. All possible combinations are shown in Table 3-27.

Table 3-27. L3 Cache/Private Memory Configurations

Total SRAM Space	All L3 Cache	Half L3 Cache and Half Private Memory	All Private Memory
1 Mbyte	L3CR L3E = 0b1 L3SIZ = 0b0 (1 Mbyte) PMEN = 0b0 PMSIZ = n/a	n/a	L3CR L3E = 0b0 L3SIZ =n/a PMEN = 0b1 PMSIZ = 0b0 (1 Mbyte)
2 Mbytes	L3CR L3E = 0b1 L3SIZ = 0b1 (2 Mbyte) PMEN = 0b0 PMSIZ = n/a	L3CR L3E = 0b1 L3SIZ = 0b0 (1 Mbyte) PMEN = 0b1 PMSIZ = 0b0 (1 Mbyte) For MPC7457, PMSIZ = 0b00 (1 Mbyte)	L3CR L3E = 0b0 L3SIZ = n/a PMEN = 0b1 PMSIZ = 0b1 (2 Mbyte) For MPC7457, PMSIZ = 0b01 (2 Mbyte)
4 Mbytes (MPC7457-specific)	L3CR L3E = 0b1 L3SIZ = 0b1 (2 Mbyte) PMEN = 0b0 PMSIZ = n/a	L3CR L3E = 0b1 L3SIZ = 0b1 (2 Mbyte) PMEN = 0b1 PMSIZ = 0b01 (2 Mbyte)	L3CR L3E = 0b0 L3SIZ = n/a PMEN = 0b1 PMSIZ = 0b10 (4 Mbyte)

Note that when all of the L3 space is used as private memory, the L3CR[L3E] must be cleared.

The private memory logic receives requests from both the L1 instruction cache and the L1 data cache as well as the L2 cache. The L1 and L2 requests are looked-up in the L3 tags and compared with the proper bits in L3PM[PMBA]. If a match with PMBA is determined, the result of the L3 tag lookup is ignored and the request is forwarded to the external SRAM interface.

The private memory space can be initialized by a sequence of program load instructions from system memory and program store instructions to the private memory space.

The private memory space does not maintain coherency state information. When the L2 cache is reloaded on a miss from private memory for a caching-allowed load or store, the resulting L2 cache state is exclusive, without being broadcast to the system bus.

If the L3 cache is enabled, it must be invalidated or flushed before enabling the L3 private memory. To ensure no livelock scenarios occur in a multiprocessor system, the addresses within the private memory range must be private addresses and not be accessed by any other part of the system.

Note that the L3DO (data-only) and L3IO (instruction-only) L3CR bits have no effect on accesses to private memory. Also, performance monitor events related to the L3 cache may not produce expected results when private memory is enabled. Specifically, hits to the private memory space are treated as L3 cache misses by the performance monitor. There are no new performance monitor events that specifically support the private memory feature.

3.7.8.1 Enabling and Initializing L3 Private Memory

The private memory feature of the MPC7450 is enabled with the L3CR[PMEN] bit and the size is determined by L3CR[PMSIZ]. If configured as one half L3 cache and one half L3 private memory, the half that is L3 cache is enabled or disabled by programming the L3CR[L3E] parameter.

Following a power-on or hard reset, the L3 interface and the L3 clocks are disabled initially. Before enabling the L3 private memory or cache, the L3 clock must first be configured through the L3CR[L3CLK] and L3CR[CLKEN] bits, and a period of time must elapse. Also before enabling the L3 private memory, all other bits in the L3CR must be set appropriately. If configured as one half private memory and one half cache, the L3 cache must be globally invalidated.

The sequence for initializing the L3 cache as private memory is as follows:

1. Set the L3CR[L3CLK] bits to the desired clock divider setting. All other L3 cache configuration bits should be set to properly configure the L3 cache interface for the SRAM type, size, and interface timing required, except do not set L3E, L3I, L3PE, or L3CLKEN.
2. Set L3CR[5] (otherwise reserved bit) to 1.
3. Set L3CR[L3CLKEN] to 1.
4. Wait for the L3 cache clocks to stabilize (100 processor cycles). This can be timed by setting the decrementer for a time period equal to the correct number of L3 cache clocks, or by performing an L3 cache global invalidate.
5. If configured as one half cache and one half private memory, perform an L3 cache global invalidate. The global invalidate could be performed before enabling the L3 clocks, or in parallel with waiting for the L3 clocks to stabilize. Refer to Section 3.7.3.6, “L3 Cache Invalidation,” for more information about L3 cache global invalidation. Note that a global invalidate always takes much longer than it takes for the L3 clocks to stabilize.
6. Clear L3CR[L3CLKEN] to zero.
7. Perform a **sync** instruction and wait 100 processor cycles.
8. Set the base address of the private memory space using L3PM[L3PMADDR]. (This step may also be performed at any time prior to this point.)
9. Set L3CR[PMEN] and configure the private memory size in L3CR[PMSIZ] and set L3CR[L3CLKEN]. If configured as one half cache and one half private memory, also set the L3E and L3SIZ bits of L3CR at this time.
10. Perform a **sync** instruction and wait 100 processor cycles.
11. I If parity is enabled, initialize the SRAM; refer to Section 3.7.8.1.1, “Initializing the L3 Private Memory when Parity is Enabled,” for details.

Note: A **sync** instruction must be performed before writing to L3CR and L3PM; **sync** and **isync** instructions must also be performed after writing to these registers. See Section 2.4.2.4, “Synchronization,” for more details.

3.7.8.1.1 Initializing the L3 Private Memory when Parity is Enabled

In private memory mode, there is no mechanism for the processor to determine if it has already modified data stored in the SRAM. Therefore, if a store to an address in private memory space occurs, the MPC7450 will load the entire cache line from the SRAM and move it into the L1 cache so that is can write the data in question while preserving the rest of the line. Because the SRAM at first contains uninitialized data, including the parity bits, the MPC7450 will take a parity exception if a store occurs and parity checking is enabled. A way to prevent the parity exception is by initializing the SRAM using a series of dcbz instructions to zero out the entire private memory as described in the following steps:

1. Enable private memory mode (L3CR[PMEN] = 0b1). L3 data parity checking (L3CR[L3PE] = 0b1) and L3 address parity checking may be enabled (L3CR[L3APE] = 0b1) at this time.
2. Execute a series of **dcbz** instructions across the entire private memory space. This causes the MPC7450 to allocate a cache line and zero it without initiating a load on the L3 interface.
3. Flush the L1 data cache. This step is not required but is recommended because it will ensure that the last 32K of private memory space is written to the SRAM. For this reason, it is recommended that the L2 cache also be flushed if it is enabled during initialization. Alternatively, the L2 cache can be disabled during private memory initialization and then enabled after it has completed.

3.7.8.2 $\overline{\text{CI}}$ and $\overline{\text{WT}}$ Accesses Not Supported for Private Memory

Cache-inhibited stores that map to the L3 private memory space are not written to the SRAM but they are passed to the system bus. Cache-inhibited loads that map to the L3 private memory space do not access the SRAM. Instead, a system bus transaction is generated and the data is read from the system bus.

Write-through stores (regardless of size) that map to the L3 private memory space are not written to the SRAM but they are passed on to the system bus. Loads from write-through memory (W = 1) that map to the L3 private memory space access the SRAM and the data is returned from the SRAM.

3.7.8.3 Castouts and Private Memory

L1 and L2 castout operations that map to the L3 private memory space are written only to the SRAM and not to the system bus. This is true for all castouts including those generated by **dcbf** and **dcbst** instructions.

3.7.8.4 Snoop Hits and Private Memory

When a snoop hit occurs in the L1 data cache or the L2 cache, and a push (or data intervention) is required, the data is written to private memory if the address is within the private memory range in addition to being written to the system bus. Note that this occurs even for cache flush operations. However, snooping is not supported to areas of private memory if data intervention is disabled (MSSCR0[EIDIS] = 1). Also, snoop pushes and castouts to the private memory space can cause a system livelock as shown in the following sequence for multiple MPC7450s:

1. Processor 1 attempts a write-through (W = 1) write with flush operation.
2. Processor 0 retries processor 1 and generates a snoop push
3. Processor 1 again attempts the write with flush operation
4. Processor 0 again retries processor 1 and generates a snoop push, and so on...

The state of memory in the entire private memory range is assumed to be exclusive modified. Thus an MPC7450 responds to any transaction on the system bus that hits in the private memory range as if the data was resident in one of the on-chip caches as exclusive modified (and no other device should cache data that corresponds to this memory range). Snoop pushes and data intervention transactions occur from the private memory as needed.

Note that in a multiprocessing system, the exclusive modified response in this case may cause a livelock if another master on the bus generates a transaction that claims it has ownership of an address in the private memory range. For example, the following situations can also cause a livelock:

- A multi-MPC7450 system with two overlapping private memory spaces
- Any bus transaction considered exclusive by an alternate master on the bus (not an MPC7450 device)

It is the responsibility of the system software to prevent these scenarios; it is recommended that only the processor using private memory access that private memory address space.

3.7.8.5 Private Memory and Instruction Interactions

All cacheable (I = 0) transactions that read or write data except **eciwx** and **ecowx** are allowed to hit in the private memory space, regardless of the other W, M, and G bit settings of WIMG. The **icbi**, **sync**, **tlbie**, **tlbsync**, **eieio**, **eciwx**, and **ecowx** instructions never hit in the private memory space and are forwarded to the system interface. Any **dcbi** instructions that hit in the private memory space are discarded (after appropriately invalidating the L1 data and L2 caches).

Also, operations caused by **dcbf**, **dcbt**, **dcbst**, **dcbz**, and **dcbi** instructions that map to the L3 private memory space are not broadcast onto the system bus. However, execution of an **icbi** instruction that maps to the L3 private memory space is broadcast on the system bus (even though it has no effect on the L3 private memory).

- Caching-allowed **stwcx.** operations are handled by the L1 data cache and L2 cache similarly to normal caching-allowed stores. The L3 interface does not treat **stwcx.** differently than a normal caching-allowed store. However, caching-inhibited **stwcx.** operations are not supported.
- **dcbz** operations that hit in the private memory space are treated as a 32-byte write-back store operations.
- **dcbf** and **dcbi** operations are issued to the L3 interface after being processed by the L1 data cache and L2 cache. If a cache block push due to a **dcbf** or **dcbi** that hits modified data in the L1 data cache or L2 cache hits in the private memory space, the cache block is written to the L3 SRAMs.
- **dcbst** instructions are issued to the L3 interface after being processed by the L1 data cache and L2 cache. If a cache block push due to a **dcbst** that hits modified data in the L1 data cache or L2 cache hits in the private memory space, the cache block is written to the L3 SRAMs.

3.7.9 L3 Cache SRAM Timing Examples

This section describes the signal timing for the following three types of SRAM supported by the MPC7450 L3 cache interface:

- MSUG2 dual data rate SRAMs that provide data synchronous to the L3_ECHO_CLK input signals to the MPC7450 and on each clock edge
- Late-write SRAMs which are required by the MPC7450 to be of the pipelined (register-register) configurations
- Pipeline burst SRAMs, referred to as PB2-type SRAMs

The timing diagrams illustrate the best case logical interface operations and are not AC timing accurate. For proper interface operation, the designer must select SRAMs that support the signal sequencing illustrated in the timing diagrams, particularly in regards to those cycles when the data bus may be driven, is required to be driven, and must not be driven by the SRAM.

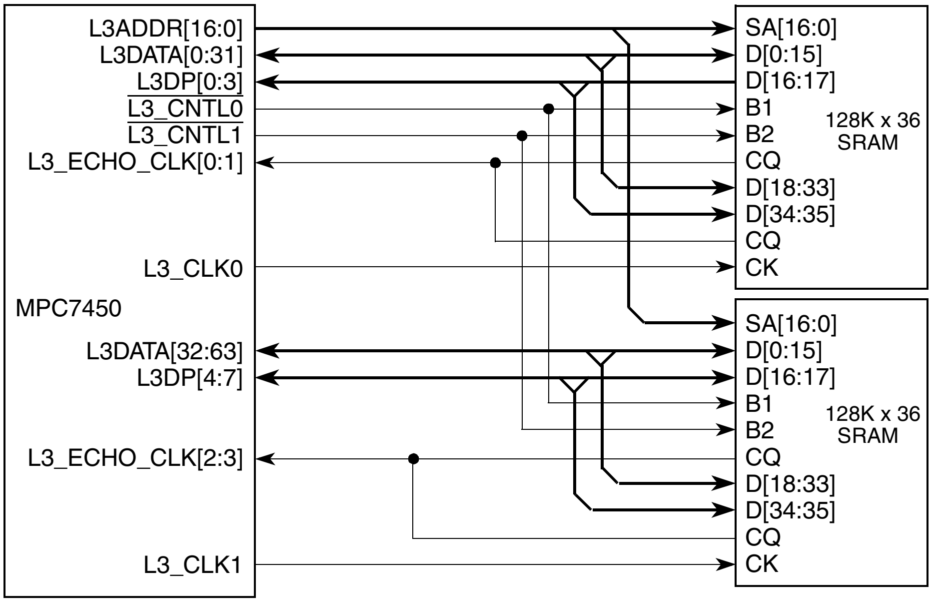
The SRAM selected for a system design is usually a function of desired system performance, L3 cache bus frequency, and SRAM unit cost. The following sections describe the operation of the three SRAM types supported by the MPC7450, and some of the design trade-offs associated with each.

3.7.9.1 MSUG2 DDR Interface Timing

MSUG2 DDR SRAMs are a new type of high performance RAM. The following three major differences exist between this RAM and other synchronous RAMs:

- Data is returned by the target SRAM asynchronously to the input clock on the SRAM.
- An additional clock is provided as an output by the SRAM that is synchronous with its returning data (echo_clock input to the L3_ECHO_CLK[0:3] signals of the MPC7450).
- Data is returned on each edge of the returned data clock.

The MPC7450 does not use the continue-burst feature of this SRAM and instead supplies two addresses for each cache line transfer. Double transfers are always selected, forcing data to transfer on each edge of the clock.Figure 3-23 shows the MPC7450 configured with a 1-Mbyte L3 cache using MSUG2 DDR.



Notes:

For a 2-Mbyte L3 cache, use address bits 17–0 (bit 0 is LSB. For the MPC7457, the L3 cache uses address bits 18–0 (bit 0 is LSB).

The routing for the point-to-point signals (L3_CLK[0:1], L3DATA[0:63], L3_DP[0:7] and L3_ECHO_CLK[0:3] to a particular SRAM device must be delay matched.

No pull-up resistors are normally required for the L3 cache interface.

The MPC7450 supports only one bank of SRAMs.

For high-speed operation, no more than two loads should be presented on each L3 address and control signal. All other L3 signals should have no more than one load.

Figure 3-23. Typical 1-Mbyte L3 Cache Using MSUG2 DDR

Figure 3-24 shows an example timing diagram of the MPC7450 L3 interface with an MSUG2 DDR SRAM shown in Figure 3-21. This type of device uses a skew-based source synchronous design instead of a delay-based synchronous model. This allows the interface to run at much higher data rates. Although in reality there are multiple clocks involved that operate asynchronously with each other, the timing in Figure 3-24 shows echo_clk (the SRAMs returned data clock) as synchronous with the processor clock signals (L3_CLK[0:1]).

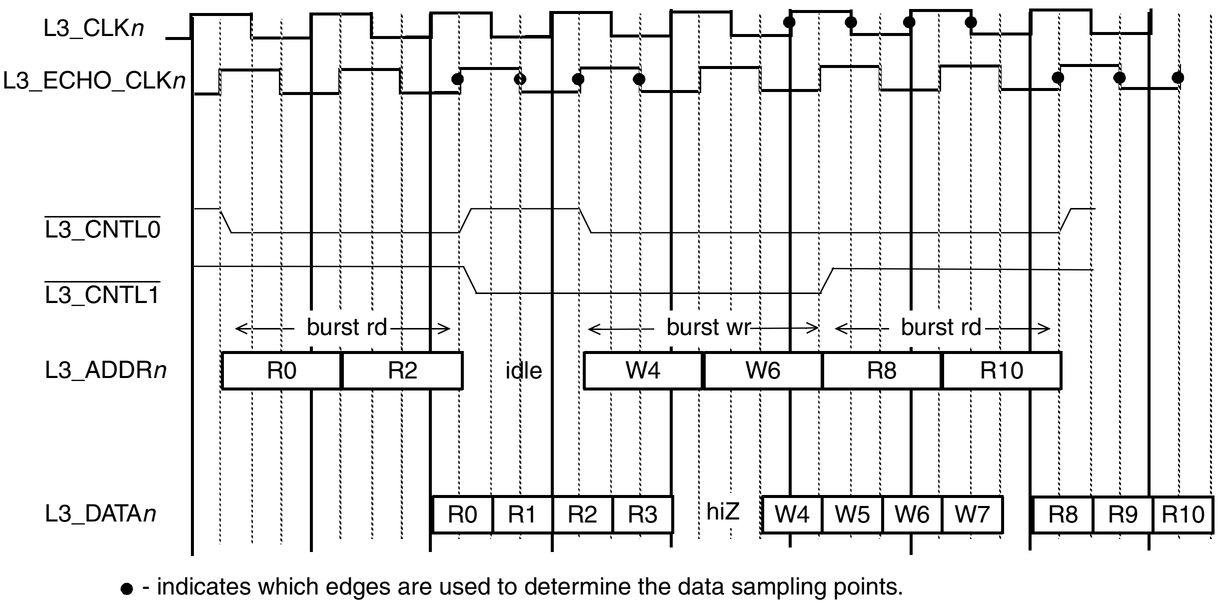


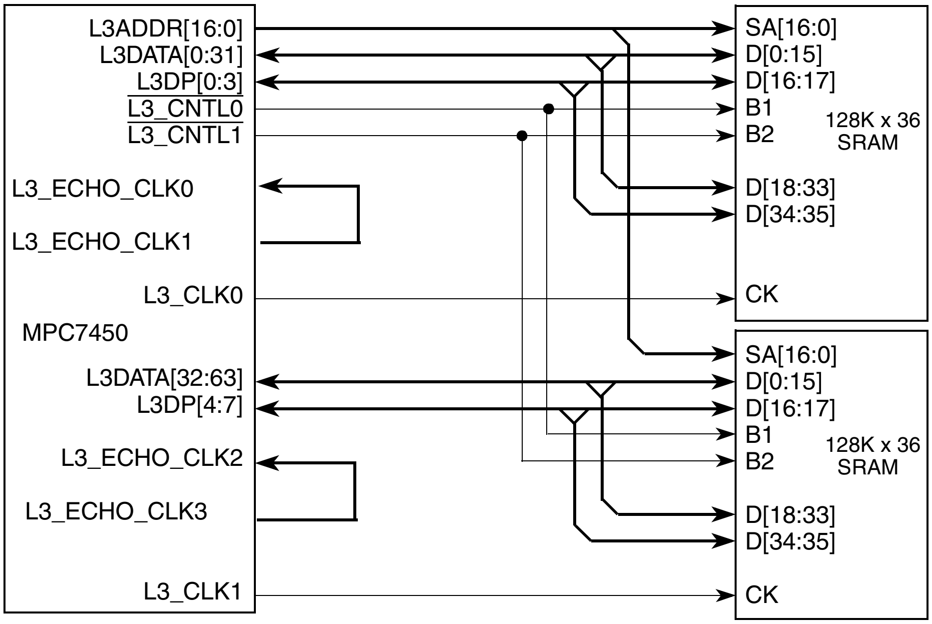
Figure 3-24. MSUG2 DDR Memory Access Example

3.7.9.2 Late-Write SRAM Timing

Late-write SRAMs offer improved performance when compared to pipelined burst SRAMs by not requiring an extra read cycle during read operations, and requiring one cycle less when transitioning from a read to a write operation. Late-write SRAMs implement an internal write queue, allowing write data to be provided one cycle after the write operation is signaled on the address and control buses. In this manner, write operations are queued on the address and data bus in the same manner as read operations, allowing transitions between read and write operations to occur more efficiently.

Note that during burst transfers into and out of the SRAM array, the MPC7450 generates an address for each data beat. That is, the MPC7450 does not use the burst feature (one address, many data beats) of the late-write SRAMs.

Figure 3-25 shows the signal connections between an MPC7450 and either late-write or PB2 SRAMS.



Notes:

For a 2-Mbyte L3 cache on the MPC7450, the L3 cache uses address bits 17–0 with bit 0 being the LSB. For the MPC7457, the L3 cache uses address bits 18-0 with bit 0 being the LSB.

The routing for the point-to-point signals (L3_CLK[0:1], L3DATA[0:63], L3_DP[0:7] and L3_ECHO_CLK[0:3] to a particular SRAM device must be delay matched.

No pull-up resistors are normally required for the L3 cache interface.

The MPC7450 supports only one bank of SRAMs.

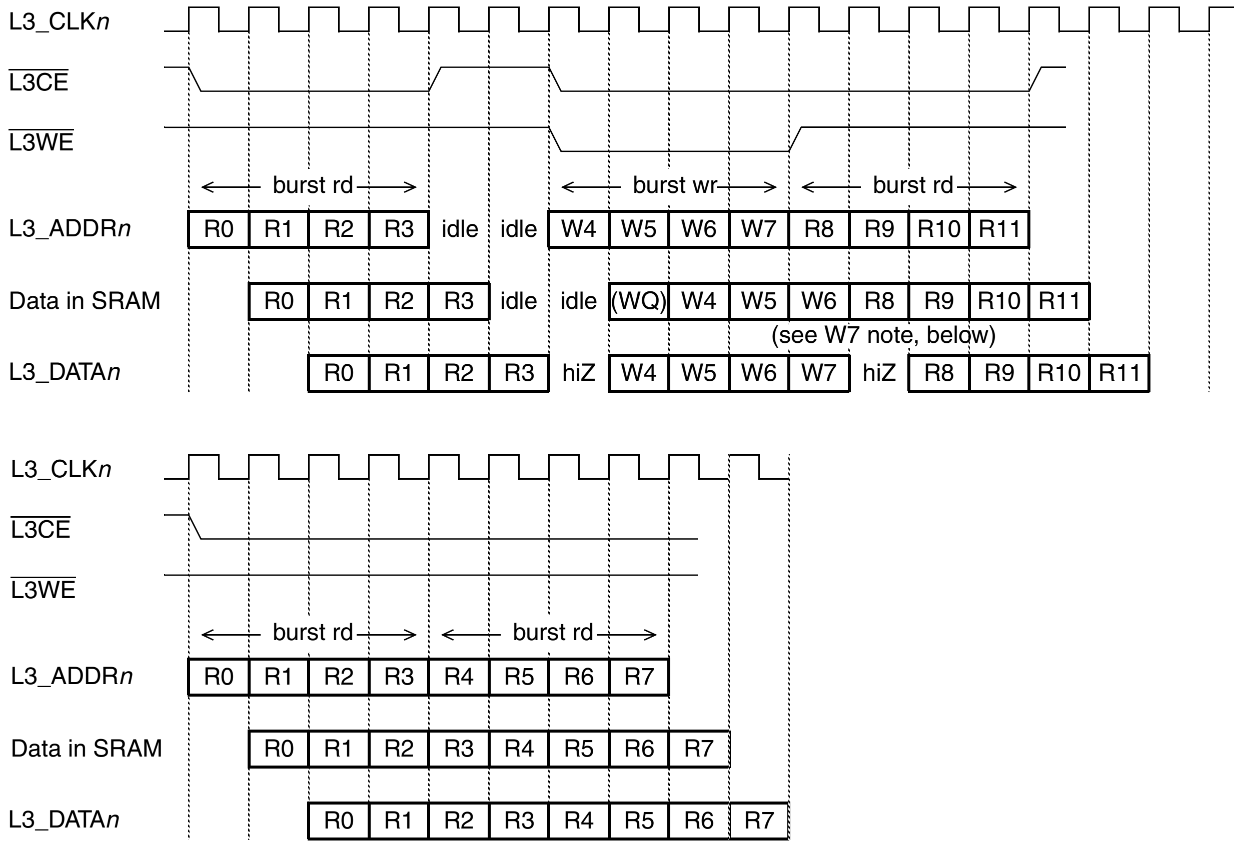
For high-speed operation, no more than two loads should be presented on each L3 address and control signal. All other L3 signals should have no more than one load.

Figure 3-25. L3 Cache Configuration for Late-Write or PB2 SRAMs

Table 3-28. Signal Function Changes for Late-Write and PB2 SRAMs

Signal Name	# of Pins	Changed Function for Late-Write and PB2			
		Active	I/O	Meaning	Comments
$\overline{\text{L3\_CNTL1}}$	1	Low	Output	Write operation ($\overline{\text{L3WE}}$)	Synchronous
$\overline{\text{L3\_CNTL0}}$	1	Low	Output	Chip enable ($\overline{\text{L3CE}}$)	Synchronous
L3_ECHO_CLK[0,2]	2	High	Input	Clock input to MPC7450 for read data synchronization.	Provides compatibility with the DDR SRAM interface
L3_ECHO_CLK[1,3]	2	High	Output	Clock output from MPC7450 to be wrapped back to clock input.	To be routed back to the clock inputs for compatibility with the DDR SRAM interface.

Figure 3-26 shows memory access timings when the L3 cache interface is configured for late-write SRAM.



Note: WQ is the last previous write that was queued in the late-write RAM.
W7 Note: W7 is queued in the late-write device and won’t appear in SRAM Memory until the next write.

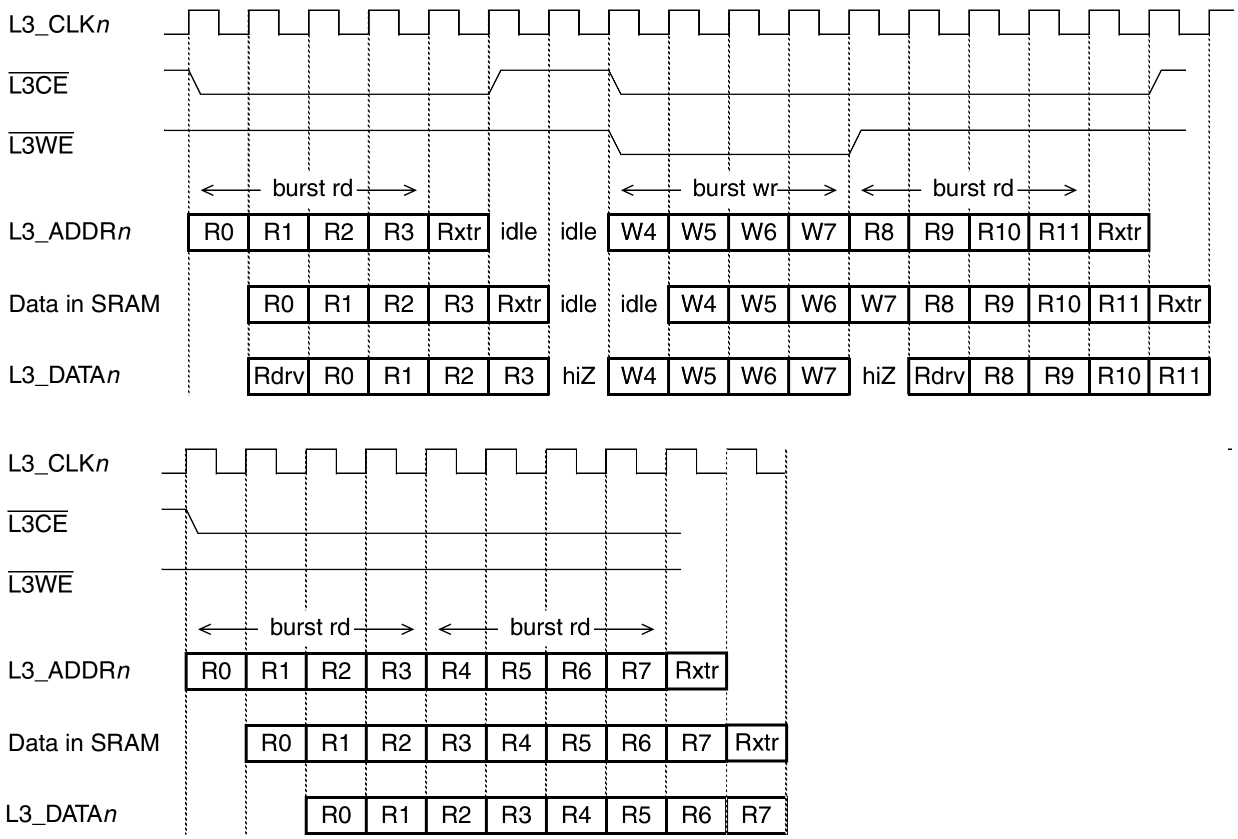
Figure 3-26. Late-Write SRAM Timing

3.7.9.3 Pipelined Burst SRAM

Pipelined burst SRAMs are sometimes referred to as PB2 (pipelined burst, 2nd generation) SRAMs. Pipelined burst SRAMs operate by clocking read data from the memory array into a buffer before driving the data onto the data bus. This causes an extra clock cycle of latency for initial read accesses, but the L3 cache bus frequencies supported can be higher. Note that the MPC7450’s L3 cache interface requires the use of single-cycle deselect pipelined burst SRAM for proper operation.

Note that during burst transfers into and out of the SRAM array, the MPC7450 generates an address for each data beat. That is, the MPC7450 does not use the burst feature (one address, many data beats) of the pipelined burst SRAMs.

Figure 3-27 shows memory access timings when the L3 cache interface is configured for pipelined burst SRAM.



Notes: Rdrv indicates where some burst RAMs may begin driving the data bus.
Rxtr indicates where an extra read cycle is signaled to keep the burst RAM driving the data bus for the last read. The MPC7450 does not support aborted reads

Figure 3-27. Pipeline Burst SRAM Timing

3.8 System Bus Interface

The system bus interface buffers bus requests from the L1 instruction cache, the L1 data cache, the L2 cache, and the L3 cache, and executes the requests per the system bus protocol. It includes address register queues, prioritizing logic, and bus control logic. The bus interface unit includes a sixteen-entry (default value is eight-entry) data transaction queue to support pipelining of multiple transactions. The bus interface also captures snoop addresses for snooping in the caches, the address register queues, and the reservation address. For additional information about the MPC7450 bus interface and the bus protocols, refer to Chapter 9, “System Interface Operation.”

3.8.1 MPC7450 Caches and System Bus Transactions

The MPC7450 transfers data to and from the caches on the system bus in single-beat transactions of up to eight bytes, in two-beat burst transfers of 16 bytes for caching-inhibited (WIMG = x1xx) or caching-allowed, write-through (WIMG = 10xx) AltiVec loads and stores (in MPX bus mode), or in four-beat transactions of 32 bytes for cache block fills. The MPC7450 transfer burst ($\overline{\text{TBST}}$) output signal indicates to the system whether the current transaction is a single-beat transaction or burst (two- or four-beat) transfer.

Single-beat bus transactions can transfer from one to eight bytes to or from the MPC7450, and can be misaligned. Single-beat transactions can be caused by caching-allowed, write-through accesses (WIMG = 10xx), caching-inhibited accesses (WIMG = x1xx), accesses when the data cache is disabled (HID0[DCE] is cleared), or accesses when the data cache is locked (HID0[DLOCK] is set).

In MPX bus mode, two-beat burst transactions are caused by quad-word (128-bit) AltiVec loads and stores that are marked write-through or caching-inhibited. These two-beat burst transactions are always aligned to a quad-word boundary. In 60x bus mode, quad-word AltiVec loads and stores cause an alignment exception if write-through or caching-inhibited.

Instruction fetches are always treated as quad-word (16-byte) entities internally. For cacheable instruction fetches, the system bus always requests a full L1 cache line (32 bytes). For noncacheable fetches in MPX bus mode, a cache-inhibited quad word (two-beat burst) request occurs. Because the 60x bus mode does not support quad-word accesses a cache-inhibited access in 60x mode is converted to a cache-line (32-byte, four-beat burst) access on the bus. When this occurs, the portion of the cache line that was not internally requested is discarded.

Cache block burst transactions on the MPC7450 always transfer 32-bytes of data in four beats of 8-bytes each, and are aligned to a double- or quad-word boundary as they are requested. Burst transactions have an assumed address order. For caching-allowed read operations, instruction fetches, or caching-allowed, write-back write operations that miss in the cache, the MPC7450 presents the double- or quad-word-aligned address associated with the load/store instruction or instruction fetch that initiated the transaction.

As shown in Figure 3-28, the first double word contains the address of the load/store or instruction fetch that missed the cache. This minimizes latency by allowing the critical code or data to be forwarded to the requesting execution unit before the rest of the block is filled. For all other burst operations, however, the entire block is transferred in order (cache-block aligned). Similar to the principles described for double-word fetches, quad-word fetches (for vector load operations and instruction fetches) are also forwarded to the requesting unit as they are requested and in critical quad-word order.

MPC7450 Cache Address
Bits (27– 28)

00	01	10	11
A	B	C	D

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

Beat

0	1	2	3
A	B	C	D

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

Beat

0	1	2	3
C	D	A	B

Figure 3-28. Double-Word Address Ordering—Critical Double Word First

3.8.2 Bus Operations Caused by Cache Control Instructions

The cache control, TLB management, and synchronization instructions supported by the MPC7450 may affect or be affected by the operation of the system bus. The operation of the instructions may also indirectly cause bus transactions to be performed, or their completion may be linked to the bus.

When memory coherency is required (WIMG = xx1x) and HID1[ABE] = 1, the **dcbst**, **dcbf**, **dcbi**, and **icbi** instructions are broadcast on the system bus (for both MPX bus and 60x bus mode) as described in Table 3-29. A **dcbi** or **dcbf** can create an address-only flush and a **dcbst** can create an address-only clean. When M = 0, **dcbst**, **dcbf**, and **dcbi** instructions are only broadcast on the bus when the cache state hits as modified in either the L1, L2, or L3 cache. Note that **dcbst**, **dcbf**, and **dcbi** instructions would create castout operations to the bus if they hit modified within the caches even when M = 0. The **icbi** instruction is never broadcast when M = 0. A **dcbz** or **dcba** is never broadcast if the internal cache state is modified. 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*.

Table 3-29. Bus Operations Caused by Cache Control Instructions (WIM = xx1)

Instruction	Current State		Next Cache State	Bus Operation	Comment
	Cache Coherency	HID1 Setting			
sync	Do not care	SYNCBE = 0	No change	None	Waits for memory queues to complete bus activity
		SYNCBE = 1		SYNC	
tlbie	Do not care	ABE = 0	No change	None	—
		ABE = 1		TLBIE	Address-only bus operation

Table 3-29. Bus Operations Caused by Cache Control Instructions (WIM = xx1) (continued)

Instruction	Current State		Next Cache State	Bus Operation	Comment
	Cache Coherency	HID1 Setting			
tlbsync	Do not care	ABE = 0	No change	None	—
		ABE = 1		TLBSYNC	Address-only bus operation
eieio	Do not care	SYNCBE = 0	No change	None	—
		SYNCBE = 1		EIEIO	Address-only bus operation
dcbt	M, E, S		No change	None	—
dcbt	I		E, S	Read	Fetched cache block is stored in the cache
dcbtst	M, E, S		No change	None	—
dcbtst	I		E, S	Read (60x mode) RCLAIM (MPX mode)	Fetched cache block is stored in the cache
dcbz	M, E		M	None	Writes over modified data
dcbz	S, I		M	Kill	—
dcbst	M		E, S, I	Write with kill	—
dcbst	E, S, I	ABE = 0	No change or I	None	—
		ABE = 1		Clean	—
dcbf, dcbi	M	—	I	Write with kill	Block is pushed
dcbf, dcbi	E, S, I	ABE = 0	I	None	—
		ABE = 1		Flush	Address-only bus operation
dcba	M, E		M	None	Writes over modified data
dcba	S, I		M	Kill	—
icbi	V, I	ABE = 0	I	None	Instruction cache only
		ABE = 1		ICBI	Instruction cache only

When memory coherency is not required (WIMG = xx0x), the **dcbz**, **dcba, dcbf**, **dcbi**, and **dcbtst** instructions are broadcast on the system bus (for both MPX bus and 60x bus mode) as described in Table 3-30.

Table 3-30. Bus Operations Caused by Cache Control Instructions (WIM = xx0)

Instruction	Current State		Next Cache State	Bus Operation	Comment
	Cache Coherency	HID1 Setting			
dcbt	M, E, S	—	No change	None	—
dcbt	I	—	E, S	Read	Fetched cache block is stored in the cache

Table 3-30. Bus Operations Caused by Cache Control Instructions (WIM = xx0) (continued)

Instruction	Current State		Next Cache State	Bus Operation	Comment
	Cache Coherency	HID1 Setting			
dcbtst	M, E, S	—	No change	None	—
dcbtst	I	—	E, S	Read (60x mode) RCLAIM (MPX mode)	Fetched cache block is stored in the cache
dcbz	M, E, S, I	—	M	None	
dcba	M, E, S, I	—	M	None	
dcbf, dcbi	M	—	I	Write with kill	Block is pushed
dcbf, dcbi	E, S, I	—	I	None	—
dcbst	M	—	E, S, I	Write with kill	Block is pushed
dcbst	E, S, I	—	I	None	—
icbi	V, I	—	I	None	Instruction cache only

For additional details about the specific bus operations performed by the MPC7450, see Chapter 9, “System Interface Operation.”

3.8.3 Transfer Attributes

In addition to the address and transfer type signals, the MPC7450 supports the transfer attribute signals $\overline{\text{TBST}}$, TSIZ[0:2], $\overline{\text{WT}}$, $\overline{\text{CI}}$, and $\overline{\text{GBL}}$. The $\overline{\text{TBST}}$ and TSIZ[0:2] signals indicate the data transfer size for the bus transaction.

The $\overline{\text{WT}}$ signal reflects the write-through/write-back status (the complement of the W bit) for the transaction as determined by the MMU address translation during write operations. $\overline{\text{WT}}$ is also asserted when the data cache is locked (with HID0[DLOCK] or LDSTCR[DCWL] = 0xFF) and for burst writes due to **dcbf** (flush) and **dcbst** (clean) instructions, snoop pushes, and **eciwx** transactions; $\overline{\text{WT}}$ is negated for **ecowx** transactions.

For read transactions, the $\overline{\text{WT}}$ signal reflects whether the access is an instruction or data access as follows:

- $\overline{\text{WT}}$ is asserted for data reads
- $\overline{\text{WT}}$ is negated for instruction reads

The $\overline{\text{CI}}$ signal reflects the caching-inhibited/caching-allowed status (the complement of the I bit) of the transaction as determined by the MMU address translation. The $\overline{\text{CI}}$ signal is asserted for data loads or stores if the L1 data cache is disabled. The $\overline{\text{CI}}$ signal is also always asserted for **eciwx**/**ecowx** bus transactions independent of the address translation.

The $\overline{\text{GBL}}$ signal reflects the memory coherency requirements (the complement of the M bit) of the transaction as determined by the MMU address translation. 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). Because cache block castouts and snoop pushes do not require snooping, the $\overline{\text{GBL}}$ signal is not asserted for these operations. Note that $\overline{\text{GBL}}$ is asserted for all data read or write operations when using real addressing mode (that is, address translation is disabled).

Table 3-31 summarizes the address and transfer attribute information presented on the bus by the MPC7450 for various master or snoop-related transactions. Note that the address ranges shown in the table apply when 36-bit physical addressing is used (HID0[XAEN] = 1).

Table 3-31. Address/Transfer Attributes Generated by the MPC7450

Bus Transaction	Addr [0:35]	TT[0:4]	$\overline{\text{TBST}}$	TSIZ[0:2]	$\overline{\text{WT}}$	$\overline{\text{CI}}$	$\overline{\text{GBL}}$
Instruction fetch operations							
Burst	PA[0:32] || 0b000	0 1 0 1 0	0	0 1 0	1	1	¬ M
Data cache operations							
Cache block fill (due to load miss)	PA[0:32] || 0b000	A 1 0 1 0	0	0 1 0	0	1	¬ M
Cache block fill (due to store miss)	PA[0:32] || 0b000	A 1 1 1 0	0	0 1 0	1	1	¬ M
Store merged to 32 bytes	PA[0:30] || 0b00000	0 1 1 0 0	0	0 1 0	1	1	¬ M
Castout (normal replacement)	CA[0:30] || 0b00000	0 0 1 1 0	0	0 1 0	1	1	1
Cache block clean due to **dcbst** hit to modified	PA[0:30] || 0b00000	0 0 1 1 0	0	0 1 0	0	1	1
Cache block flush due to **dcbf** hit to modified	PA[0:30] || 0b00000	0 0 1 1 0	0	0 1 0	0	1	1
Snoop copyback	CA[0:30] || 0b00000	0 0 1 1 0	0	0 1 0	0	1	1
dcbt, **dst**, **dstt**	PA[0:30] || 0b00000	0 1 0 1 0	0	0 1 0	0	1	¬ M
dcbtst, **dstst**, **dststt** (60x bus mode)	PA[0:30] || 0b00000	0 1 0 1 0	0	0 1 0	0	1	¬ M
dcbtst, **dstst**, **dststt** (MPX bus mode)	PA[0:30] || 0b00000	0 1 1 1 1	0	0 1 0	0	1	¬ M
Data cache bypass operations							
Single-beat read (caching-inhibited or cache disabled)	PA[0:35]	0 1 0 1 0	1	S S S	0	¬ I	¬ M
AltiVec load (caching-inhibited, write-through, or cache disabled) in MPX bus mode	PA[0:32] || 0b000	0 1 0 1 0	0	0 0 1	0	¬ I	¬ M
Single-beat write (caching-inhibited, write-through, cache disabled, or cache completely locked)	PA[0:35]	0 0 0 1 0	1	S S S	¬ W	¬ I	¬ M
Special instructions							
icbi (addr-only)	PA[0:30] || 0b00000	0 1 1 0 1	0	0 1 0	¬ W	¬ I	¬ M
dcba (addr-only)	PA[0:30] || 0b00000	0 1 1 0 0	0	0 1 0	1	1	0
dcbz (addr-only)	PA[0:30] || 0b00000	0 1 1 0 0	0	0 1 0	1	1	0

Table 3-31. Address/Transfer Attributes Generated by the MPC7450 (continued)

Bus Transaction	Addr [0:35]	TT[0:4]	$\overline{\text{TBST}}$	TSIZ[0:2]	$\overline{\text{WT}}$	$\overline{\text{CI}}$	$\overline{\text{GBL}}$
dcbf, dcbi (addr-only)	PA[0:30] || 0b00000	0 0 1 0 0	0	0 1 0	¬ W	¬ I	¬ M
dcbst (addr-only)	PA[0:30] || 0b00000	0 0 0 0 0	0	0 1 0	¬ W	¬ I	¬ M
sync (addr-only)	0x0_0000_0000	0 1 0 0 0	0	0 1 0	1	1	0
tlbsync (addr-only)	0x0_0000_0000	0 1 0 0 1	0	0 1 0	1	1	0
tlbie (addr-only)	0b0000 || EA[0:31]	1 1 0 0 0	0	0 1 0	1	1	0
eieio (addr-only)	0x0_0000_0000	1 0 0 0 0	0	0 1 0	1	1	0
eciwx	PA[0:33] || 0b00	1 1 1 0 0	EAR[28:31]		0	0	1
ecowx	PA[0:33] || 0b00	1 0 1 0 0	EAR[28:31]		1	0	1

Notes: PA = Physical address, CA = Cache address, EA = Effective address.

W,I,M = WIM state from address translation; ¬ = complement; 0 or 1 = WIM state implied by transaction type in table.

A = Atomic; high if **stwcx.** or **lwarx**, low otherwise

S = Transfer size

Special instructions listed may not generate bus transactions depending on cache state.

TT[0:4] = 0b01011 (RWNITC) is snooped by the MPC7450, but is not generated by the MPC7450.

TT[0:4] = 0b00001 (**lwarx** reservation set) is neither snooped nor generated by the MPC7450.

3.8.4 Snooping of External Transactions

The MPC7450 maintains data cache coherency in hardware by coordinating activity between the data cache, the L2 cache, the L3 cache, the memory subsystem, and the bus. The MPC7450 has a write-back caching capability that relies on bus snooping to maintain cache coherency with other caches in the system. For the MPC7450, the coherency size of the bus is 32 bytes, the size of a cache block. This means that any bus transactions that cross an aligned 32-byte boundary must present a new address onto the bus at that boundary for proper snoop operation by the MPC7450, or they must operate noncoherently with respect to the MPC7450.

As bus operations are performed on the bus by other bus masters, the MPC7450 bus snooping logic monitors the addresses and transfer attributes that are referenced. The MPC7450 must see all system coherency snoops to function properly in a symmetric multiprocessing (SMP) environment. The MPC7450 cannot support external devices that filter out snoop traffic on the bus (for example, an external, in-line cache).

The MPC7450 snoops bus transactions during the cycle that $\overline{\text{TS}}$ is asserted for all global transactions ($\overline{\text{GBL}}$ asserted).

Every assertion of $\overline{\text{TS}}$ detected by the MPC7450 (whether snooped or not) must be followed by an accompanying assertion of $\overline{\text{AACK}}$.

3.8.4.1 Types of Transactions Snooped by MPC7450

There are several bus transaction types defined for the system bus. As shown in Table 3-32, the MPC7450 snoops many, but not all, system transactions. The transactions in Table 3-32 correspond to the transfer type signals TT[0:4], which are described in Section 8.3.4.2, “Transfer Type (TT[0:4]).”

Table 3-32. Snooped Bus Transaction Summary

Transaction	TT[0:4]	Snooped by MPC7450
Clean	00000	Yes
Flush	00100	Yes
sync	01000	Yes
Kill	01100	Yes
eieio	10000	No
External control word write	10100	No
TLB invalidate	11000	Yes
External control word read	11100	No
lwarx reservation set	00001	No
Reserved	00101	No
tlbsync	01001	Yes
icbi	01101	Yes
Reserved	1XX01	No
Write-with-flush	00010	Yes
Write-with-kill	00110	Yes
Read	01010	Yes
Read-with-intent-to-modify (RWITM)	01110	Yes
Write-with-flush-atomic	10010	Yes
Reserved	10110	No
Read-atomic	11010	Yes
Read-with-intent-to-modify-atomic	11110	Yes
Reserved	00011	No
Reserved	00111	No
Read-with-no-intent-to-cache (RWNITC)	01011	Yes
Read-claim (RCLAIM) (MPX bus mode only)	01111	Yes
Reserved	1XX11	No

Once a qualified snoop condition is detected on the bus, the snooped address associated with $\overline{\text{TS}}$ is compared against the data cache tags, the LSU and memory subsystem queues, reservation address, and/or other storage elements as appropriate. The L1 data cache tags, L2 cache tags, and L3 cache tags are snooped for standard data cache coherency support. No snooping is done in the instruction cache for coherency (except that the **icbi** instruction can cause matching entries to be invalidated).

The LSU and memory subsystem queues are snooped for pipeline collisions and memory coherency collisions. A pipeline collision is detected when another bus master addresses any portion of a line that this MPC7450 is currently processing in its caches. A memory coherency collision occurs when another bus master addresses any portion of a line that the MPC7450 has currently queued to write to memory from the data cache (castout or push), but has not yet been granted bus access to perform.

If the snooped address does not hit in the cache, snooping finishes with no action taken. If, however, the address hits in the cache, the MPC7450 reacts according to the coherency protocol diagrams shown in Section 3.3.2.5, “MESI State Transitions.”

3.8.4.2 L1 Cache State Transitions and Bus Operations Due to Snoops

Table 3-35 shows the state transitions in the L1 caches for each snoop type. For each snoop, the L1 responds with valid if the line was shared, exclusive, or modified, and modified if the line was modified. The snoop types in Table 3-35 are listed in Table 3-33.

Table 3-33. Definitions of Snoop Type for L1 Cache/Snoop Summary

Snoop Type	Definition
Snoop Kill Reservation	If the snoop address matches a valid reservation in the core, kill the reservation after the response window if there is no retry. This operation is caused by RWITM, RWITM ATOMIC, RCLAIM, KILL external snoops.
Snoop Flush-Kill	Push any modified data to the L1 push buffer and invalidate the line. Return the initial MESI state of the line. This operation is caused by stores with I = 0 and W = 0, **stwcx** with W = 0 and I = 0, **dcbtst**, **dstst**, **dcbz**, or **dcba** instructions on-chip.
Snoop Flush	Push any modified data to the L1 push buffer and invalidate the line. Return the initial MESI state of the line. This operation is caused by dcbf instruction on-chip, RWITM, RWITM ATOMIC, RCLAIM, KILL, WRITE W/FLUSH, WRITE W/FLUSH ATOMIC, or FLUSH external snoop
Snoop Read	Push any modified data to the L1 push buffer. If the line was valid, leave it shared. Return the initial MESI state of the line. This operation is caused by **dcbst**, load, **lwarx** or touch instructions on-chip or CLEAN, RWNITC, READ, READ ATOMIC external snoops.
Snoop **icbi**	Invalidate the line in the instruction cache. This operation is caused by an **icbi** instruction on-chip or ICBI external snoop.
Snoop **tlbie**	Invalidate all matching PTEs in the Instruction and Data TLBs. Mark all outstanding memory accesses that used old translations. This operation is caused by an TLBIE external snoop.

Table 3-34 defines some terms used in Table 3-35.

Table 3-34. Definitions of Other Terms for L1 Cache/Snoop Summary

Term	Definition
Snoop Type	The local snoop type. See Section 3.6.4.5, “L2 and L3 Operations Caused by L1 Requests,” and Section 3.8.4.3, “L2 and L3 Operations Caused by External Snoops,” for a list of the operations and states that cause the various L1 snoop types
Initial L1 State	The MESI state of the cache before the snoop begins.
Final L1 State	If the L1 MESI state is unchanged, then the entry is marked as same; otherwise, the MESI state at the end of the snoop operations

Table 3-35. L1 Cache State Transitions Due to Snoops

Snoop Type	Initial L1 State	Final L1 State	Action	Comments
Kill Reservation	n/a	same	Kill Reservation after response window if the address matches and no core retried the operation.	—
Flush-Kill	I	same	If L1 = M, data is moved to push buffer. MSS will request it. Kill Reservation after response window if the address matches and no core retried the operation.	Cache line is invalidated.
	S/E/M	I		
Flush	I	same	If L1=M, data is moved to push buffer. MSS will request it.	Cache line is invalidated.
	S/E/M	I		
Read	I/S	same	If L1 = M, data is moved to push buffer. MSS will request it.	If cache line was valid, leave it shared. Note: if the reservation address matched but the cache line was invalid, the L1 Snoop Logic will synthesize a shared response to the LMQ or the bus.
	E/M	S		
icbi	n/a	n/a	Invalidate matching line in the instruction cache.	—
tlbie	n/a	n/a	Invalidate matching TLB entries in the ITLB and DTLB.	—

3.8.4.3 L2 and L3 Operations Caused by External Snoops

The L1, L2, and L3 cache states affect the response to external snoops. Some snoop types do not affect the caches. Table 3-37 shows the response to all snooped bus operations, depending on the initial cache state. See Section 3.5.9, “L1 Cache Operation Summary,” for a description of the L1 snoop responses.

Table 3-37 shows the state transitions in the L1, L2, and L3 caches for each external snoop operation. The table lists only legal state combinations. The columns are defined in Table 3-36. Note that the L3 cache is not supported by the MPC7441, MPC7445, MPC7447, MPC7447A, or MPC7448.

Table 3-36. Definitions for L2/L3 Cache/Snoop Summary

Term	Definition
Snoop Type	The bus transfer type as described in Table 3-35.
L1 Snoop Type	The L1 snoop operation (if any) triggered by this operation.
L1 Response	The invalid/valid/modified response of the core. If the L1 response is retry, the external bus response is always retry.
Initial L2 State	The MESI state of this address in the L2 cache before the snoop operation.
Initial L3 State	The MESI state of this address in the L3 cache before the snoop operation.
Final L2 State:	The MESI state of this address in the L2 cache when all operations triggered by this snoop are complete.
Final L3 State	The MESI state of this address in the L2 cache when all operations triggered by this snoop are complete.
Bus Response	Shared indicates there is a valid copy of the data and the data stays valid (if the bus operation supports shared response). Modified indicates there is a modified copy of the data and the cache will provide intervention data. Retry indicates the master must try the operation again to get the most up-to-date data and a clean response. Shared and retry together indicates the this device must perform a push. Shared and modified together indicates this device will provide intervention data and retain a valid copy of the line.

Note the following:

- Snoop kill reservation is performed only if a matching reservation exists in the L1 (part of the L1 snoop response).
- For write-with-kill and kill external snoops, a flush operation is sent to the L1’s. If the L1 is modified, a push is generated in the SMC. The external snoop logic and SMC contain circuitry to drop this push if the snoop is not retried on the bus. The MPC7450 does not respond retry if there is modified data in the L1, L2 or L3.
- The atomic bus operations have the same snoop responses as the non-atomic ones.
- Because the MPC7450 only snoops global accesses ($\overline{\text{GBL}}$ asserted), that is assumed for all of the tables. The MPC7450 will not issue a snoop response ($\overline{\text{ARTRY}}$ and $\overline{\text{HIT}}$) for transactions in which $\overline{\text{GBL}}$ is not asserted.

Table 3-37. External Snoop Responses and L1, L2, and L3 Actions

Snoop Type	L1 Snoop Type	L1 Resp	Initial L2 State	Initial L3 State	Final L2 State	Final L3 State	Response to Snoop	Comments
Flush	Flush	I/V	I//S/E	I/S/E	I	I	None	Invalidate L1/L2/L3.
				M	I	I	GBL	Intervene from L3 and invalidate L1/L2/L3.
			M	I/S/E	I	I	GBL	Intervene from L2 and invalidate L1/L2/L3.
				M	I	I	GBL	
		M	I/S/E	I/S/E	I	I	GBL	Intervene from L1 and invalidate L1/L2/L3.
				M	I	I	GBL	
			M	I/S/E	I	I	GBL	
				M	I	I	GBL	
Write W/flush	Flush[1]	I/V	I/S/E	I/S/E	I	I	None	Invalidate L1.
				M	I	I	Retry	Cache paradox: Push block from L3. Invalidate L1/L2/L3.
			M	I/S/E/M	I	I	Retry	Cache paradox: Push block from L2. Invalidate L1/L2/L3.
		M	I/S/E/M	I/S/E/M	I	I	Retry	Cache paradox: Push block from L1. Invalidate L1.
Kill	Flush Kill Reservation	I/V/M	I/S/E	I/S/E	I	I	None	Invalidate L1/L2/L3. Kill reservation (if necessary.) after ARTRY window. If L1 response is M, the push in SMC is killed if bus response is not ARTRY.
				M	I	I	None	
			M	I/S/E	I	I	None	
				M	I	I	None	
Write W/kill	Flush	I/V/M	I/S/E	I/S/E	I	I	None	Invalidate L1/L2/L3. If L1 response is M, the push in SMC is killed if bus response is not ARTRY.
				M	I	I	None	
			M	I/S/E	I	I	None	
				M	I	I	None	

Table 3-37. External Snoop Responses and L1, L2, and L3 Actions (continued)

Snoop Type	L1 Snoop Type	L1 Resp	Initial L2 State	Initial L3 State	Final L2 State	Final L3 State	Response to Snoop	Comments
Read	Read	I/V	I	I	same	same	None	—
				S/E	same	S	$\overline{\text{SHDx}}$[2]	Set L1/L3 shared.
				M	same	S	$\overline{\text{SHDx}}$ + GBL	Intervene from L3 and set L1/L3 shared.
			S/E	I	S	same	$\overline{\text{SHDx}}$[2]	Set L1/L2 shared.
				S/E	S	S		Set L1/L2/L3 shared.
				M	S	S	$\overline{\text{SHDx}}$ + GBL	Intervene from L3 and set L1/L2/L3 shared.
			M	I	S	same	$\overline{\text{SHDx}}$ + GBL	Intervene from L2, consume data in L3 if valid, and set L1/L2/L3 shared.
				S/E	S	S		
				M	S	S	$\overline{\text{SHDx}}$ + GBL	
		M	I	I	same	same	$\overline{\text{SHDx}}$ + GBL	Intervene from L1, consume data in L2 and L3 if valid, and set L1/L2/L3 shared.
				S/E	same	S		
				M	same	S		
			S/E	I	S	same		
				S/E	S	S		
				M	S	S		
			M	I	S	same		
				S/E	S	S		
				M	S	S		

Table 3-37. External Snoop Responses and L1, L2, and L3 Actions (continued)

Snoop Type	L1 Snoop Type	L1 Resp	Initial L2 State	Initial L3 State	Final L2 State	Final L3 State	Response to Snoop	Comments
Clean RWNITC	Read	I/V	I	I/S/E	same	same	None	—
				M	same	E	GBL	Intervene from L3 and clean L1/L3.
			S/E	I/S/E	same	same	None	Clean L1/L2.
				M	same	E	GBL	Intervene from L3 and clean L1/L2/L3.
			M	I/S/E	E	same	GBL	Intervene from L2 and clean L1/L2/L3. Consume intervention data in L3 if valid.
				M	E	E		
		M	I	I/S/E	I	same	GBL	Intervene from L1 and clean L1/L2/L3. Consume intervention data in L2/L3 if valid.
				M	I	E		
			S/E	I/S/E	S	same		
				M	S	E		
			M	I/S/E	S	same		
				M	S	E		
RWITM RCLAIM	Flush Kill Reservation	I/S/E	I/S/E	I/S/E	I	I	None	Invalidate L1. Kill reservations (if necessary.) after ARTRY window.
				M	I	I	GBL	Intervene from L3. Invalidate L2/L2/L3. Kill reservation (if necessary.) after ARTRY window.
			M	I/S/E	I	I	GBL	Intervene from L2. Invalidate L2/L2/L3. Kill reservation (if necessary.) after ARTRY window.
				M	I	I	GBL	
		M	I/S/E	I/S/E	I	I	GBL	Intervene from L1. Invalidate L2/L2/L3. Kill reservation (if necessary.) after ARTRY window.
				M	I	I	GBL	
			M	I/S/E	I	I	GBL	
				M	I	I	GBL	
TLBIE	**tlbie**	n/a	n/a	n/a	n/a	n/a	None	Snoop core only, no L2/L3 action.
ICBI	**icbi**	n/a	n/a	n/a	n/a	n/a	None	

Table 3-37. External Snoop Responses and L1, L2, and L3 Actions (continued)

Snoop Type	L1 Snoop Type	L1 Resp	Initial L2 State	Initial L3 State	Final L2 State	Final L3 State	Response to Snoop	Comments
SYNC	None	n/a	n/a	n/a	n/a	n/a	None	The **sync** instruction does not need to query.
TLBSYNC	None	n/a	n/a	n/a	n/a	n/a	None	The **tlbsync** instruction does not need to query. It asserts a retry if and only if there is a pending marked transaction from a previous **tlbie**.
EIEIO	n/a	n/a	n/a	n/a	n/a	n/a	None	MPC7450 does not snoop **eieio**.
LWARX RESERVE	n/a	n/a	n/a	n/a	n/a	n/a	None	MPC7450 does not snoop **lwarx** Reserve.
xferdata	n/a	n/a	n/a	n/a	n/a	n/a	None	MPC7450 does not snoop xferdata in or out.

[1] Snoop W = 1 or I = 1 Write w/Flush need not kill reservations because **lwarx** is not supported in W = 1 or I = 1 space, and aliasing W = 1 and W = 0 or I = 1 and I = 0 across processors is illegal.

[2] It is possible to get a shared response to a read snoop for a transient condition. For example, if a previous flush found the data modified in the L1 or L2, the intervention that changes the L3 state to invalid may not have been performed in the L3 when the read is snooped. Since the address tenure is complete for the flush, a hit against an intervention operation in the queues is not retried. The memory system ensures ordering of the data.

Chapter 4
Exceptions

The OEA portion of the PowerPC architecture defines the mechanism by which processors implement exceptions. Exception conditions may be defined at other levels of the architecture. For example, the UISA defines conditions that may cause floating-point exceptions; the OEA defines the mechanism by which the exception is taken.

The exception mechanism allows the processor that implements the PowerPC architecture to change to supervisor state as a result of unusual conditions arising in the execution of instructions and from external signals, bus errors, or various internal conditions. When exceptions occur, information about the state of the processor is saved to certain registers and the processor begins execution at an address (exception vector) predetermined for each exception. Processing of exceptions begins in supervisor mode.

Although multiple exception conditions can map to a single exception vector, often a more specific condition may be determined by examining a register associated with the exception—for example, the DSISR and the floating-point status and control register (FPSCR). Also, software can explicitly enable or disable some exception conditions.

The PowerPC architecture requires that exceptions be taken in program order; therefore, although a particular implementation 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. In addition, if a single instruction encounters multiple exception conditions, those exceptions are taken and handled sequentially. Likewise, exceptions that are asynchronous and precise are recognized when they occur, but are not handled until all instructions currently in the execute stage successfully complete execution and report their results. To prevent loss of state information, exception handlers must save the information stored in the machine status save/restore registers, SRR0 and SRR1, soon after the exception is taken to prevent this information from being lost due to another exception being taken. Because exceptions can occur while an exception handler routine is executing, multiple exceptions can become nested. It is up to the exception handler to save the necessary state information if control is to return to the excepting program.

In many cases, after the exception handler handles an exception, there is an attempt to execute the instruction that caused the exception. Instruction execution continues until the next exception condition is encountered. Recognizing and handling exception conditions sequentially guarantees that the machine state is recoverable and processing can resume without losing instruction results.

In this book, the following terms are used to describe the stages of exception processing:

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

In this book, the term ‘interrupt’ is used to describe the external interrupt, the system management interrupt, and sometimes the asynchronous exceptions, in general. Note that the PowerPC architecture uses the word ‘exception’ to refer to IEEE-defined floating-point exception conditions that may cause a program exception to be taken; see Section 4.6.7, “Program Exception (0x00700).” The occurrence of these IEEE exceptions may or may not cause an exception to be taken. IEEE-defined exceptions are referred to as IEEE floating-point exceptions or floating-point exceptions in this book.

AltiVec Technology and the Exception Model

Only the four following exceptions may result from execution of an AltiVec instruction:

- An AltiVec unavailable exception occurs with an attempt to execute any non-stream AltiVec instruction with MSR[VEC] = 0. After this exception occurs, execution resumes at offset 0x00F20 from the physical base address indicated by MSR[IP]. This exception does not occur for data streaming instructions (**dst**[**t**], **dstst**[**t**] **dss**, and **dssall**). Also note that the VRSAVE register is not protected by this exception; this is consistent with the *AltiVec Programming Environments Manual*.
- A DSI exception occurs for an AltiVec load or store only if the load or store operation encounters a page fault (does not find a valid PTE during a table search operation) or a protection violation. Also a DSI exception occurs if an AltiVec load or store attempts to access a SR[T] = 1 (direct-store) memory location.
- An AltiVec assist exception may occur if an AltiVec floating-point instruction detects denormalized data as an input or output in Java mode. After this exception occurs, execution resumes at offset 0x01600 from the physical base address indicated by MSR[IP].
- AltiVec loads and stores

NOTE

> The 60x bus protocol does not support a 16-byte bus transaction. Therefore, cache-inhibited AltiVec loads and stores and write-through stores take an alignment exception. This requires a re-write of the alignment exception routines in software that supports AltiVec quad word access in 60x bus mode on the MPC7450.

4.1 MPC7450 Microprocessor Exceptions

As specified by the PowerPC architecture, exceptions can be either precise or imprecise and either synchronous or asynchronous. Asynchronous exceptions are caused by events external to the processor's execution; synchronous exceptions are caused by instructions.

The types of exceptions are shown in Table 4-1. Note that all exceptions except for the performance monitor, AltiVec unavailable, instruction address breakpoint, system management, AltiVec assist, and the 3 software table search exceptions are described in Chapter 6, "Exceptions," in the *Programming Environments Manual*.

Table 4-1. MPC7450 Microprocessor Exception Classifications

Synchronous/Asynchronous	Precise/Imprecise	Exception Types
Asynchronous, nonmaskable	Imprecise	System reset, machine check
Asynchronous, maskable	Precise	External interrupt, system management interrupt, decrementer exception, performance monitor exception
Synchronous	Precise	Instruction-caused exceptions

The exception classifications are discussed in greater detail in Section 4.2, "MPC7450 Exception Recognition and Priorities." For a better understanding of how the MPC7450 implements precise exceptions, see Chapter 6, "Instruction Timing." Exceptions implemented in the MPC7450, and conditions that cause them, are listed in Table 4-2. Table 4-2 notes when an exception is implementation-specific to the MPC7450. The three software table search exceptions are used by the MPC7450 when HID0[STEN] = 1, to support the software page table searching. Refer to Section 4.6.15, "TLB Miss Exceptions," and Chapter 5, "Memory Management," for more information about the software table search operations.

Table 4-2. Exceptions and Conditions

Exception Type	Vector Offset	Causing Conditions
Reserved	0x00000	—
System reset	0x00100	Assertion of either $\overline{\text{HRESET}}$ or $\overline{\text{SRESET}}$ or at power-on reset
Machine check	0x00200	Assertion of $\overline{\text{TEA}}$ during a data bus transaction, assertion of $\overline{\text{MCP}}$, an address bus parity error on MPX bus, a data bus parity error on MPXbus, an L1 instruction cache error, and L1 data cache error, a memory subsystem detected error including the following: • L2 data parity error • L2 tag parity error • L3 SRAM error • L3 tag parity error • Single-bit and multiple-bit L2 ECC errors MSR[ME] must be set. Note that the L3 cache is not supported on the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448.

Table 4-2. Exceptions and Conditions (continued)

Exception Type	Vector Offset	Causing Conditions
DSI	0x00300	As specified in the PowerPC architecture. Also includes the following: • A hardware table search due to a TLB miss on load, store, or cache operations results in a page fault. • Any load or store to a direct-store segment (SR[T] = 1). • A **lwarx** or **stwcx.** instruction to memory with cache-inhibited or write-through memory/cache access attributes.
ISI	0x00400	As specified in the PowerPC architecture
External interrupt	0x00500	MSR[EE] = 1 and $\overline{\text{INT}}$ is asserted
Alignment	0x00600	• A floating-point load/store, **stmw**, **stwcx.**, **lmw**, **lwarx**, **eciwx**, or **ecowx** instruction operand is not word-aligned. • A multiple/string load/store operation is attempted in little-endian mode • An operand of a **dcbz** instruction is on a page that is write-through or cache-inhibited for a virtual mode access. • An attempt to execute a **dcbz** instruction occurs when the cache is disabled or locked.
Program	0x00700	As specified in the PowerPC architecture
Floating-point unavailable	0x00800	As specified in the PowerPC architecture
Decrementer	0x00900	As defined by the PowerPC architecture, when the msb of the DEC register changes from 0 to 1 and MSR[EE] = 1.
Reserved	0x00A00–00BFF	—
System call	0x00C00	Execution of the System Call (**sc**) instruction
Trace	0x00D00	MSR[SE] =1 or a branch instruction is completing and MSR[BE] =1. The MPC7450 operates as specified in the OEA by taking this exception on an **isync**.
Reserved	0x00E00	The MPC7450 does not generate an exception to this vector. Other processors that implement the PowerPC architecture may use this vector for floating-point assist exceptions.
Reserved	0x00E10–00EFF	—
Performance monitor	0x00F00	The limit specified in PMC*n* is met and MMCR0[ENINT] = 1 (MPC7450-specific).
AltiVec unavailable	0x00F20	Occurs due to an attempt to execute any non-streaming AltiVec instruction when MSR[VEC] = 0. This exception is not taken for data streaming instructions (**dst*x***, **dss,** or **dssall**) (MPC7450-specific).
ITLB miss	0x01000	An instruction translation miss exception is caused when HID0[STEN] = 1 and the effective address for an instruction fetch cannot be translated by the ITLB (MPC7450-specific).
DTLB miss-on-load	0x01100	A data load translation miss exception is caused when HID0[STEN] = 1 and the effective address for a data load operation cannot be translated by the DTLB (MPC7450-specific).

Table 4-2. Exceptions and Conditions (continued)

Exception Type	Vector Offset	Causing Conditions
DTLB miss-on-store	0x01200	A data store translation miss exception is caused when HID0[STEN] = 1 and the effective address for a data store operation cannot be translated by the DTLB, or when a DTLB hit occurs, and the changed bit in the PTE must be set due to a data store operation (MPC7450-specific).
Instruction address breakpoint	0x01300	IABR[0–29] matches EA[0–29] of the next instruction to complete and IABR[BE] = 1 (MPC7450-specific).
System management interrupt	0x01400	MSR[EE] = 1 and $\overline{\text{SMI}}$ is asserted (MPC7450-specific).
Reserved	0x01500–015FF	—
AltiVec assist	0x01600	This MPC7450-specific exception supports denormalization detection in Java mode as specified in the *AltiVec Technology Programming Environments Manual*.
Reserved	0x01700–02FFF	—

4.2 MPC7450 Exception Recognition and Priorities

Exceptions are roughly prioritized by exception class, as follows:

1. Nonmaskable, asynchronous exceptions such as system reset and machine check exceptions, have priority over all other exceptions although the machine check exception condition can be disabled so the condition causes the processor to go directly into the checkstop state. These exceptions cannot be delayed and do not wait for completion of any precise exception handling.
2. Synchronous, precise exceptions are caused by instructions and are taken in strict program order.
3. Imprecise exceptions (imprecise mode floating-point enabled exceptions) are caused by instructions and they are delayed until higher priority exceptions are taken. Note that the MPC7450 does not implement an exception of this type.
4. Maskable asynchronous exceptions (external interrupt, decrementer, system management interrupt, and performance monitor exceptions) are delayed until higher priority exceptions are taken.

The following list of categories describes how the MPC7450 handles exception conditions up to the point that the exception is taken. Note that a recoverable state is reached if the completed store queue is empty and any instruction that is next in program order, and has been signaled to complete, has completed. If MSR[RI] = 0, the MPC7450 is in a nonrecoverable state. Also, instruction completion is defined as updating all architectural registers associated with that instruction, and then removing that instruction from the completion buffer. When all the pending store instructions have been committed to memory, the completed store queue is empty.

- Exceptions caused by asynchronous events (interrupts). These exceptions are further distinguished by whether they are maskable and recoverable.
  - Asynchronous, nonmaskable, nonrecoverable

    System reset for assertion of $\overline{\text{HRESET}}$—Has highest priority and is taken immediately regardless of other pending exceptions or recoverability (includes power-on reset).
  - Asynchronous, maskable, nonrecoverable

    Machine check exception—Has priority over any other pending exception except system reset for assertion of $\overline{\text{HRESET}}$ (or power-on reset). Taken immediately regardless of recoverability.
  - Asynchronous, nonmaskable, recoverable

    System reset for $\overline{\text{SRESET}}$—Has priority over any other pending exception except system reset for $\overline{\text{HRESET}}$ (or power-on reset), or machine check. Taken immediately when a recoverable state is reached.
  - Asynchronous, maskable, recoverable

    System management interrupt, performance monitor, external interrupt, and decrementer exceptions—Before handling this type of exception, the next instruction in program order must complete. If that instruction causes another type of exception, that exception is taken and the asynchronous, maskable recoverable exception remains pending until the instruction completes. Further instruction completion is halted. The asynchronous, maskable recoverable exception is taken when a recoverable state is reached.
- Instruction-related exceptions. These exceptions are further organized into the point in instruction processing at which they generate an exception.
  - Instruction fetch and ITLB miss
    - ISI exceptions—Once this type of exception is detected, fetching stops and the current instruction stream is allowed to drain out of the machine. If completing any of the instructions in this stream causes an exception, that exception is taken and the instruction fetch exception is discarded, but may be encountered again when instruction processing resumes. Otherwise, once all pending instructions have executed and a recoverable state is reached, the ISI or ITLB miss exception is taken.
  - Instruction dispatch/execution
    - Program, DSI, alignment, floating-point unavailable, AltiVec unavailable, AltiVec assist, system call, instruction address breakpoint, data address breakpoint, and DTLB miss (if HID0[STEN] = 1)—This type of exception is determined during dispatch or execution of an instruction. The exception remains pending until all instructions before the exception-causing instruction in program order complete. The exception is then taken without completing the exception-causing instruction. If completing these previous instructions causes an exception, that exception takes priority over the pending instruction dispatch/execution exception, which is discarded, but may be encountered again when instruction processing resumes.

- Post-instruction execution
  - Trace—Trace exceptions are generated following execution and completion of an instruction while trace mode is enabled. If executing the instruction produces conditions for another type of exception, that exception is taken and the post-instruction exception is ignored for that instruction.

Note that these exception classifications correspond to how exceptions are prioritized, as described in Table 4-3.

Table 4-3. MPC7450 Exception Priorities

Priority	Exception	Cause
Asynchronous Exceptions (Interrupts)		
0	System reset	Power-on reset, assertion of $\overline{\mathrm{HRESET}}$ and $\overline{\mathrm{TRST}}$ (hard reset)
1	Machine check	Any enabled machine check condition (assertion of $\overline{\mathrm{TEA}}$ or $\overline{\mathrm{MCP}}$, or memory subsystem error as defined in MSSSR0 (see Section 2.2.5.4, “Memory Subsystem Status Register (MSSSR0),” for further details), address or data parity error, L1 address or data parity error, data cache error, instruction cache error, L2 data parity error, L2 tag error, L2 tag parity error, L3 SRAM error, L3 tag parity error, single-bit and multiple-bit L2 ECC error [1])
2	System reset	Assertion of $\overline{\mathrm{SRESET}}$ (soft reset)
3	System management interrupt	Assertion of $\overline{\mathrm{SMI}}$
4	External interrupt	Assertion of $\overline{\mathrm{INT}}$
5	Performance monitor	Any programmer-specified performance monitor condition
6	Decrementer	Decrementer passes through zero.
Instruction Fetch Exceptions		
0	ISI	ISI exception conditions due to: 1. No-execute segment 2. Direct-store (T=1) segment
1	ITLB Miss	Instruction table miss exception due to miss in ITLB with HID0[STEN] = 1
2	ISI	ISI exception conditions due to: 1. Effective address can not be translated (page fault) 2. Instruction fetch from guarded memory 3. Protection violation
Instruction Dispatch/Execution Exceptions		
0	Instruction address breakpoint	Any instruction address breakpoint exception condition
1	Program	Illegal instruction, privileged instruction, or trap exception condition. Note that floating-point enabled program exceptions have lower priority.
2	System call	System call (**sc**) instruction

Table 4-3. MPC7450 Exception Priorities (continued)

Priority	Exception	Cause
3	Floating-point unavailable	Any floating-point unavailable exception condition
4	AltiVec unavailable	Any unavailable AltiVec exception condition
5	Program	A floating-point enabled exception condition (lowest-priority program exception)
6	Alignment	Any alignment exception condition, prioritized as follows: 1. Floating-point access not word-aligned 2. **lmw**, **stmw**, **lwarx**, or **stwcx.** not word-aligned 3. **eciwx** or **ecowx** not word-aligned 4. Multiple or string access with MSR[LE] set 5. **dcbz** to a locked or disabled L1 data cache, WT, or CI page 6. **stvx**, **stvxl**, **lvx**, or **lvxl** to a disabled L1 cache, or all ways locked when in 60x bus mode
7	DSI	DSI exception due to execution of **stvx**, **stvxl**, **lvx**, or **lvxl** with all of the following conditions: • SR[T] =0 (with BAT miss) • cache-inhibited or write-through space • 60x bus mode.
8	Alignment	Alignment exception due to execution of **stvx**, **stvxl**, **lvx**, or **lvxl** with all of the following conditions: • SR[T]=1(with BAT miss) • cache-inhibited or write-through space • 60x bus mode. exception due to **stvx**, **stvxl**, **lvx**, or **lvxl** to cache-inhibited or write-through page when in 60x bus mode on a BAT hit or to SR[T] = 0 space.
9	DSI	DSI exception due to **eciwx** or **ecowx** with EAR[E] = 0 (DSISR[11]).
10	DSI	DSI exception due to **lwarx**/**stwcx.** with caching disabled or if all ways are locked.
11	DSI	DSI exception due to the following: • BAT/page protection violation (DSISR[4]), or • **lwarx**/**stwcx.** to BAT entry with write-through attributes (W = 1), or to a page table entry (or BAT entry) with caching disallowed attributes (I = 1), or to a page table entry (or BAT entry) with caching-allowed attributes (I = 0), but with a locked L1 data cache (DSISR[5]) Note that if both occur simultaneously, both bits 4 and 5 of the DSISR are set.
12	DSI	DSI exception due to any access except cache operations to a segment where SR[T] = 1 (DSISR[5]) or an access crosses from a T = 0 segment to one where T = 1 (DSISR[5])
13	DTLB miss on store	Data table miss on store exception due to store miss in DTLB with HID0[STEN] = 1
14	DTLB miss-on-load	Data table miss-on-load exception due to load miss in DTLB with HID0[STEN] = 1
15	DSI	DSI exception due to: • TLB translation detects page protection violation (DSISR[4]) • TLB translation detects **lwarx**/**stwcx.** to a page table entry with write-through attributes (W = 1), or to a page table entry (or BAT entry) with caching disallowed attributes (I = 1), or to a page table entry (or BAT entry) with caching-allowed attributes (I = 0), but with a locked L1 data cache (DSISR[5]). • Hardware table search page fault (DSISR[1]) Note that if both 1 and 2 occur simultaneously, both bits 4 and 5 of the DSISR are set.

Table 4-3. MPC7450 Exception Priorities (continued)

Priority	Exception	Cause
16	DTLB miss on store (data store access and C bit = 0)	Data TLB miss on store exception when HID0[STEN] = 1 and the PTE changed bit is not set (C = 0) for a store operation.
17	DSI	DSI exception due to DABR address match (DSISR[9]). Note that even though DSISR[5] and DSISR[9] are set by exceptions with different priorities, they can be set simultaneously.
18	AltiVec assist	Denormalized data detected as input or output in the AltiVec vector floating-point unit (VFPU) while in Java mode (VSCR[NJ] = 0)
Post-Instruction Execution Exceptions		
19	Trace	MSR[SE] = 1 (or MSR[BE] = 1 for branches)

[1] L2 ECC option available in the MPC7448.

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 an interrupted exception may be lost; therefore, these exceptions are typically nonrecoverable. An exception may or may not be taken immediately when it is recognized.

4.3 Exception Processing

When an exception is taken, the processor uses SRR0 and SRR1 to save the contents of the MSR for the current context and to identify where instruction execution should resume after the exception is handled.

When an exception occurs, the address saved in SRR0 helps determine where instruction processing should resume when the exception handler returns control to the interrupted process. Depending on the exception, this may be the address in SRR0 or at the next address in the program flow. All instructions in the program flow preceding this one will have completed execution and no subsequent instruction will have begun execution. This may be the address of the instruction that caused the exception or the next one (as in the case of a system call or trace exception). The SRR0 register is shown in Figure 4-1.

SRR0 (Holds EA for Instruction in Interrupted Program Flow)	
0	31

Figure 4-1. Machine Status Save/Restore Register 0 (SRR0)

SRR1 is used to save machine status (selected MSR bits and possibly other status bits) on exceptions and to restore those values when an **rfi** instruction is executed. SRR1 is shown in Figure 4-2.

Exception-Specific Information and MSR Bit Values	
0	31

Figure 4-2. Machine Status Save/Restore Register 1 (SRR1)

Typically, when an exception occurs, SRR1[0–15] are loaded with exception-specific information and MSR[16–31] are placed into the corresponding bit positions of SRR1. For most exceptions, SRR1[0–5] and SRR1[7–15] are cleared, and MSR[6, 16–31] are placed into the corresponding bit positions of SRR1. Table 4-4 provides a summary of the SRR1 bit settings when a machine check exception occurs. For a specific exception’s SRR1 bit settings, see Section 4.6, “Exception Definitions.”

The MPC7450’s MSR is shown in Figure 4-3.

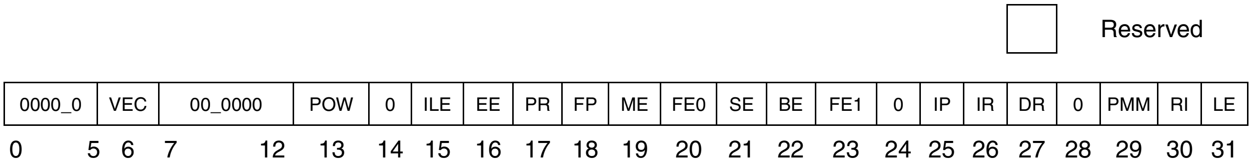


Figure 4-3. Machine State Register (MSR)

The MSR bits are defined in Table 4-4.

Table 4-4. MSR Bit Settings

Bit(s)	Name	Description
0–5	—	Reserved
6	VEC [1, 2]	AltiVec vector unit available 0 The processor prevents dispatch of AltiVec instructions (excluding the data streaming instructions—**dst**, **dstt**, **dstst**, **dststt**, **dss**, and **dssall)**. The processor also prevents access to the vector register file (VRF) and the vector status and control register (VSCR). Any attempt to execute an AltiVec instruction that accesses the VRF or VSCR, excluding the data streaming instructions generates the AltiVec unavailable exception. The data streaming instructions are not affected by this bit; the VRF and VSCR registers are available to the data streaming instructions even when the MSR[VEC] is cleared. 1 The processor can execute AltiVec instructions and the VRF and VSCR registers are accessible to all AltiVec instructions. Note that the VRSAVE register is not protected by MSR[VEC].
7–12	—	Reserved
13	POW [1, 3]	Power management enable 0 Power management disabled (normal operation mode). 1 Power management enabled (reduced power mode). Power management functions are implementation-dependent. See Chapter 10, “Power and Thermal Management.”
14	—	Reserved. Implementation-specific
15	ILE	Exception little-endian mode. When an exception occurs, this bit is copied into MSR[LE] to select the endian mode for the context established by the exception.
16	EE	External interrupt enable 0 The processor delays recognition of external interrupts and decrementer exception conditions. 1 The processor is enabled to take an external interrupt or the decrementer exception.
17	PR [4]	Privilege level 0 The processor can execute both user- and supervisor-level instructions. 1 The processor can only execute user-level instructions.
18	FP [2]	Floating-point available 0 The processor prevents dispatch of floating-point instructions, including floating-point loads, stores, and moves. 1 The processor can execute floating-point instructions and can take floating-point enabled program exceptions.

Table 4-4. MSR Bit Settings (continued)

Bit(s)	Name	Description
19	ME	Machine check enable 0 Machine check exceptions are disabled. 1 Machine check exceptions are enabled.
20	FE0 [2]	IEEE floating-point exception mode 0 (see Table 4-5)
21	SE	Single-step trace enable 0 The processor executes instructions normally. 1 The processor generates a single-step trace exception upon the successful execution of every instruction except **rfi** and **sc**. Successful execution means that the instruction caused no other exception.
22	BE	Branch trace enable 0 The processor executes branch instructions normally. 1 The processor generates a branch type trace exception when a branch instruction executes successfully.
23	FE1 [2]	IEEE floating-point exception mode 1 (see Table 4-5)
24	—	Reserved. This bit corresponds to the AL bit of the POWER architecture.
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. 0 Exceptions are vectored to the physical address 0x000*n_nnnn*. 1 Exceptions are vectored to the physical address 0xFFF*n_nnnn*.
26	IR [5]	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 [4]	Data address translation 0 Data address translation is disabled. 1 Data address translation is enabled. For more information see Chapter 5, “Memory Management.”
28	—	Reserved
29	PMM [1]	Performance monitor marked mode 0 Process is not a marked process. 1 Process is a marked process. This bit can be set when statistics need to be gathered on a specific (marked) process. The statistics will only be gathered when the marked process is executing. MPC7451-specific; defined as optional by the PowerPC architecture. For more information about the performance monitor marked mode bit, see Section 11.4, “Event Counting.”
30	RI	Indicates whether system reset or machine check exception is recoverable. 0 Exception is not recoverable. 1 Exception is recoverable. The RI bit indicates whether from the perspective of the processor, it is safe to continue (that is, processor state data such as that saved to SRR0 is valid), but it does not guarantee that the interrupted process is recoverable.
31	LE [6]	Little-endian mode enable 0 The processor runs in big-endian mode. 1 The processor runs in little-endian mode.

[1] Optional to the PowerPC architecture

[2] A context synchronizing instruction must follow a mtmsr instruction.

[3] A dssall and sync must precede a mtmsr instruction and then a context synchronizing instruction must follow.

[4] A dssall and sync must precede a mtmsr and then a sync and context synchronizing instruction must follow. Note that if a user is not using the AltiVec data streaming instructions, then a dssall is not necessary prior to accessing the MSR[DR] or MSR[PR] bit.

[5] A context synchronizing instruction must follow a mtmsr. When changing the MSR[IR] bit the context synchronizing instruction must reside at both the untranslated and the translated address following the mtmsr.

[6] A dssall and sync must precede an rfi to guarantee a solid context boundary. Note that if a user is not using the AltiVec data streaming instructions, then a dssall is not necessary prior to accessing the MSR[LE] bit.

Note that setting MSR[EE] masks not only the architecture-defined external interrupt and decrementer exceptions but also the MPC7451-specific system management, and performance monitor exceptions.

The IEEE floating-point exception mode bits (FE0 and FE1) together define whether floating-point exceptions are handled precisely, imprecisely, or whether they are taken at all. As shown in Table 4-5, if either FE0 or FE1 are set, the MPC7450 treats exceptions as precise. MSR bits are guaranteed to be written to SRR1 when the first instruction of the exception handler is encountered. For further details, see Chapter 2, “PowerPC Register Set” and Chapter 6, “Exceptions,” of the *Programming Environments Manual*.

Table 4-5. IEEE Floating-Point Exception Mode Bits

FE0	FE1	Mode
0	0	Floating-point exceptions disabled
0	1	Imprecise nonrecoverable. For this setting, the MPC7450 operates in floating-point precise mode.
1	0	Imprecise recoverable. For this setting, the MPC7450 operates in floating-point precise mode.
1	1	Floating-point precise mode

4.3.1 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 as follows:

- System reset exceptions cannot be masked.
- 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 the following bits: HID1[EMCP], HID1[EBA], HID1[EBD], ICTRL[EIEC], ICTRL[EDCE], L2CR[L2PE], L3CR[L3PE], L3CR[L3APE], L2ERRINTEN[TPARINTEN], L2ERRINTEN[MBECCINTEN], and L2ERRINTEN[SBECCINTEN], which are described in Table 4-8. Note that the L3 cache is not supported on the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448.
- Asynchronous, maskable exceptions (such as the external interrupt and decrementer) are enabled by setting MSR[EE]. When MSR[EE] = 0, recognition of these exception conditions is delayed. MSR[EE] is cleared automatically when an exception is taken to delay recognition of conditions causing those exceptions.
- The performance monitor exception is enabled for a specific process by setting MSR[PMM].
- The floating-point unavailable exception can be masked by setting MSR[FP].
- The AltiVec unavailable exception can be masked by setting MSR[VEC].

- IEEE floating-point enabled exceptions (a type of program exception) are ignored when both MSR[FE0] and MSR[FE1] are cleared. If either bit is set, all IEEE enabled floating-point exceptions are taken and cause a program exception.
- The trace exception is enabled by setting either MSR[SE] or MSR[BE].
- The software tablewalk exceptions can be prevented by clearing HID0[STEN]. Note that this forces hardware tablewalks to be performed. See Section 4.6.15, "TLB Miss Exceptions," for more information.

4.3.2 Steps for Exception Processing

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

1. SRR0 is loaded with an instruction address that depends on the type of exception. See the individual exception description for details about how this register is used for specific exceptions.
2. SRR1[0, 7–9] are cleared;
   SRR1[1–5, 10–15] are loaded with information specific to the exception type;
   and SRR1[6, 16–31] are loaded with a copy of the corresponding MSR bits.
3. The MSR is set as described in Table 4-6. The new values take effect as the first instruction of the exception-handler routine is fetched.

   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.
4. Instruction fetch and execution resumes, using the new MSR value, at a location specific to the exception type. The location is determined by adding the exception's vector (see Table 4-2) to the base address determined by MSR[IP]. If IP is cleared, exceptions are vectored to the physical address 0x000*n_nnnn*. If IP is set, exceptions are vectored to the physical address 0xFFF*n_nnnn*. For a machine check exception that occurs when MSR[ME] = 0 (machine check exceptions are disabled), the checkstop state is entered (the machine stops executing instructions). See Section 4.6.2, "Machine Check Exception (0x00200)."

4.3.3 Setting MSR[RI]

An operating system may handle MSR[RI] as follows:

- In the machine check and system reset exceptions—If MSR[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 SRR0 and SRR1 appropriately, and then execute **rfi**.
- Note that the RI bit being set indicates that, with respect to the processor, enough processor state data remains valid for the processor to continue, but it does not guarantee that the interrupted process can resume.

4.3.4 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.
- Previous instructions complete execution in the context (privilege, protection, and address translation) under which they were issued.
- The **rfi** instruction copies SRR1 bits back into the MSR.
- Instructions fetched after this instruction execute in the context established by this instruction.
- Program execution resumes at the instruction indicated by SRR0.

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

4.4 Process Switching

The following instructions are useful for restoring proper context during process switching:

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

The operating system should set MSR[RI] as described in Section 4.3.3, “Setting MSR[RI].”

4.5 Data Stream Prefetching and Exceptions

As described in Chapter 5, “Cache, Exceptions, and Memory Management,” of the *AltiVec Technology Programming Environments Manual*, exceptions do not automatically cancel data stream prefetching. The operating system must stop streams explicitly when warranted—for example, when switching processes or changing virtual memory context. Care must be taken if data stream prefetching is used while in supervisor mode (MSR[PR] = 0).

4.6 Exception Definitions

Table 4-6 shows all the types of exceptions that can occur with the MPC7450 and the MSR settings when the processor goes into supervisor mode due to an exception. Depending on the exception, certain of these bits are stored in SRR1 when an exception is taken.

Table 4-6. MSR Setting Due to Exception

Exception Type	MSR Bit Name MSR Bit Number																
	VEC 6	POW 13	ILE 15	EE 16	PR 17	FP 18	ME 19	FE0 20	SE 21	BE 22	FE1 23	IP 25	IR 26	DR 27	PM 29	RI 30	LE 31
System reset	0	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	0	ILE
DSI	0	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	0	ILE
External interrupt	0	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	0	ILE
Program	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	0	ILE
Floating-point unavailable	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	0	ILE
Decrementer	0	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	0	ILE
Trace exception	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	0	ILE
Performance monitor	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	0	ILE
AltiVec unavailable	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	0	ILE
ITLB miss	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	0	ILE
DTLB miss on load	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	0	ILE
DTLB miss on store	0	0	—	0	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	0	ILE

Table 4-6. MSR Setting Due to Exception (continued)

Exception Type	MSR Bit Name MSR Bit Number																
	VEC 6	**POW 13**	**ILE 15**	**EE 16**	**PR 17**	**FP 18**	**ME 19**	**FE0 20**	**SE 21**	**BE 22**	**FE1 23**	**IP 25**	**IR 26**	**DR 27**	**PM 29**	**RI 30**	**LE 31**
System management	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	0	ILE
AltiVec assist	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	0	ILE

Key: 0 Bit is cleared
ILE Bit is copied from the MSR[ILE]
— Bit is not altered
Reserved bits are read as if written as 0

The setting of the exception prefix bit (IP) determines how exceptions are vectored. If the bit is cleared, exceptions are vectored to the physical address 0x000*n_nnnn* (where *n_nnnn* is the vector offset); if IP is set, exceptions are vectored to physical address 0xFFF*n_nnnn*. Table 4-2 shows the exception vector offset of the first instruction of the exception handler routine for each exception type.

4.6.1 System Reset Exception (0x00100)

The MPC7450 implements the system reset exception as defined in the PowerPC architecture (OEA). The system reset exception is a nonmaskable, asynchronous exception signaled to the processor through the assertion of system-defined signals. In the MPC7450, the exception is signaled by the assertion of either the $\overline{\text{HRESET}}$ or $\overline{\text{SRESET}}$ input signals, described more fully in Chapter 8, “Signal Descriptions.”

A hard reset is initiated by asserting $\overline{\text{HRESET}}$. A hard reset is used primarily for power-on reset (POR) (in which case $\overline{\text{TRST}}$ must also be asserted), but can also be used to restart a running processor. The $\overline{\text{HRESET}}$ signal must be asserted during power up and must remain asserted for a period that allows the PLL to achieve lock and the internal logic to be reset. This period is specified in the hardware specifications. If $\overline{\text{HRESET}}$ is asserted for less than the required interval, the results are not predictable.

If a hard reset request occurs ($\overline{\text{HRESET}}$ asserted), the processor immediately branches to the system reset exception vector (0xFFF0_0100) without attempting to reach a recoverable state. If $\overline{\text{HRESET}}$ is asserted during normal operation, all operations cease and the machine state is lost. The MPC7450 internal state after a hard reset is defined in Table 2-45.

A soft reset is initiated by asserting $\overline{\text{SRESET}}$. If $\overline{\text{SRESET}}$ is asserted, the processor is first put in a recoverable state. To do this, the MPC7450 allows any instruction at the point of completion to either complete or take an exception (note that load/store string or multiple accesses are not split), blocks completion of any following instructions and allows the completion queue to empty. If the soft reset request is made while the MPC7450 is in trace mode (MSR[SE] = 1 or MSR[BE] = 1), the exception is set as nonrecoverable and SRR1[30] is cleared (SRR1[30] = 0). The state before the exception occurred is then saved as specified in the PowerPC architecture and instruction fetching begins at the system reset exception vector offset, 0x00100. The vector base address for a soft reset depends on the setting of MSR[IP] (either 0x0000_0100 or 0xFFF0_0100). Soft resets are third in priority, after hard reset and machine check. Except for the trace mode condition, this exception is recoverable provided attaining a recoverable state does not generate a machine check.

$\overline{\text{SRESET}}$ is an edge-sensitive signal that can be asserted and negated asynchronously, provided there are two bus cycles in between, see Section 8.4.3.4.1, “Soft Reset (SRESET)—Input,” for more details. The system reset exception modifies the MSR, SRR0, and SRR1, as described in the *Programming Environments Manual*. Unlike hard reset, soft reset does not directly affect the states of output signals. Attempts to use $\overline{\text{SRESET}}$ during a hard reset sequence or while the JTAG logic is non-idle can cause unpredictable results.

The MPC7450 implements HID0[NHR], which helps software distinguish a hard reset from a soft reset. Because this bit is cleared by a hard reset, but not by a soft reset, software can set this bit after a hard reset and determine whether a subsequent reset is a hard or soft reset (by examining whether this bit is still set). See Section 2.2.5.1, “Hardware Implementation-Dependent Register 0 (HID0).”

Table 4-7 lists register settings when a system reset exception is taken.

Table 4-7. System Reset Exception—Register Settings

Register	Setting Description			
SRR0	Cleared to zero by a hard reset On a soft reset, set to the effective address of the instruction that the processor would have attempted to execute next if no exception conditions were present.			
SRR1	0–5 Cleared 6 Loaded with equivalent MSR bit 7–15 Cleared 16–31 Loaded with equivalent MSR bits Note that if the processor state is corrupted to the extent that execution cannot resume reliably, MSR[RI] (SRR1[30]) is cleared.			
MSR	VEC 0 POW 0 ILE — EE 0 LE ILE	PR 0 FP 0 ME — FE0 0	SE 0 BE 0 FE1 0 IP —	IR 0 DR 0 PM 0 RI 0

Key:	0	Bit is cleared
	ILE	Bit is copied from the MSR[ILE]
	—	Bit is not altered

4.6.2 Machine Check Exception (0x00200)

The MPC7450 implements the machine check exception as defined in the PowerPC architecture (OEA). The MPC7450 conditionally initiates a machine check exception if MSR[ME] = 1 and a system bus error ($\overline{\text{TEA}}$ assertion on data bus), assertion of the machine check ($\overline{\text{MCP}}$) signal, address bus parity error on MPXbus, data bus parity error on MPX bus, L1 data cache error, L1 instruction cache error, or a memory subsystem error is detected including:

- L2 data parity error
- L2 tag parity error
- L3 SRAM error

- L3 tag parity errors
- Single-bit and multiple-bit L2 ECC errors

As defined in the PowerPC architecture, the exception is not taken if MSR[ME] is cleared, in which case the processor enters a checkstop state.

Certain machine check conditions can be enabled and disabled using HID1, ICTRL, L2CR, and L3CR bits, as described in Table 4-8. Note that the L3 cache and the L3 cache interface are not supported on the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448.

Table 4-8. Machine Check Enable Bits

Bit	Name	Function
HID1[0]	EMCP	Enable $\overline{\text{MCP}}$. The primary purpose of this bit is to mask further machine check exceptions caused by assertion of $\overline{\text{MCP}}$, similar to how MSR[EE] can mask external interrupts. 0 Masks $\overline{\text{MCP}}$. Assertion of $\overline{\text{MCP}}$ does not generate a machine check exception or a checkstop. 1 Assertion of $\overline{\text{MCP}}$ causes a checkstop if MSR[ME] = 0 or a machine check exception if MSR[ME] = 1.
HID1[2]	EBA	Enable/disable 60x/MPX bus address parity checking. 0 Prevents address parity checking. 1 Allows an address parity error to cause a checkstop if MSR[ME] = 0 or a machine check exception if MSR[ME] = 1. EBA and EBD allow the processor to operate with memory subsystems that do not generate parity.
HID1[3]	EBD	Enable 60x/MPX bus data parity checking 0 Parity checking is disabled. 1 Allows a data parity error to cause a checkstop if MSR[ME] = 0 or a machine check exception if MSR[ME] = 1. EBA and EBD allow the processor to operate with memory subsystems that do not generate parity.
ICTRL[4]	EIEC	Instruction cache parity error enable 0 When the bit is cleared, any parity error in the L1 instruction cache is masked and does not cause machine checks or checkstop 1 Enables instruction cache parity errors. When an instruction cache parity error occurs, a machine check exception is taken if MSR[ME] = 1. When this condition occurs, SRR1[1] is set. For details on the machine check exception see Section 4.6.2, “Machine Check Exception (0x00200).”
ICTRL[5]	EDCE	Data cache parity error enable 0 When the bit is cleared, any parity error in the L1 data cache is masked and does not cause machine checks or checkstop 1 Enables data cache parity errors. When a data cache parity error occurs, a machine check exception is taken if MSR[ME] = 1. When this condition occurs, SRR1[2] is set. For details on the machine check exception see Section 4.6.2, “Machine Check Exception (0x00200).”

Table 4-8. Machine Check Enable Bits (continued)

Bit	Name	Function
L2CR[1]	L2PE	L2 tag and data parity checking enable 0 L2 tag and data parity disabled 1 L2 tag and data parity enabled Enables or disables the checking of L2 tag and data parity
		L2 data parity checking enable for the MPC7448 0 L2 data parity checking disabled. 1 L2 data parity checking enabled if L2ERRDIS[MBECCDIS]=1 and L2ERRDIS[SBECCDIS]=1. If ECC is enabled (L2ERRDIS[MBECCDIS]=0 or L2ERRDIS[SBECCDIS]=0), setting L2PE has no effect; ECC checking will still be performed. Note: MPC7450–MPC7447A used this bit to enable/disable tag parity checking as well as data parity checking. MPC7448 has moved tag parity enable/disable to the new L2ERRDIS register. Data parity can only be enabled with L2CR[L2PE] if ECC is disabled in the L2ERRDIS register. By default, tag parity and data ECC checking are enabled on MPC7448.
L3CR[1][1]	L3PE	L3 data parity checking enable 0 L3 odd data parity checking disabled 1 L3 odd data parity checking enabled Enables odd parity checking for the L3 data RAM interface. When L3PE is set, it allows a data parity error on the L3 interface to cause a checkstop if MSR[ME] = 0 or a machine check exception if MSR[ME] = 1. The MPC7450 always generates L3 data parity.
L3CR[2] [1]	L3APE	L3 address parity checking enable 0 L3 address parity checking disabled 1 L3 address parity checking enabled Enables odd parity checking for the L3 address bus interface. When L3APE is set, it allows an address parity error on either the on-chip tags or the L3 address bus to cause a checkstop if MSR[ME] = 0 or a machine check exception if MSR[ME] = 1. The MPC7450 always generates L3 address parity.
L2ERRINTEN[28] [2]	MBECCINTEN[3]	Multiple-bit ECC error reporting enable 0 Multiple-bit ECC error reporting disabled 1 Multiple-bit ECC error reporting enabled
L2ERRINTEN[29] [2]	SBECCINTEN[3]	Single-bit ECC error reporting enable 0 Single-bit ECC error reporting disabled 1 Single-bit ECC error reporting enabled
L2ERRINTEN[27] [2]	TPARINTEN[3]	Tag parity error reporting enable 0 Tag parity error reporting disabled 1 Tag parity error reporting enabled.

[1] Note that the L3 cache is not supported on the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448.

[2] MPC7448-specific bit.

[3] The corresponding bit in L2ERRDIS must be cleared (enabled).

A $\overline{\text{TEA}}$ indication on the bus can result from any load or store operation initiated by the processor. In general, $\overline{\text{TEA}}$ is expected to be used by a memory controller to indicate that a memory parity error or an uncorrectable memory ECC error has occurred. Note that the resulting machine check exception is imprecise and unordered with respect to the instruction that originated the bus operation.

For other memory subsystem errors, if MSR[ME] and the appropriate HID1, ICTRL, L2CR, and L3CR bits are set, the exception is recognized and handled; otherwise, in most cases, the processor generates an internal checkstop condition (an example of an exception to this rule is if MSR[ME] = 1, HID1[EMCP] = 0, and $\overline{\text{MCP}}$ is asserted, then $\overline{\text{MCP}}$ is ignored and neither a machine check exception nor checkstop occur). When a processor is in checkstop state, instruction processing is suspended and generally cannot continue without restarting the processor. Note that many conditions may lead to the checkstop condition; the disabled machine check exception is only one of these. Note that the L3 cache is not supported on the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448.

A machine check exception may result from referencing a nonexistent physical address, either directly (with MSR[DR] = 0) or through an invalid translation. If a **dcbz** instruction introduces a block into the cache associated with a nonexistent physical address, a machine check exception can be delayed until an attempt is made to store that block to main memory. Not all processors that implement the PowerPC architecture provide the same level of error checking. Checkstop sources are implementation-dependent.

Machine check exceptions are enabled when MSR[ME] = 1; this is described in Section 4.6.2.1, “Machine Check Exception Enabled (MSR[ME] = 1).” If MSR[ME] = 0 and a machine check occurs, the processor enters the checkstop state. The checkstop state is described in Section 4.6.2.2, “Checkstop State (MSR[ME] = 0).”

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

Machine check exceptions are enabled when MSR[ME] = 1. When a machine check condition occurs, the MPC7450 waits for the processor to quiesce (defined in the “Glossary of Terms and Abbreviations”) and the memory subsystem to empty all its queues and terminate all pending data tenures. Then the vector touch engine (VTE) stops all streams when a machine check is detected. Once the processor and the memory subsystem have quiesced, a machine check exception is taken. When a machine check exception is taken, registers are updated as shown in Table 4-9.

Table 4-9. Machine Check Exception—Register Settings

Register	Setting Description
SRR0	On a best-effort basis the MPC7450 sets this to an EA of some instruction that was executing or about to be executing when the machine check condition occurred.
SRR1	0 Cleared 1 L1 instruction cache error 2 L1 data cache error 3–5 Normally cleared, used in debug. 6 Loaded with equivalent MSR bit 7–9 Cleared 10 Normally cleared, used in debug. 11 MSS error. Set for an L2 cache tag parity or L2 data parity error. Also set for an L3 SRAM or L3 tag parity error; otherwise zero. Refer to Section 2.2.5.4, “Memory Subsystem Status Register (MSSSR0),” for more information. Note that the L3 cache is not supported on the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448. 12 MCP. Set when $\overline{\text{MCP}}$ signal is asserted; otherwise 0 13 TEA. Set when $\overline{\text{TEA}}$ signal is asserted; otherwise 0 14 DP. Set when a data bus parity error is detected on MPXbus; otherwise 0 15 AP. Set when a address bus parity error is detected on MPXbus; otherwise 0 16–29 Loaded with equivalent MSR bits 30 Set in case of a recoverable exception 31 Loaded with equivalent MSR bits

Table 4-9. Machine Check Exception—Register Settings (continued)

L2ERRDET (for the MPC7448)	0 Multiple L2 errors (Bit reset, write-1-to-clear) 1–26 Reserved 27 Tag parity error (Bit reset, write-1-to-clear) 28 Multiple-bit ECC error (Bit reset, write-1-to-clear) 29 Single-bit ECC error (Bit reset, write-1-to-clear) 30–31 Reserved			
MSSSR0	0–12 Normally cleared, used in debug, writing nonzero values may cause boundedly undefined results 13 L2 tag parity error 14 L2 data parity error 15 L3 tag parity error 16 L3 data parity error 17 Address bus parity error 18 Data bus parity error 19 Bus transfer error acknowledge 20–31 Reserved			
	For the MPC7448: 0–12 Normally cleared, used in debug, writing nonzero values may cause boundedly undefined results 13 L2 tag parity error 14 L2 data parity error 15 Normally cleared, used in debug, writing nonzero values may cause boundedly undefined results 16 Normally cleared, used in debug, writing nonzero values may cause boundedly undefined results. 17 Address bus parity error 18 Data bus parity error 19 Bus transfer error acknowledge 20–31 Reserved			
MSR	VEC 0 POW 0 ILE — EE 0 PR 0	FP 0 ME 0 FE0 0 SE 0	BE 0 FE1 0 IP — IR 0	DR 0 PM 0 RI 0 LE ILE

Key: 0 Bit is cleared
ILE Bit is copied from the MSR[ILE]
— Bit is not altered

Note that to handle another machine check exception, the exception handler should set MSR[ME] as soon as it is practical after a machine check exception is taken. Otherwise, subsequent machine check exceptions cause the processor to enter the checkstop state.

When the MPC7450 takes the machine check exception, it sets one or more error bits in SRR1. The MPC7450 has multiple data parity error sources that can cause a machine check exception. The MSS error indicates one of many possible L2 or L3 parity errors as described more completely in Section 2.2.5.4, “Memory Subsystem Status Register (MSSSR0).” Memory subsystem errors in an ICTRL field need to be enabled to cause an error, see Section 2.2.5.5, “Instruction and Data Cache Registers,” for details. The SRR1[MCP] bit (SRR1[12]) indicates that the machine check signal was asserted. The TEA bit (SRR1[13]) indicates that the machine check was caused by a $\overline{\text{TEA}}$ assertion on the system bus. Note that the L3 cache is not supported on the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448.

The machine check exception is usually unrecoverable in the sense that execution cannot resume in the context that existed before the exception. If the condition that caused the machine check does not otherwise prevent continued execution, MSR[ME] is set by software to allow the processor to continue execution at the machine check exception vector address. Typically, earlier processes cannot resume; however, operating systems can use the machine check exception handler to try to identify and log the cause of the machine check condition.

When a machine check exception is taken, instruction fetching resumes at offset 0x00200 from the physical base address indicated by MSR[IP].

4.6.2.2 Checkstop State (MSR[ME] = 0)

If MSR[ME] = 0 and a machine check condition occurs, the processor enters the checkstop state.

When a processor is in checkstop state, instruction processing is suspended and generally cannot resume without the processor being reset. The contents of all latches are frozen within six cycles upon entering checkstop state.

Note that the MPC7450 has a $\overline{\text{CKSTP\_OUT}}$ signal (open-drain) that is asserted when the MPC7450 enters the checkstop state. Also, external logic can cause the MPC7450 to enter the checkstop state by asserting CKSTP_IN. See Section 8.4.3.5, "Checkstop Input (CKSTP_IN)—Input," and Section 8.4.3.6, "Checkstop Output (CKSTP_OUT)—Output," for more information on these checkstop signals.

4.6.3 DSI Exception (0x00300)

A DSI exception occurs when no higher priority exception exists and an error condition related to a data memory access occurs. The DSI exception is implemented as it is defined in the PowerPC architecture (OEA). For details on the DSI exception, see "DSI Exception (0x00300)," in the *Programming Environments Manual*. For example, a **lwarx** or **stwcx.** instruction that addresses memory to be mapped with the write-through (W = 1) or caching-inhibited (I = 1) attribute causes a DSI exception.

4.6.3.1 DSI Exception—Page Fault

When hardware table searching is enabled, HID0[STEN] = 0, and there is a TLB miss for a load, store, or cache operation, a DSI exception is taken if the resulting hardware table search causes a page fault. When software table searching is enabled, HID0[STEN] = 1, the TLB miss handlers configure SRR1 and DSISR appropriately for a page fault in this case and branch to the DSI exception handlers as described in Section 5.5.5.2, "Example Software Table Search Operation."

The condition that caused the exception is defined in the DSISR. These conditions also use the data address register (DAR) as shown in Table 4-10.

Table 4-10. DSI Exception—Register Settings

Register	Setting Description
DSISR	0 Cleared 1 Set by the hardware (if HID0[STEN]=0) or the DTLB miss exception handler if the translation of an attempted access is not found in the primary page table entry group (PTEG), or in the rehashed secondary PTEG, 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 write-through (W =1) or caching-inhibited (I = 1) memory. 6 Set for a store operation and cleared for a load operation. 7–8 Cleared 9 Set if DABR match occurs, otherwise cleared. 10 Cleared 11 Set if **eciwx** or **ecowx** instruction is executed when EAR[E] = 0; otherwise cleared. 12–31 Cleared
DAR	Set to the effective address of a memory element that caused the DSI, as described in the *Programming Environments Manual.*

4.6.3.2 DSI Exception—Data Address Breakpoint Facility

The MPC7450 also implements the data address breakpoint facility, which is defined as optional in the PowerPC architecture and is supported by the optional data address breakpoint register (DABR) and the DSI exception. Although the architecture does not strictly prescribe how this facility must be implemented, the MPC7450 follows the recommendations provided by the architecture and described in Chapter 2, “Programming Model,” and Chapter 6 “Exceptions,” in the *Programming Environments Manual*. The granularity of the data address breakpoint compare is a double word for all accesses except AltiVec quad-word loads and stores. For AltiVec accesses, the least significant bit of the DAB field (DABR[28]) is ignored, thus providing quad-word granularity. For these quad-word DAB matches, the DAR register is loaded with a quad-word-aligned address.

When a DSI exception is taken, instruction fetching resumes at offset 0x00300 from the physical base address indicated by MSR[IP].

4.6.4 ISI Exception (0x00400)

An ISI exception occurs when no higher priority exception exists and an attempt to fetch the next instruction fails. This exception is implemented as it is defined by the PowerPC architecture (OEA), and is taken for the following conditions:

- The effective address cannot be translated.
- The fetch access is to a no-execute segment (SR[N] = 1).
- The fetch access is to guarded storage and MSR[IR] = 1.
- The fetch access violates memory protection.

When an ISI exception is taken, instruction fetching resumes at offset 0x00400 from the physical base address indicated by MSR[IP].

4.6.5 External Interrupt Exception (0x00500)

An external interrupt is signaled to the processor by the assertion of the external interrupt signal ($\overline{\text{INT}}$) when MSR[EE] = 1. The $\overline{\text{INT}}$ signal is expected to remain asserted until the MPC7450 takes the external interrupt exception. If $\overline{\text{INT}}$ is negated early, recognition of the interrupt request is not guaranteed. After the MPC7450 begins execution of the external interrupt handler, the system can safely negate $\overline{\text{INT}}$. When the MPC7450 detects assertion of $\overline{\text{INT}}$, it stops dispatching and waits for all pending instructions to complete, including string and multiple instructions. This allows any instructions in progress that need to take an exception to do so before the external interrupt is taken. After all instructions have vacated the completion buffer, the MPC7450 takes the external interrupt exception as defined in the PowerPC architecture (OEA).

The MPC7450 also allows supervisor software to cause an external interrupt exception through the ICTRL[CIRQ] bit. When ICTRL[CIRQ] is set (and MSR[EE] = 1), the MPC7450 functions as if $\overline{\text{INT}}$ has been asserted, and it stop dispatching and waits for all pending instructions to complete. After all instructions have vacated the completion buffer, the MPC7450 takes the external interrupt exception. Note that if both ICTRL[CIRQ] is set and $\overline{\text{INT}}$ is asserted, only one interrupt is taken. Refer to Section 2.2.5.5.21, “Instruction Cache and Interrupt Control Register (ICTRL),” for more information on the setting and clearing of the ICTRL[CIRQ] bit.

An external interrupt may be delayed by other higher priority exceptions or if MSR[EE] is cleared when the exception occurs.

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

Table 4-11 lists register settings when an external interrupt exception is taken.

Table 4-11. External Interrupt Exception—Register Settings

Register	Setting Description			
SRR0	Set to the effective address of the instruction that the processor would have attempted to execute next if no exception conditions were present.			
SRR1	0 Cleared 1 Set when an external interrupt exception is caused by the ICTRL[CIRQ] bit 2–5 Cleared 6 Loaded with equivalent MSR bits 7–9 Cleared 10 Set when an external interrupt exception is caused by $\overline{\text{INT}}$ assertion 11–15 Cleared 16–31 Loaded with equivalent MSR bits			
MSR	VEC 0 POW 0 ILE — EE 0 LE ILE	PR 0 FP 0 ME — FE0 0	SE 0 BE 0 FE1 0 IP —	IR 0 DR 0 PM 0 RI 0

Key: 0 Bit is cleared
ILE Bit is copied from the MSR[ILE]
— Bit is not altered

4.6.6 Alignment Exception (0x00600)

The MPC7450 implements the alignment exception as defined by the PowerPC architecture (OEA). An alignment exception is initiated when any of the following occurs:

- The operand of a floating-point load or store is not word-aligned.
- The operand of **lmw**, **stmw**, **lwarx**, or **stwcx.** is not word-aligned.
- The operand of **dcbz** is in a page that is write-through or cache-inhibited.
- An attempt is made to execute **dcbz** when the data cache is disabled or locked.
- An **eciwx** or **ecowx** is not word-aligned
- A multiple or string access is attempted with MSR[LE] set
- In 60x bus mode, an access caused by **stvx**, **stvxl**, **lvx**, or **lvxl** instruction to a cache-inhibited page, write-through page, disabled L1 cache, or if all ways of the cache are locked.
- In 60x bus mode, an access caused by cache-inhibited AltiVec loads, stores, and write-through stores. The 60x bus mode does not support 16-byte bus transactions. Note this requires a re-write of the alignment exception routines in software that supports AltiVec quad-word access in 60x bus mode on the MPC7450.

Note that the MPC7450 does not take an alignment exception for load/store string accesses that cross a protection boundary or for a load/store multiplex access that crosses a segment or BAT boundary.

When an alignment exception is taken, instruction fetching resumes at offset 0x00600 from the physical base address indicated by MSR[IP].

The register settings for alignment exceptions are shown in Table 4-12.

Table 4-12. Alignment Interrupt—Register Settings

Register	Setting
DSISR	0—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) for instructions that use the update form. For **lmw**, **lswi**, and **lswx** instructions, 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. Otherwise undefined.
DAR	Set to the EA of the data access as computed by the instruction causing the alignment exception.

4.6.7 Program Exception (0x00700)

The MPC7450 implements the program exception as it is defined by the PowerPC architecture (OEA). A program exception occurs when no higher priority exception exists and one or more of the exception conditions defined in the OEA occur.

The MPC7450 invokes the system illegal instruction program exception when it detects any instruction from the illegal instruction class. The MPC7450 fully decodes the SPR field of the instruction. If an undefined SPR is specified, a program exception is taken.

The UISA defines **mtspr** and **mfspr** with the record bit (Rc) set as causing a program exception or giving a boundedly undefined result. In the MPC7450, the appropriate condition register (CR) should be treated as undefined. Likewise, the PowerPC architecture states that the Floating Compared Unordered (**fcmpu**) or Floating Compared Ordered (**fcmpo**) instructions with the record bit set can either cause a program exception or provide a boundedly undefined result. In the MPC7450, the BF field in an instruction encoding for these cases is considered undefined.

The MPC7450 does not support either of the two floating-point imprecise modes supported by the PowerPC architecture. Unless exceptions are disabled (MSR[FE0] = MSR[FE1] = 0), all floating-point exceptions are treated as precise.

When a program exception is taken, instruction fetching resumes at offset 0x00700 from the physical base address indicated by MSR[IP]. Chapter 6, “Exceptions,” in the *Programming Environments Manual* describes register settings for this exception.

4.6.8 Floating-Point Unavailable Exception (0x00800)

The floating-point unavailable exception is implemented as defined in the PowerPC architecture. A floating-point unavailable exception occurs when no higher priority exception exists, an attempt is made to execute a floating-point instruction (including floating-point load, store, or move instructions), and the floating-point available bit in the MSR is disabled, (MSR[FP] = 0). Register settings for this exception are described in Chapter 6, “Exceptions,” in the *Programming Environments Manual.*

When a floating-point unavailable exception is taken, instruction fetching resumes at offset 0x00800 from the physical base address indicated by MSR[IP].

4.6.9 Decrementer Exception (0x00900)

The decrementer exception is implemented in the MPC7450 as it is defined by the PowerPC architecture. The decrementer exception occurs when no higher priority exception exists, a decrementer exception condition occurs (for example, the decrementer register has completed decrementing), and MSR[EE] = 1. In the MPC7450, the decrementer register is decremented at one fourth the bus clock rate. Register settings for this exception are described in Chapter 6, “Exceptions,” in the *Programming Environments Manual.*

When a decrementer exception is taken, instruction fetching resumes at offset 0x00900 from the physical base address indicated by MSR[IP].

4.6.10 System Call Exception (0x00C00)

A system call exception occurs when a System Call (**sc**) instruction is executed. In the MPC7450, the system call exception is implemented as it is defined in the PowerPC architecture. Register settings for this exception are described in Chapter 6, "Exceptions," in the *Programming Environments Manual.*

When a system call exception is taken, instruction fetching resumes at offset 0x00C00 from the physical base address indicated by MSR[IP].

4.6.11 Trace Exception (0x00D00)

The trace exception is taken if MSR[SE] = 1 or if MSR[BE] = 1 and the currently completing instruction is a branch. Each instruction considered during trace mode completes before a trace exception is taken. When a **mtmsr** instruction is executed and the MSR[SE] transitions from 0 to 1, following the completion of that **mtmsr**, a trace exception is taken.

When a trace exception is taken, instruction fetching resumes at offset 0x00D00 from the base address indicated by MSR[IP].

4.6.12 Floating-Point Assist Exception (0x00E00)

The optional floating-point assist exception defined by the PowerPC architecture is not implemented in the MPC7450.

4.6.13 Performance Monitor Exception (0x00F00)

The MPC7450 microprocessor provides a performance monitor facility to monitor and count predefined events such as processor clocks, misses in either the instruction cache or the data cache, instructions dispatched to a particular execution unit, mispredicted branches, and other occurrences. An overflow of the counter in such events can be used to trigger the performance monitor exception. The performance monitor facility is not defined by the PowerPC architecture.

The performance monitor provides the ability to generate a performance monitor exception triggered by an enabled condition or event. This exception is triggered by an enabled condition or event defined as follows:

- A PMC*x* register overflow condition occurs
  - MMCR0[PMC1CE] and PMC1[OV] are both set
  - MMCR0[PMC*n*CE] and PMC*n*[OV] are both set ($n > 1$)
- A time base event—MMCR0[TBEE] = 1 and the TBL bit specified in MMCR0[TBSEL] changes from 0 to 1

MMCR0[PMXE] must be set for any of these conditions to signal a performance monitor exception.

Although the performance monitor exception may occur with MSR[EE] = 0, the exception is not taken until MSR[EE] = 1.

As a result of a performance monitor exception being generated, the performance monitor saves in the SIAR the effective address of the last instruction completed before the exception is generated. Note that SIAR is not updated if performance monitor counting has been disabled by setting MMCR0[0].

The performance monitor can be used for the following:

- To increase system performance with efficient software, especially in a multiprocessing system. Memory hierarchy behavior must be monitored and studied to develop algorithms that schedule tasks (and perhaps partition them) and that structure and distribute data optimally.
- To help system developers bring up and debug their systems.

The performance monitor uses the following SPRs:

- The performance monitor counter registers (PMC1–PMC6) are used to record the number of times a certain event has occurred. UPMC1–UPMC6 provide user-level read access to these registers.
- The monitor mode control registers (MMCR0–MMCR2) are used to enable various performance monitor exception functions. UMMCR0–UMMCR2 provide user-level read access to these registers.
- The sampled instruction address register (SIAR) contains the effective address of an instruction executing at or around the time that the processor signals the performance monitor exception condition. The USIAR register provides user-level read access to the SIAR.

Table 4-13 lists register settings when a performance monitor exception is taken.

Table 4-13. Performance Monitor Exception—Register Settings

Register	Setting Description			
SRR0	Set to the effective address of the instruction that the processor would have attempted to execute next if no exception conditions were present.			
SRR1	0–5 Cleared 6 Loaded with equivalent MSR bit 7–15 Cleared 16–31 Loaded with equivalent MSR bits			
MSR	VEC 0 POW 0 ILE — EE 0 LE ILE	PR 0 FP 0 ME — FE0 0	SE 0 BE 0 FE1 0 IP —	IR 0 DR 0 PM 0 RI 0

Key: 0 Bit is cleared
ILE Bit is copied from the MSR[ILE]
— Bit is not altered

As with other exceptions, the performance monitor exception follows the normal PowerPC exception model with a defined exception vector offset (0x00F00). The priority of the performance monitor exception lies between the external exception and the decrementer exception (see Table 4-3). The contents of the SIAR are described in Section 2.2.5.9, “Performance Monitor Registers.” The performance monitor is described in Chapter 11, “Performance Monitor.”

4.6.14 AltiVec Unavailable Exception (0x00F20)

The AltiVec facility includes another instruction-caused, precise exception in addition to the exceptions defined by the PowerPC architecture (OEA). An AltiVec unavailable exception occurs when no higher priority exception exists (see Table 4-3), and an attempt is made to execute an AltiVec instruction that accesses the vector register (VR) or the vector status and control register (VSCR) when MSR[VEC] = 0.

Note that the data streaming instructions, **dss**, **dst**, and **dstst** do not cause an AltiVec unavailable exception: the VR and VSCR registers are available to the data streaming instructions even when MSR[VEC] = 0.

4.6.15 TLB Miss Exceptions

When software table searching is enabled (HID0[STEN] = 1), and the effective address for a fetch can not be translated by the BATs or on-chip TLBs, one of three TLB miss exceptions is generated:

- ITLB miss exception
- DTLB miss-on-load
- DTLB miss-on-store

When the exception occurs, the effective address of the access that requires the software table search is saved in the TLBMISS register. Also, when the exception occurs, the fields of the PTEHI register are loaded automatically with the corresponding SR[VSID] information and the API of the missed page address. These registers are set to facilitate the searching of the page tables in software and their settings are shown in this section.

As described in the example code (Section 5.5.5.2.2, “Code for Example Exception Handlers”), if a TLB miss exception handler fails to find the desired PTE, then a page fault must be synthesized.

An example code sequence for a software table search operation (including a handler for these exceptions) is provided in Section 5.5.5.2, “Example Software Table Search Operation.”

Table 4-14 details the register settings when one of the TLB miss exceptions occurs.

Table 4-14. TLB Miss Exceptions—Register Settings

Register	Setting Description
TLBMISS	0–30 Effective page address for the access that caused the TLB miss exception 31 LRU Way
PTEHI	0 Set to1 1–24 The virtual segment ID (VSID) of the missed page address, SR[VSID] is copied to this field. 25 Set to 0 26–31The effective address’s abbreviated page index (EA[API]).
SRR0	Set to the effective address of the instruction that the processor would have attempted to execute next if no exception conditions were present.

Table 4-14. TLB Miss Exceptions—Register Settings (continued)

SRR1	0–5 Cleared 6 Loaded with equivalent MSR bits 7–11 Cleared except when DTLB miss on store exception occurs with C = 0, then SRR1[11] = 1. Refer to Section 4.6.15.3, “Data Table Miss-On-Store Exception—DTLB Miss-On-Store (0x01200),” for details. 12 Key for TLB Miss When the access is a user access (MSR[PR] = 0), this bit is set equal to SR[Ks]. When access is a supervisor access (MSR[PR] = 1), this bit is set equal to SR[Kp]. 13–15 Cleared 16–31 Loaded with equivalent MSR bits			
MSR	VEC 0 POW 0 ILE — EE 0 LE ILE	PR 0 FP 0 ME — FE0 0	SE 0 BE 0 FE1 0 IP —	IR 0 DR 0 PM 0 RI 0

Key:

0	Bit is cleared
ILE	Bit is copied from the MSR[ILE]
—	Bit is not altered

4.6.15.1 Instruction Table Miss Exception—ITLB Miss (0x01000)

When software table searching is enabled (HID0[STEN] = 1), and the effective address for an instruction fetch cannot be translated by the IBATs or ITLB, an ITLB miss exception is generated. Table 4-14 details the register settings for TLBMISS and PTEHI when an ITLB miss exception occurs

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].

4.6.15.2 Data Table Miss-On-Load Exception—DTLB Miss-On-Load (0x01100)

When software table searching is enabled (HID0[STEN] = 1), and the effective address for a load or cache load operation cannot be translated by the DBATs or DTLB, a DTLB miss on load exception is generated. If a TLB miss occurs in the middle of a load string or multiple access, the MPC7450 takes the DTLB miss-on-load exception when it occurs; after the exception is handled, the instruction is restarted. Table 4-14 details the register settings for the TLBMISS and PTEHI when a DTLB miss-on-load exception occurs.

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

4.6.15.3 Data Table Miss-On-Store Exception—DTLB Miss-On-Store (0x01200)

When the effective address for a data store or cache store operation can not be translated by the DBAT or DTLB, a DTLB miss-on-store exception is generated. The data TLB miss-on-store exception is also taken when the changed bit for a matching DTLB entry needs to be updated in memory for a store operation (C = 0). If a TLB miss occurs in the middle of a store string or multiple access, the MPC7450 takes the DTLB miss-on-store exception.

Table 4-14 details the register settings for TLBMISS and PTEHI when a TLB miss exception occurs. Note that SRR1[11] is set when a DTLB hit occurs and the matching entry must have its changed bit in the PTE set due to a data store operation (PTE C bit = 0, and must be set to 1).

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].

4.6.16 Instruction Address Breakpoint Exception (0x01300)

An instruction address breakpoint exception occurs when all of the following conditions are met:

- The instruction breakpoint address IABR[0–29] matches EA[0–29] of the next instruction to complete in program order. The instruction that triggers the instruction address breakpoint exception is not executed before the exception handler is invoked.
- The IABR[TE] bit matches the MSR[IR] bit.
- The breakpoint enable bit (IABR[BE]) is set.

The instruction tagged with the match does not complete before the breakpoint exception is taken.

Table 4-15 lists register settings when an instruction address breakpoint exception is taken.

Table 4-15. Instruction Address Breakpoint Exception—Register Settings

Register	Setting Description			
SRR0	Set to the effective address of the instruction that the processor would have attempted to execute next if no exception conditions were present.			
SRR1	0–5 Cleared 6 Loaded with equivalent MSR bit 7–15 Cleared 16–31 Loaded with equivalent MSR bits			
MSR	VEC 0 POW 0 ILE — EE 0 LE Set to value of ILE	PR 0 FP 0 ME — FE0 0	SE 0 BE 0 FE1 0 IP —	IR 0 DR 0 PM 0 RI 0

Key:		
	0	Bit is cleared
	ILE	Bit is copied from the MSR[ILE]
	—	Bit is not altered

The MPC7450 requires that an **mtspr** to the IABR be followed by a context-synchronizing instruction. The MPC7450 cannot generate a breakpoint response for that context-synchronizing instruction if the breakpoint is enabled by the **mtspr**[IABR] immediately preceding it. The MPC7450 also cannot block a breakpoint response on the context-synchronizing instruction if the breakpoint was disabled by the **mtspr**[IABR] instruction immediately preceding it. The format of the IABR register is shown in Section 2.2.5.6, "Instruction Address Breakpoint Register (IABR)."

When an instruction address breakpoint exception is taken, instruction fetching resumes at offset 0x01300 from the base address indicated by MSR[IP].

4.6.17 System Management Interrupt Exception (0x01400)

The MPC7450 implements a system management interrupt, which is not defined by the PowerPC architecture. The system management interrupt is very similar to the external interrupt and it must be enabled with MSR[EE] = 1. It is particularly useful in implementing the nap mode. It has priority over an external interrupt (see Table 4-3) and uses a different vector in the exception table (offset 0x01400).

Table 4-16 lists register settings when a system management interrupt is taken.

Table 4-16. System Management Interrupt Exception—Register Settings

Register	Setting Description			
SRR0	Set to the effective address of the instruction that the processor would have attempted to execute next if no exception conditions were present.			
SRR1	0–5 Cleared 6 Loaded with equivalent MSR bit 7–15 Cleared 16–31 Loaded with equivalent MSR bits			
MSR	VEC 0 POW 0 ILE — EE 0 LE Set to value of ILE	PR 0 FP 0 ME — FE0 0	SE 0 BE 0 FE1 0 IP —	IR 0 DR 0 PM 0 RI 0

Key: 0 Bit is cleared
ILE Bit is copied from the MSR[ILE]
— Bit is not altered

Like the external interrupt, a system management interrupt is signaled to the MPC7450 by the assertion of an input signal. The system management interrupt signal ($\overline{\text{SMI}}$) is expected to remain asserted until the exception is taken. If $\overline{\text{SMI}}$ is negated early, recognition of the interrupt request is not guaranteed. After the MPC7450 begins execution of the system management interrupt handler, the system can safely negate $\overline{\text{SMI}}$. After the assertion of $\overline{\text{SMI}}$ is detected, the MPC7450 stops dispatching instructions and waits for all pending instructions to complete. This allows any instructions in progress that need to take an exception to do so before the system management interrupt exception is taken. Note that the MPC7450 waits for any load/store string or multiple instructions that have begun to be complete before taking the system management interrupt exception.

When a system management interrupt exception is taken, instruction fetching resumes as offset 0x01400 from the base address indicated by MSR[IP].

4.6.18 AltiVec Assist Exception (0x01600)

The MPC7450 implements an AltiVec assist exception to handle denormalized numbers in Java mode (VSCR[NJ] = 0). An AltiVec assist exception occurs when no higher priority exception exists and an instruction causes a trap condition as defined in Section 7.1.2.5, “Java Mode, NaNs, Denormalized Numbers, and Zeros.” Note that the MPC7450 handles most denormalized numbers in Java mode by taking a trap to the AltiVec assist exception, but for some instructions, the MPC7450 can produce the exact result without trapping.

Table 4-17 lists register settings when an AltiVec assist exception is taken.

Table 4-17. AltiVec Assist Exception—Register Settings

Register	Setting Description			
SRR0	Set to the effective address of the instruction that caused the exception.			
SRR1	0–5 Cleared 6 Loaded with equivalent MSR bit 7–15 Cleared 16–31 Loaded with equivalent MSR bits			
MSR	VEC 0 POW 0 ILE — EE 0 LE Set to value of ILE	PR 0 FP 0 ME — FE0 0	SE 0 BE 0 FE1 0 IP —	IR 0 DR 0 PM 0 RI 0

Key:	
0	Bit is cleared
ILE	Bit is copied from the MSR[ILE]
—	Bit is not altered

When an AltiVec assist exception is taken, instruction fetching resumes at offset 0x01600 from the base address indicated by MSR[IP].

Chapter 5
Memory Management

This chapter describes the MPC7450 microprocessor's implementation of the memory management unit (MMU) specifications provided by the operating environment architecture (OEA) for processors that implement the PowerPC architecture. The primary function of the MMU in a 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 and the implementation-specific changes in the MPC7450 MMU model to support 36-bit physical addressing. Refer to Chapter 7, "Memory Management," in the *Programming Environments Manual* for a complete description of the conceptual model used for 32-bit physical addressing. Note that the MPC7450 does not implement the optional direct-store facility.

Two general types of memory accesses generated by processors that implement the PowerPC architecture require address translation—instruction accesses and data accesses generated by load and store instructions. In addition, the addresses specified by cache instructions and the optional external control instructions also require translation. Generally, the address translation mechanism is defined in terms of the segment descriptors and page tables that the processors use to locate the effective-to-physical address mapping for memory accesses. The segment information translates the effective address (EA) to an interim virtual address, and the page table information translates the virtual address (VA) to a physical address (PA).

The segment descriptors, used to generate the interim virtual addresses, are stored as on-chip segment registers on 32-bit implementations (such as the MPC7450). In addition, two translation lookaside buffers (TLBs) are implemented on the MPC7450 to keep recently used page address translations on-chip. Although the PowerPC OEA describes one MMU (conceptually), the MPC7450 hardware maintains separate TLBs and table search resources for instruction and data accesses that can be performed independently (and simultaneously). Therefore, the MPC7450 is described as having two MMUs, one for instruction accesses (IMMU) and one for data accesses (DMMU).

The block address translation (BAT) mechanism is a software-controlled array that stores the available block address translations on-chip. BAT array entries are implemented as pairs of BAT registers that are accessible as supervisor special-purpose registers (SPRs). There are separate instruction and data BAT mechanisms. In the MPC7450, 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 4.3, "Exception Processing," describes the MSR that controls some of the critical functionality of the MMUs.

AltiVec Technology and the MMU Implementation

The AltiVec functionality in the MPC7450 affects the MMU model in the following ways:

- A data stream instruction (**dst**[**t**] or **dstst**[**t**]) can cause table search operations to occur after the instruction is retired.
- MMU exception conditions can cause a data stream operation to abort.
- Aborted VTQ-initiated table search operations can cause a line fetch skip.
- Execution of a **tlbsync** instruction can cancel an outstanding table search operation for a VTQ.

5.1 MMU Overview

The MPC7450 implements the memory management specification of the PowerPC OEA for 32-bit implementations but adds capability for supporting 36-bit physical addressing. 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 MPC7450 MMUs use an interim virtual address (52 bits) and hashed page tables in the generation of 32-bit or 36-bit physical addresses (depending on the setting of HID0[XAEN]). Processors that implement the PowerPC architecture also have a BAT mechanism for mapping large blocks of memory. For the MPC7441, MPC7450, and MPC7451, block sizes range from 128 Kbyte to 256 Mbyte and are software-programmable. For the MPC7445, MPC7447, MPC7447A, MPC7448, MPC7455, and MPC7457, block sizes range from 128 Kbytes to 4 Gbytes and are also software-programmable.

Basic features of the MPC7450 MMU implementation defined by the OEA are as follows:

- Support for real addressing mode—Effective-to-physical address translation can be disabled separately for data and instruction accesses.
- Block address translation—Each of the BAT array entries (four IBAT entries and four DBAT entries) provides a mechanism for translating blocks as large as:
  - 256 Mbytes for the MPC7441, MPC7450, and MPC7451
  - 4 Gbytes for the MPC7445, MPC7447, MPC7447A, MPC7448, MPC7455, and MPC7457

  from the 32-bit effective address space into the physical memory space. This can be used for translating large address ranges whose mappings do not change frequently. Four additional IBAT and DBAT entries are provided for the MPC7445, MPC7447, MPC7447A, MPC7448, MPC7455, and MPC7457 that can be enabled by setting HID0[HIGH_BAT_EN], for a total of eight IBAT entries and eight DBAT entries.
- Segmented address translation—The 32-bit effective address is extended to a 52-bit virtual address by substituting 24 bits of upper address bits from the segment register. The 4 upper bits of the EA are used as an index into the segment register file. This 52-bit virtual address space is divided into 4-Kbyte pages, each of which can be mapped to a physical page.

The MPC7450 processor also provides the following features that are not required by the PowerPC architecture:

- Separate translation lookaside buffers (TLBs)—The 128-entry, two-way set-associative ITLBs and DTLBs keep recently used page address translations on-chip.
- Table search operations performed in hardware—The 52-bit virtual address is formed and the MMU attempts to fetch the PTE that contains the physical address from the appropriate TLB on-chip. If the translation is not found in either the BAT array or in a TLB (that is, a TLB miss occurs), the hardware performs a table search operation (using a hashing function) to search for the PTE. Hardware table searching is the default mode for the MPC7450; however, if HID0[STEN] = 1, software table searching is performed.
- Table search operations performed in software—The MPC7450 also supports software table searching (when HID0[STEN] is set) for TLB misses. In this case, the TLBMISS register saves the effective address of the access that requires a software table search. The PTEHI and PTELO registers, as well as the **tlbli** and **tlbld** instructions are resources used in reloading the TLBs during a software table search operation. Also there are three exceptions used to support software table searching when HID0[STEN] = 1 and a TLB miss occurs. They are as follows:
  - For an instruction fetch, an ITLB miss exception,
  - For a data load, a DTLB miss-on-load exception,
  - For a data store, a DTLB miss-on-store exception.
- TLB invalidation—The MPC7450 implements the optional TLB invalidate entry (**tlbie**) and TLB synchronize (**tlbsync**) instructions that can be used to invalidate TLB entries. For more information on the **tlbie** and **tlbsync** instructions, see Section 5.4.4.2, “TLB Invalidation.”
- Extended 36-bit physical addresses provide for 64 Gbytes of physical memory when HID0[XAEN] is set.

Table 5-1 summarizes the MPC7450 MMU features, including those defined by the PowerPC architecture (OEA) for 32-bit processors and those specific to the MPC7450.

Table 5-1. MMU Features Summary

Feature Category	Architecturally Defined/ MPC7450-Specific	Feature
Address ranges	Architecturally defined	2^{32} bytes of effective address
		2^{52} bytes of virtual address
		2^{32} bytes of physical address
	MPC7450-specific	optional 2^{36} bytes of physical address
Page size	Architecturally defined	4 Kbytes
Segment size	Architecturally defined	256 Mbytes

Table 5-1. MMU Features Summary (continued)

Feature Category	Architecturally Defined/ MPC7450-Specific	Feature
Block address translation	Architecturally defined	Range of 128 Kbyte–256 Mbyte sizes for the MPC7441, MPC7450, and the MPC7451
		Four IBAT and four DBAT entries in the BAT array for the MPC7441, MPC7450, and the MPC7451
	MPC7445-, MPC7447-, MPC7447A-, MPC7448-, MPC7455-, and MPC7457-specific	Range of 128 Kbyte–4 Gbyte block sizes for the MPC7445, MPC7447, MPC7447A, MPC7448, MPC7455, and MPC7457
		Eight IBAT and eight DBAT entries in BAT array for the MPC7445, MPC7447, MPC7447A, MPC7448, MPC7455, and MPC7457
Memory protection	Architecturally defined	Segments selectable as no-execute
		Pages selectable as user/supervisor and read-only or guarded
		Blocks selectable as user/supervisor and read-only or guarded
Page history	Architecturally defined	Referenced and changed bits defined and maintained
Page address translation	Architecturally defined	Translations stored as PTEs in hashed page tables in memory
		Page table size determined by mask in SDR1 register
TLBs	Architecturally defined	Instructions for maintaining TLBs (**tlbie** and **tlbsync** instructions in MPC7450)
	MPC7450-specific	128-entry, two-way set associative ITLB 128-entry, two-way set associative DTLB LRU replacement algorithm
Segment descriptors	Architecturally defined	Stored as segment registers on-chip (two identical copies maintained)
Page table search support—Hardware	MPC7450-specific	The MPC7450 can perform the table search operation in hardware (or software, as listed below).
Page table search support—Software	MPC7450-specific	TLBMISS register (missed effective address) PTEHI and PTELO registers (contents of corresponding PTE)
		Three MMU exceptions, defined: ITLB miss exception, DTLB miss on load exception, and DTLB miss on store (or store and C = 0) exception; MMU-related bits are set in SRR1 for these exceptions
		tlbli rB instruction for loading ITLB entries **tlbld r**B instruction for loading DTLB entries

5.1.1 Memory Addressing

A program references memory using the effective (logical) address computed by the processor when it executes a load, store, branch, or cache instruction, and when it fetches the next instruction. The effective address is translated to a physical address according to the procedures described in Chapter 7, “Memory Management,” in the *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 2.4.2.3, “Effective Address Calculation.”

5.1.2 MMU Organization

Figure 5-1 shows the conceptual organization of a PowerPC MMU in a 32-bit implementation that generates 32-bit physical addresses. 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, hardware support for the automatic search of the page tables for PTEs, and other hardware features (invisible to the system software) not shown. Also, the MPC7450 generates a 36-bit physical address which is not represented by the 32-bit physical address in Figure 5-1.

The instruction addresses are generated by the processor for sequential instruction fetches and addresses that correspond to a change of program flow. Data addresses are generated by load, store, and cache instructions.

As shown in Figure 5-1, when the default 32-bit physical addresses are generated, the high-order bits of the effective address, EA[0–19] (or a smaller set of address bits, EA[0–*n*], in the cases of blocks), are translated into physical address bits PA[0–19]. The low-order address bits, EA[20–31], are untranslated and are therefore identical for both effective and physical addresses. After translating the address, the MMU passes the resulting 32-bit physical address to the memory subsystem.

The MMUs record whether the translation is for an instruction or data access, whether the processor is in user or supervisor mode and, for data accesses, whether the access is a load or a store operation. The MMUs use this information to appropriately direct the address translation and to enforce the protection hierarchy programmed by the operating system. Section 4.3, “Exception Processing,” describes the MSR that controls some of the critical functionality of the MMUs.

Figure 5-2 and Figure 5-3 contain the block diagrams of the IMMU and DMMU of the MPC7450 and show how a 36-bit physical address is generated. Address bits EA[20–26] index into the on-chip instruction and data caches to select a cache set. The remaining physical address bits are then compared with the tag fields (comprised of bits PA[0–23]) of the two selected cache blocks to determine if a cache hit has occurred. In the case of a cache miss on the MPC7450, the instruction or data access is then forwarded to the L2 cache tags to check for an L2 cache hit. In case of a miss, the access is forwarded to the L3 interface tags to check for an L3 cache hit. In the case of an L3 cache miss, the access is forwarded to the bus interface unit. Note that the L3 cache and L3 cache interface are not supported on the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448.

Figure 5-2 and Figure 5-3 also show the two on-chip TLBs maintained by the MPC7450 that have the following characteristics:

- 128 entries, two-way set associative (64 x 2), LRU replacement
- Hardware or software table search operations and TLB reloads
- Hardware or software update of referenced (R) and changed (C) bits in the translation table
- 36-bit physical addresses

In the event of a TLB miss, the TLB entry must be loaded. The TLB is loaded automatically by the hardware or by the software table search algorithm, depending on the HID0[STEN] setting.

Figure 5-2 and Figure 5-3 show the detailed routing of addresses that are generated by the IMMU and DMMU respectively when 36-bit addressing (extended addressing) is used. In this case, EA[0–19] (or a smaller subset EA[0–n], in the case of blocks) are translated into physical address bits PA[0–23] and the low-order address bits, EA[20–31] are untranslated, but shifted down to comprise PA[24–35]. Also, in this case, EA[20–26] index into the on-chip caches so that PA[0–23] from the MMU can be compared with the tag fields (comprised of PA[0–23]) to determine if a cache hit has occurred.

Figure 5-3 shows the detailed routing of addresses for the MPC7445, MPC7447, MPC7447A, MPC7448, MPC7455, and MPC7457 that are generated by the DMMU when 36-bit addressing (extended addressing) is used. Also the extended block size is enabled so that the EA[0–19] is translated into physical address bits PA[0–23] and the low-order address bits, EA[20–31], are untranslated but shifted down to comprise PA[24–35]. Also, in this case, additional BATs are available (DBAT4U to DBAT7L) for use. The same features, extended block size and additional BATs would be generated by the IMMU as well.

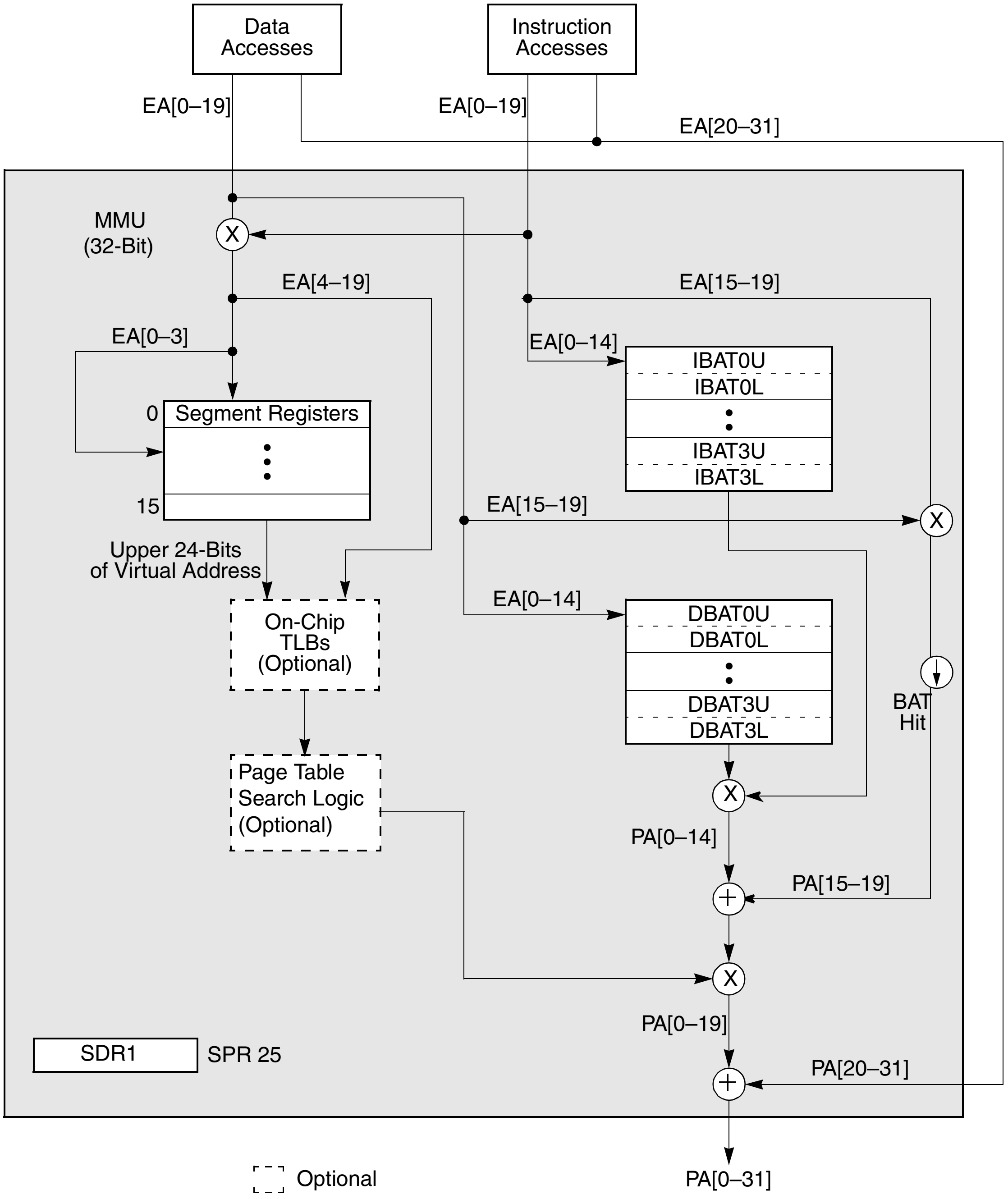


Figure 5-1. MMU Conceptual Block Diagram for a 32-Bit Physical Address (Not MPC7450)

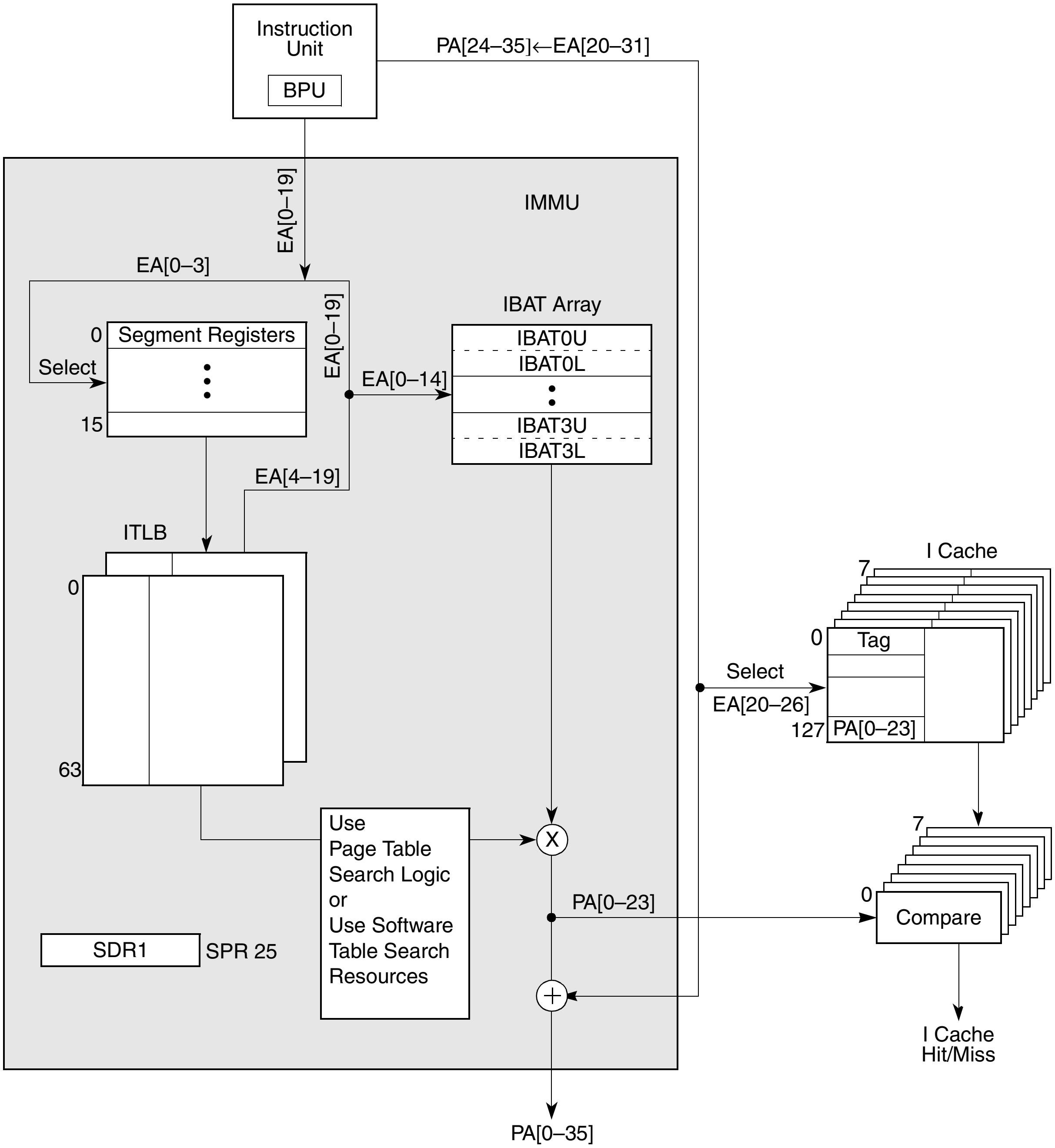


Figure 5-2. MPC7450 Microprocessor IMMU Block Diagram, 36-Bit Physical Addressing

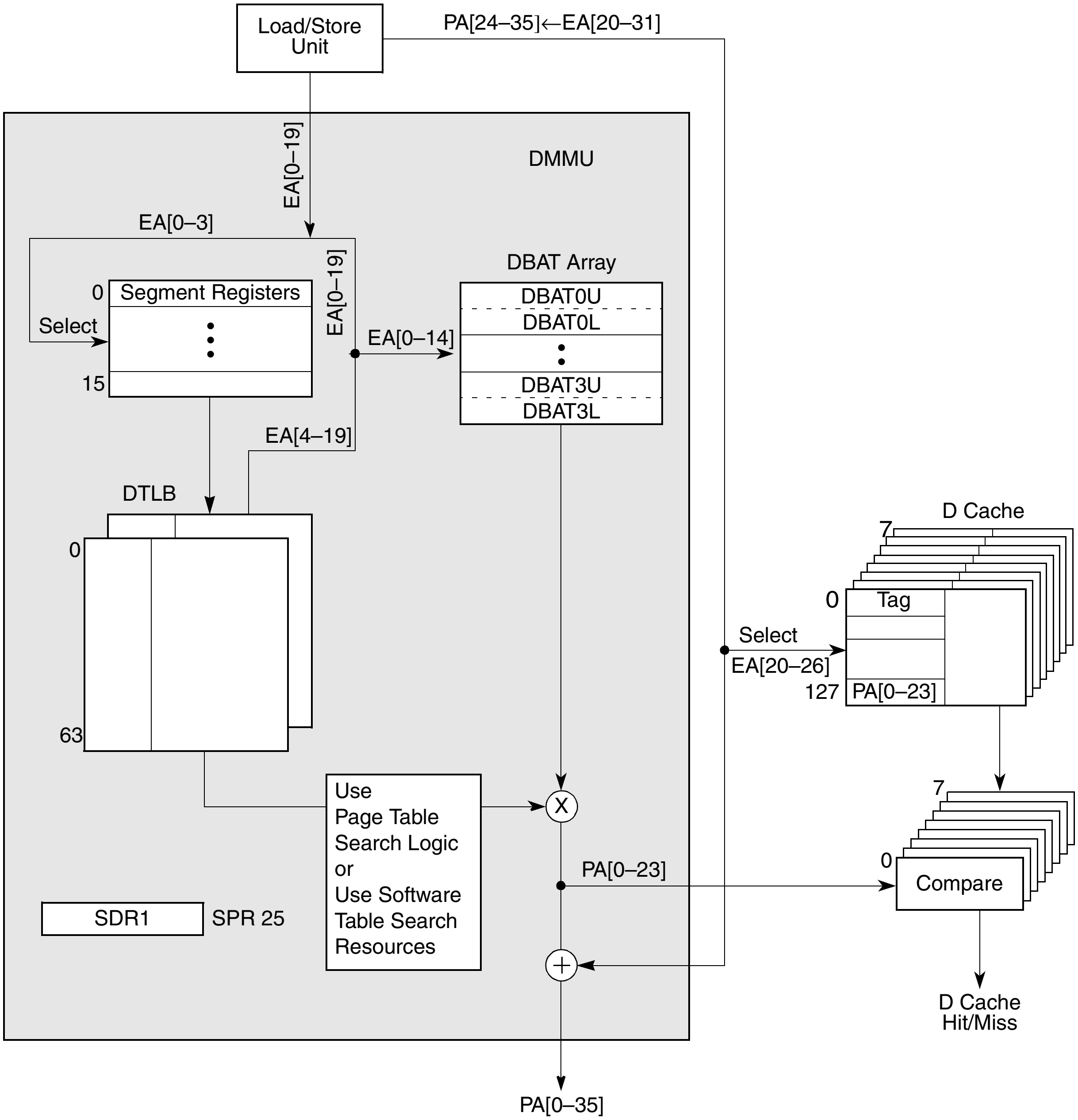


Figure 5-3. MPC7450 Microprocessor DMMU Block Diagram, 36-Bit Physical Addressing

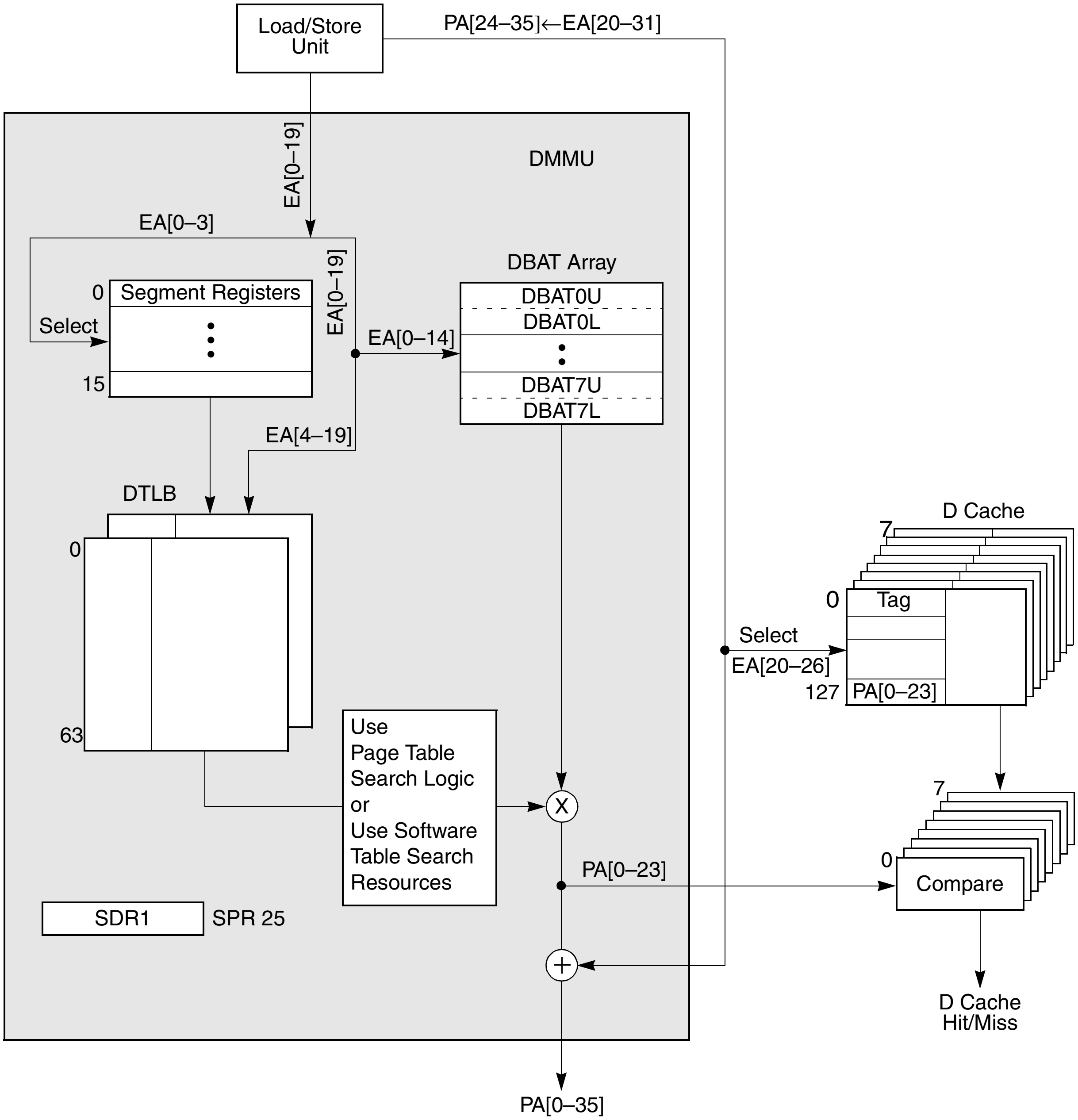


Figure 5-4. MPC7445, MPC7447, MPC7447A, MPC7448, MPC7455, and MPC7457 Microprocessor DMMU Block Diagram with Extended Block Size and Additional BATs

5.1.3 Address Translation Mechanisms

Processors that implement the PowerPC architecture support the following 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 Kbytes to 256 Mbytes (MPC7441, MPC7450, MPC7451) or 128 Kbytes to 4 GBytes (MPC7445, MPC7447, MPC7447A, MPC7448, MPC7455, and the MPC7457)
- Real addressing mode—Address translation is disabled; therefore, no translation is done and the physical address is identical to the effective address.

Figure 5-5 shows the three address translation mechanisms provided by the MMUs for 32-bit physical addressing and Figure 5-6 shows the same mechanism for 36-bit physical addressing. The segment descriptors shown in the figures control the page address translation mechanism. When an access uses page address translation, the appropriate segment descriptor is required. The appropriate segment descriptor is selected from the 16 on-chip segment registers 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 (SR*n*[T]). Note that the direct-store interface was present in the architecture only for compatibility with existing I/O devices that used this interface. The MPC7450 does not support the direct-store interface (SR*n*[T] = 1). When an access is determined to be to the direct-store interface space, the MPC7450 takes a DSI exception if it is a data access (see Section 4.6.3, “DSI Exception (0x00300)”), and takes an ISI exception if it is an instruction access (see Section 4.6.4, “ISI Exception (0x00400)”).

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 or 36-bit physical address used by the memory subsystem. In most cases, the physical address for the page resides in an on-chip TLB and is available for quick access. However, if the page address translation misses in the on-chip TLB, the MMU causes a search of the page tables in memory. Page tables can be searched by hardware using the virtual address information and a hashing function to locate the required physical address or the MPC7450 vectors to exception handlers that use software to search the page tables (if HID0[STEN] = 1).

Because blocks are larger than pages, there are fewer higher-order effective address bits to be translated into physical address bits (more low-order address bits (at least 17) are untranslated to form the offset into a block) for block address translation. Also, instead of segment descriptors and a TLB, block address translations use the on-chip BAT registers as a BAT array. If an effective address matches the corresponding field of a BAT register, the information in the BAT register is used to generate the physical address; in this case, the results of the page translation (occurring in parallel) are ignored.

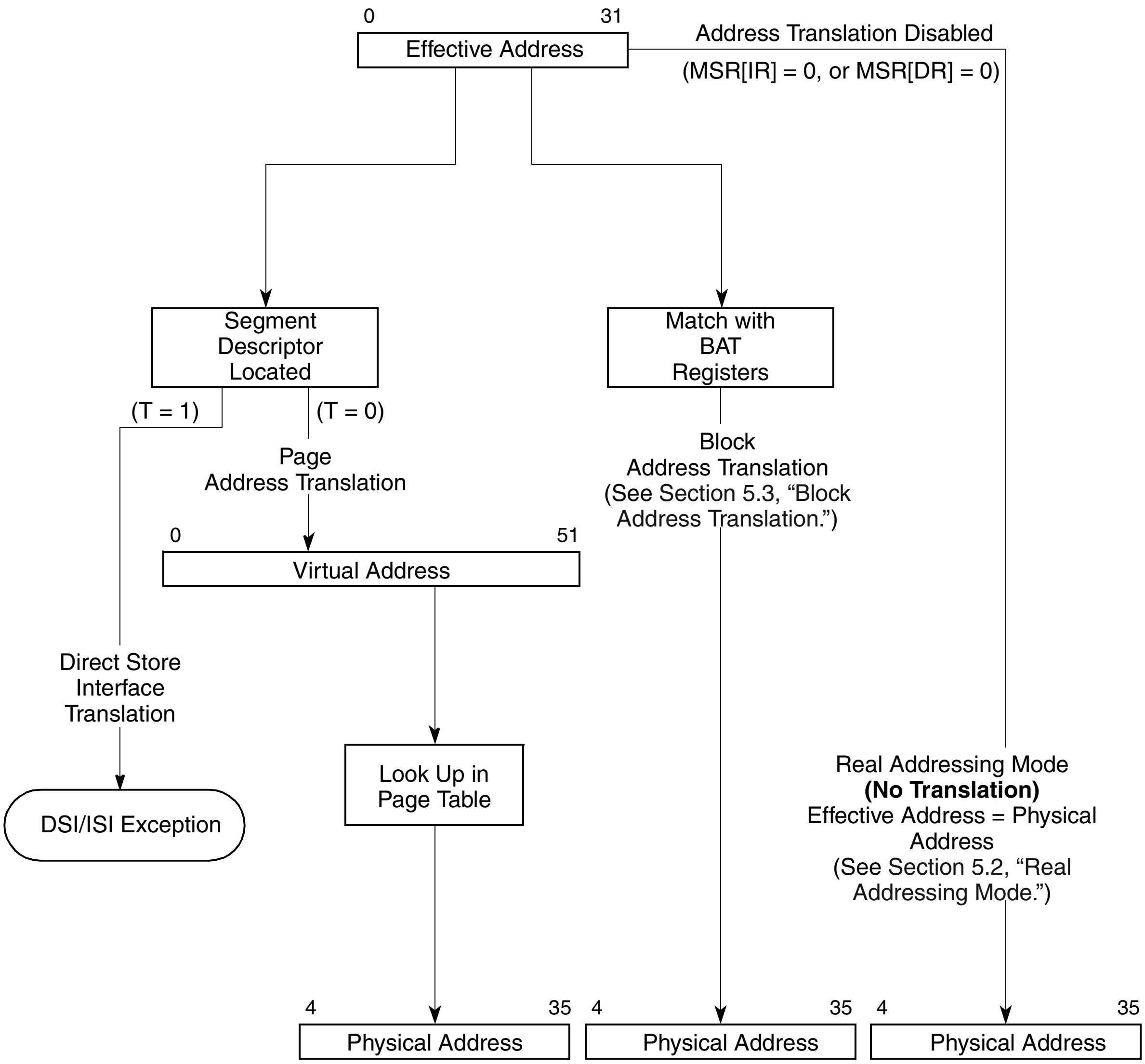


Figure 5-5. Address Translation Types for 32-Bit Physical Addressing

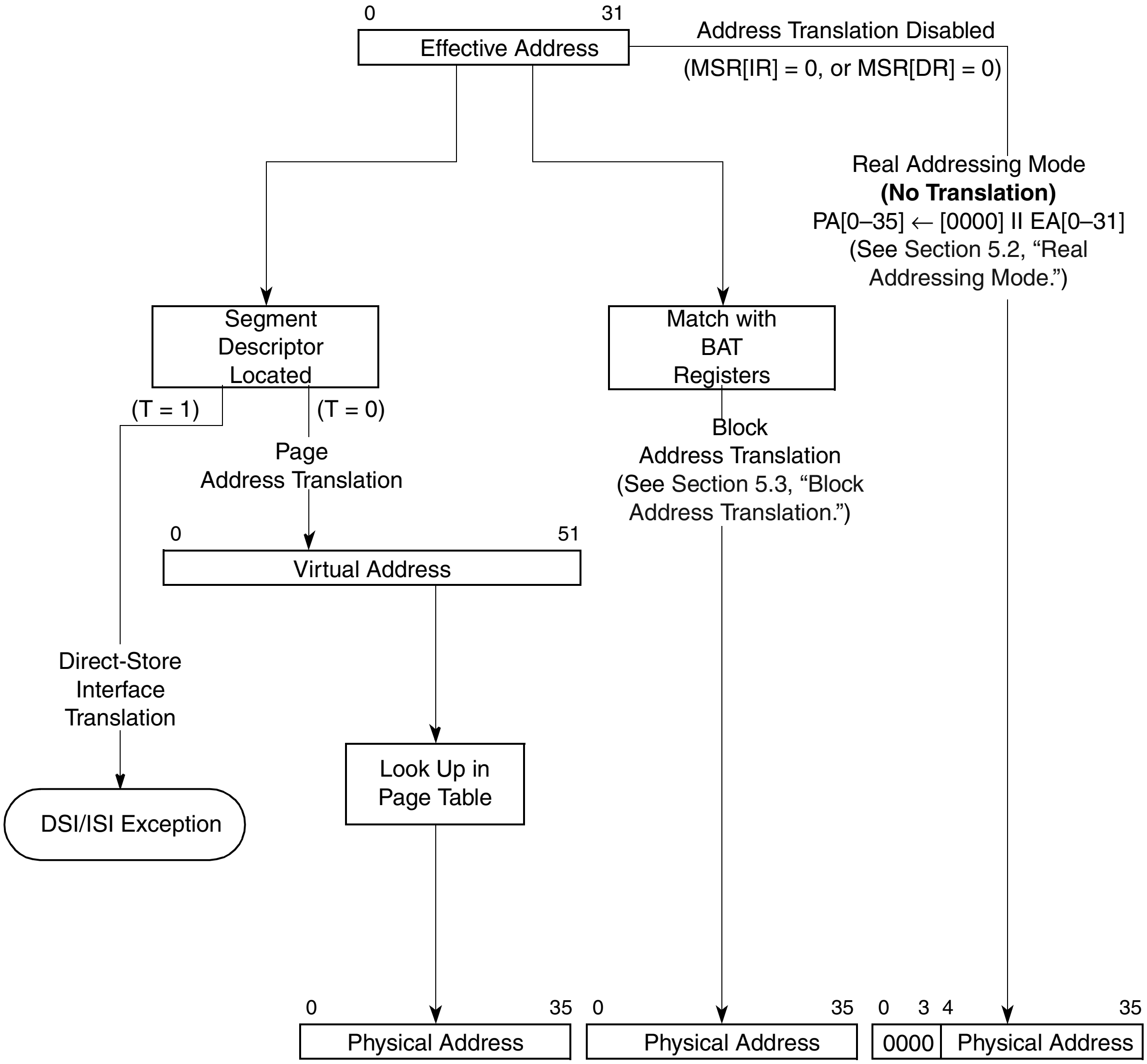


Figure 5-6. Address Translation Types for 36-Bit Physical Addressing

When the processor generates an access, and the corresponding address translation enable bit in MSR is cleared (MSR[IR] = 0 or MSR[DR] = 0), the resulting physical address is identical to the effective address and all other translation mechanisms are ignored. Instruction address translation and data address translation are enabled by setting MSR[IR] and MSR[DR], respectively.

When extended addressing is enabled, HID0[XAEN] = 1, and the corresponding address translation bit in MSR is cleared (MSR[IR] = 0 or MSR[DR] = 0), the 36-bit physical address is formed by concatenating 4 leading zeros to the 32-bit effective address.

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 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 (user-/supervisor) read-only	√	√	—	√	√	—
Both (user/supervisor) read-only-no-execute	—	√	—	—	√	—

√ Access permitted
— Protection violation

The no-execute option provided in the segment register lets the operating system program determine whether instructions can be fetched from an area of memory. The remaining options are enforced based on a combination of information in the segment descriptor and the page table entry. Thus the supervisor-only option allows only read and write operations generated while the processor is operating in supervisor mode (MSR[PR] = 0) to access the page. User accesses that map into a supervisor-only page cause an exception.

Finally, a facility in the VEA and OEA 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 used to control I/O devices can be marked as guarded so accesses do not occur unless they are explicitly required by the program.

For more information on memory protection, see the section, “Memory Protection Facilities” in Chapter 7, “Memory Management,” in the *Programming Environments Manual*.

5.1.5 Page History Information

The MMUs of processors that support the PowerPC architecture also define referenced (R) and changed (C) bits in the page address translation mechanism that can be used as history information relevant to the page. The operating system can use these bits to determine which areas of memory to write back to disk when new pages must be allocated in main memory. While these bits are initially programmed by the operating system into the page table, the architecture specifies that they can be maintained either by the processor hardware (automatically) or by some software-assist mechanism.

When loading the TLBs in hardware, the MPC7450 checks the state of the changed and referenced bits for the matched PTE. If the referenced bit is not set and the table search operation is initially caused by a load operation or by an instruction fetch, the MPC7450 automatically sets the referenced bit in the translation table. Similarly, if the table search operation is caused by a store operation and either the referenced bit or the changed bit is not set, the hardware automatically sets both bits in the translation table. In addition, when the address translation of a store operation hits in the DTLB, the MPC7450 checks the state of the changed bit. If the bit is not already set, the hardware automatically updates the DTLB and the translation table in memory to set the changed bit. For more information, see Section 5.4.2, “Page History Recording.”

When software table searching is enabled (HID0[STEN] = 1), the software table search routines used by the MPC7450 can set the R bit when a PTE is accessed. Also, the MPC7450 causes an exception (to vector to the software table search routines) when the C bit in the TLB is cleared but a store occurs, allowing the corresponding PTE to be updated by software.

5.1.6 General Flow of MMU Address Translation

The following sections describe the general flow used by processors that implement the PowerPC architecture, to translate effective addresses to physical addresses. There are three types of addressing translations used by the PowerPC architecture, page address, block address, and real addressing mode. Two sizes of physical addresses, 32-bit or 36-bit, can be generated depending on whether extended addressing is enabled (HID0[XAEN] = 1). Details for how an effective address is translated to a 32-bit physical address are described in Chapter 7, “Memory Management,” in the *Programming Environments Manual*. The following sections describe the differences in address translation for an extended physical address (36-bits).

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 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 5-7 shows the flow the MMUs use in determining which translation to select: real addressing mode, block address, or page address.

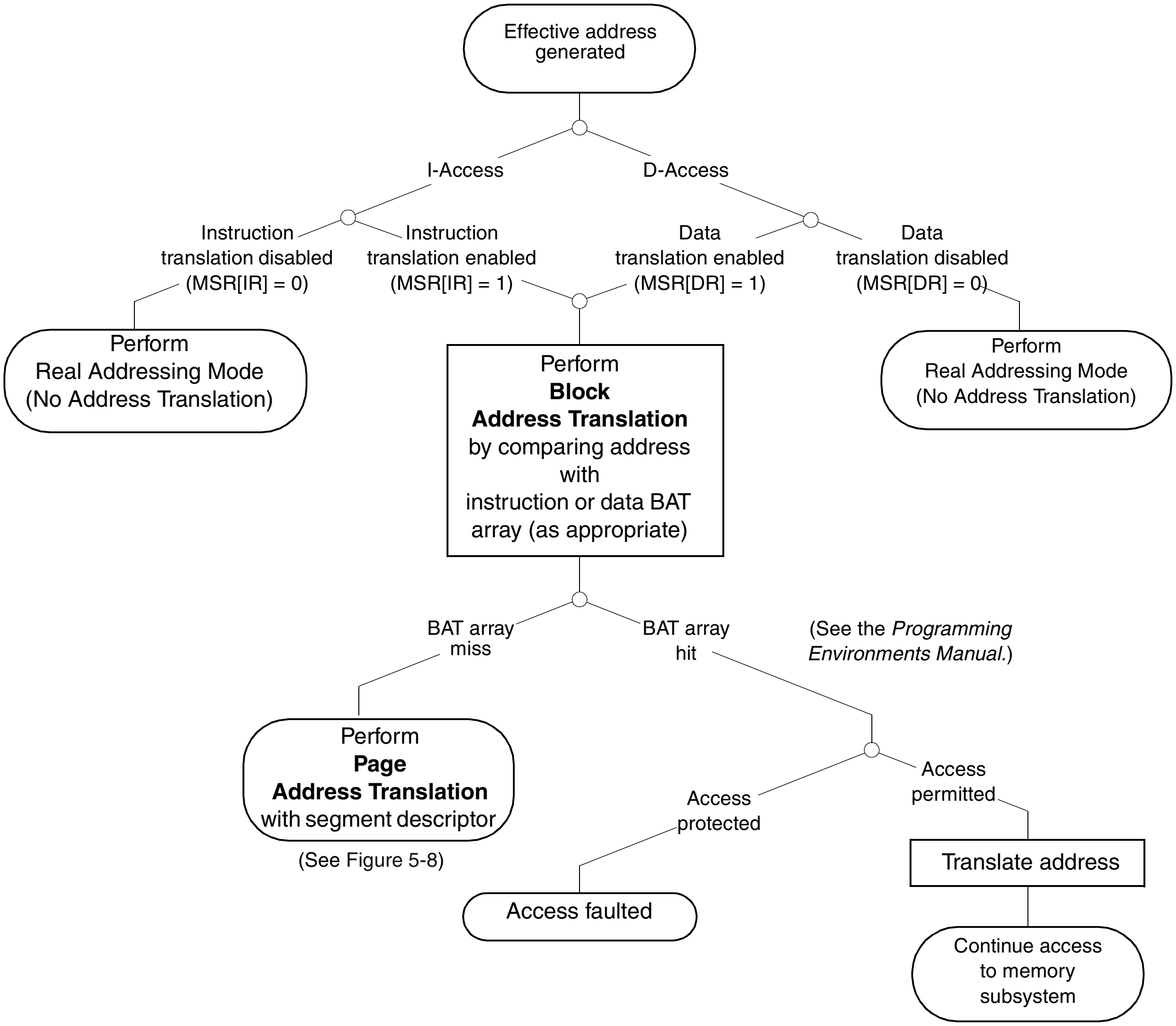


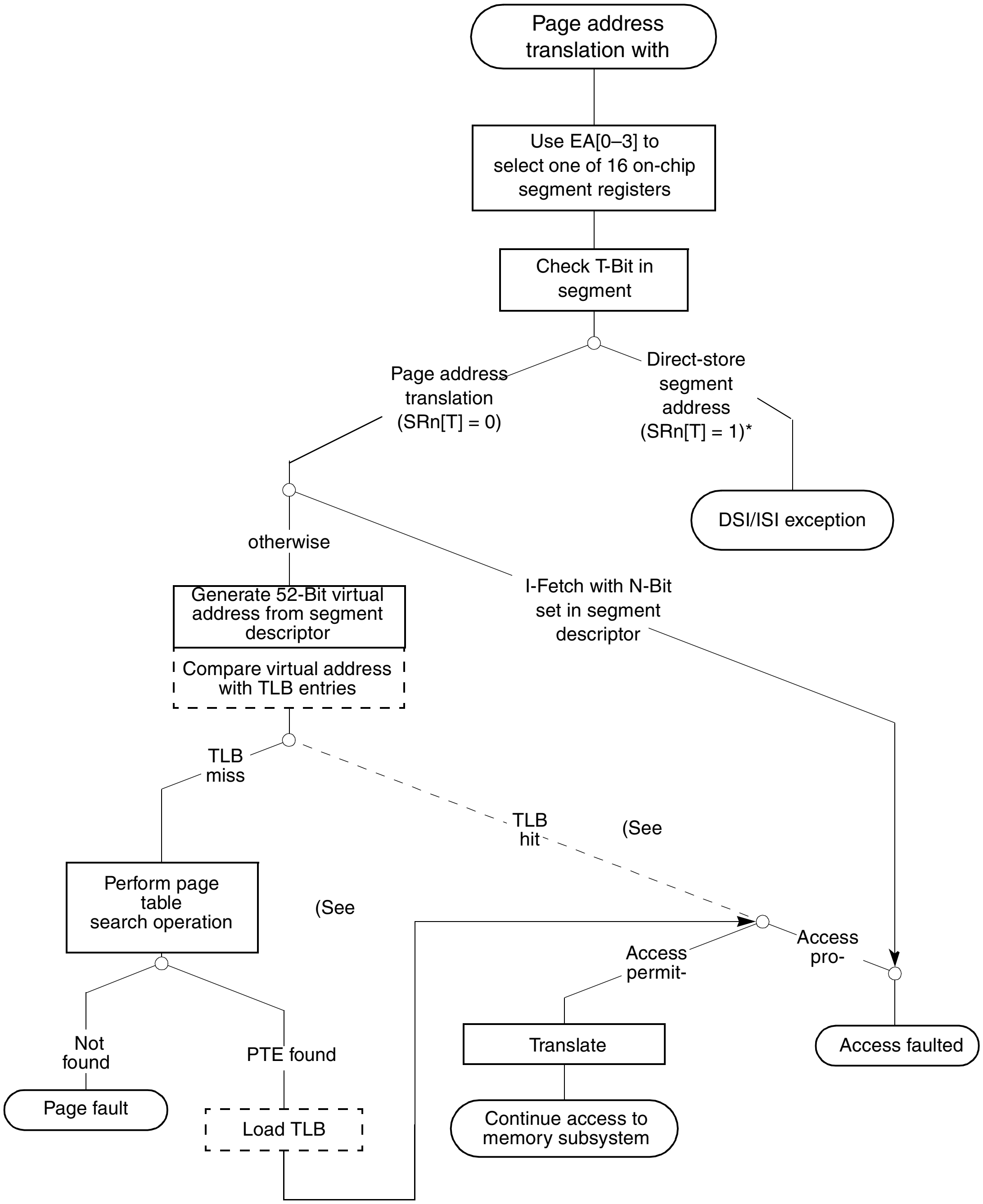
Figure 5-7. General Flow in Selection of Which Address Translation to Use

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 and the effective address information does not match a BAT array entry, the segment descriptor must be located. When the segment descriptor is located, the T bit in the segment descriptor selects whether the translation is to a page or to a direct-store segment as shown in Figure 5-8. The segment descriptor for an access is contained in one of 16 on-chip segment registers; effective address bits EA[0–3] select one of the 16 segment registers.

Note that the MPC7450 does not implement the direct-store interface, and accesses to these segments cause a DSI or ISI exception. In addition, Figure 5-8 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 Section 5.4.3, “Page Memory Protection.” 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. Because the MPC7450 implements TLBs, these branches are valid and are described in more detail throughout this chapter.



- - - - Optional to the PowerPC architecture. Implemented in the

* In the case of instruction accesses, causes ISI exception.

Figure 5-8. General Flow of Page Translation

If SR[T] = 0, 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 MPC7450 has two on-chip TLBs to cache recently used translations on-chip.

If an access hits in the appropriate TLB, page translation succeeds and the physical address bits are forwarded to the memory subsystem. If the required translation is not resident, the MMU performs a search of the page table. In this case, the MPC7450 either initiates a search of the page table in hardware or the MPC7450 traps to one of three exception handlers for the system software to perform the page table search (if HID0[STEN] = 1). If the required PTE is found, a TLB entry is allocated and the page translation is attempted again. This time, the TLB is guaranteed to hit. When the translation is located, the access is qualified with the appropriate protection bits. If the access causes a protection violation, either an ISI or DSI exception is generated.

If the PTE is not found by the table search operation, a page fault condition exists and an ISI or DSI exception occurs so software can handle the page fault.

5.1.7 MMU Exceptions Summary

To complete any memory access, the effective address must be translated to a physical address. As specified by the architecture, an MMU exception condition occurs if this translation fails for one of the following reasons:

- Page fault—There is no valid entry in the page table for the page specified by the effective address (and segment descriptor) and there is no valid BAT translation.
- An address translation is found but the access is not allowed by the memory protection mechanism.

Additionally, because the MPC7450 can use software to perform table search operations, the processor also takes an exception when HID0[STEN] = 1 and:

- There is a miss in the corresponding (instruction or data) TLB, or
- 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.

When software table searching is selected, 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 MPC7450 hardware). Therefore, it does not cause an MPC7450 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 devices that implement the PowerPC architecture, 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 5.5.5, “Implementation-Specific Software 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[1] SRR1[1] = 1
		D access: DSI exception[1] 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[2] SRR1[4] = 1
		D access: DSI exception[2] 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)	Attempt to perform load or store (including FP load or store) when SR[T] = 1	DSI exception DSISR[5] =1
Instruction fetch from guarded memory	Attempt to fetch instruction when MSR[IR] = 1 and either matching xBAT[G] = 1, or no matching BAT entry and PTE[G] = 1	ISI exception SRR1[3] =1

[1] The MPC7450 hardware vectors to these exceptions automatically when HID0[STEN] = 0. When HID0[STEN] = 1, 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.

[2] The table search software can also vector to these exception conditions.

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 MPC7450. These exception conditions map to processor exceptions as shown in Table 5-4. For example, the MPC7450 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 an alignment exception for data accesses.

For more detailed information about the conditions that cause an alignment exception (in particular for string/multiple instructions), see Section 4.6.6, “Alignment Exception (0x00600).”

Note that some exception conditions depend upon whether the memory area is set up as write-through (W = 1) or cache-inhibited (I = 1). These bits are described fully in the section titled “Memory/Cache Access Attributes,” in Chapter 5, “Cache Model and Memory Coherency,” of the *Programming Environments Manual.* Refer to Chapter 4, “Exceptions,” in this book and to Chapter 6, “Exceptions,” in the

Programming Environments Manual for a complete description of the SRR1 and DSISR bit settings for these exceptions.

Even though, for data accesses, the MPC7450 LSU initiates out-of-order accesses, the MMU prevents the changed bit in the PTE from being updated erroneously in these cases; but the LRU algorithm is updated. The MMU does not initiate exception processing for any exception conditions until the instruction that caused the exception is the next instruction to be retired. Also, the MPC7450 MMU does not initiate a search operation due to a TLB miss (including misses for **dcbt**, **dst**, and **dstst**) until the request is required by the program flow.

Table 5-4. Other MMU Exception Conditions

Condition	Description	Exception
TLB miss for an instruction fetch (HID0[STEN] = 1)	No matching entry found in IBAT or ITLB	ITLB miss exception. For details on other bits set for this exception, see Section 4.6.15, "TLB Miss Exceptions."
TLB miss for a data load access (HID0[STEN] = 1)	No matching entry found in DBAT or DTLB for data load access	DTLB miss on load exception For details on other bits set for this exception, see Section 4.6.15, "TLB Miss Exceptions."
TLB miss for a data store access, or data store access and C = 0 (HID0[STEN] = 1)	No matching entry found in DBAT or DTLB for data store access, or matching DLTB entry has C = 0 and the PTE's C bit must be set due to a data store operation	DTLB miss on store exception SRR1[11] =0 For details on the bits set for this exception, see Section 4.6.15, "TLB Miss Exceptions."
		DTLB hit on store exception with data store access and C = 0 SRR1[11] =1 For details on the bits sets during the exception, see Section 4.6.15, "TLB Miss Exceptions."
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)
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.
Load or store that results in a direct-store error	Does not occur in MPC7450	Does not apply

Table 5-4. Other MMU Exception Conditions (continued)

Condition	Description	Exception
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 a floating-point load/store, **stmw**, **stwcx.**, **lmw**, **lwarx**, **eciwx**, or **ecowx** instruction operand is not word-aligned	Alignment exception (some of these cases are implementation-specific). See Section 2.3.3, "Alignment and Misaligned Accesses."

5.1.8 MMU Instructions and Register Summary

The MMU instructions and registers allow the operating system to set up the block address translation areas and the page tables in memory.

Note that because the implementation of TLBs is optional, the instructions that refer to these structures are also optional in the architecture. However, as these structures serve as caches of the page table, the architecture specifies a software protocol for maintaining coherency between these caches and the tables in memory whenever the tables in memory are modified. When the tables in memory are changed, the operating system purges these caches of the corresponding entries, allowing the translation caching mechanism to refetch from the tables when the corresponding entries are required.

Note that the MPC7450 implements all TLB-related instructions except **tlbia**, which is treated as an illegal instruction.

Because the MMU specification for processors that implement the PowerPC architecture is so flexible, it is recommended that the software using these instructions and registers be encapsulated into subroutines to minimize the impact of migrating across the family of implementations.

Table 5-5 summarizes MPC7450 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. MPC7450 Microprocessor Instruction Summary—Control MMUs

Instruction	Description
PowerPC Instructions	
mtsr SR,**r**S	Move to Segment Register SR[SR#]← **r**S
mtsrin rS,**r**B	Move to Segment Register Indirect SR[**r**B[0–3]]←**r**S
mfsr rD,SR	Move from Segment Register **r**D←SR[SR#]

Table 5-5. MPC7450 Microprocessor Instruction Summary—Control MMUs

Instruction	Description
mfsrin rD,**r**B	Move from Segment Register Indirect **r**D←SR[**r**B[0–3]]
PowerPC Optional Instructions	
tlbie rB	TLB Invalidate Entry For effective address specified by **r**B, TLB[V]←0 The **tlbie** instruction invalidates all TLB entries indexed by the EA, and operates on both the instruction and data TLBs simultaneously invalidating four TLB entries. The index corresponds to EA[14–19]. In addition, execution of this instruction causes all entries in the congruence class corresponding to the EA to be invalidated in the other processors attached to the same bus. Software must ensure that instruction fetches or memory references to the virtual pages specified by the **tlbie** instruction have been completed prior to executing the **tlbie** instruction.
tlbsync	TLB Synchronize Synchronizes the execution of all other **tlbie** instructions in the system. Specifically, this instruction causes a global (M = 1) TLBSYNC address-only transaction (TT[0–4] = 01001) on the bus. The TLBSYNC transaction terminates normally (without a retry) when all processors on the bus have completed pending TLB invalidations. See Section 5.4.4.2, “TLB Invalidation,” for more detailed information on the **tlbsync** instruction.
Implementation-Specific Instructions[1]	
tlbld	Load Data TLB Entry Loads the contents of the PTEHI and PTELO registers into the DTLB; used for software table searching.
tlbli	Load Instruction TLB Entry Loads the contents of the PTEHI and PTELO registers into the ITLB; used for software table searching.

[1] These instructions are MPC7450-, MPC7441/MPC7451-, MPC7445/MPC7455-, MPC7447/MPC7457-, MPC7447A-, MPC7448-specific.

Table 5-6 summarizes the registers that the operating system uses to program the MPC7450 MMUs. These registers are accessible to supervisor-level software only with the **mtspr** and **mfspr** instructions. The PowerPC registers are described in Chapter 2, “Register Set,” in the *Programming Environments Manual*. For MPC7450-specific registers, see Chapter 2, “Programming Model,” of this book.

Table 5-6. MPC7450 Microprocessor MMU Registers

Register	Description
PowerPC Registers	
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). These are special-purpose registers that are accessed by the **mtspr** and **mfspr** instructions.
SDR1	The SDR1 register specifies the variables used in accessing the page tables in memory. This special-purpose register is accessed by the **mtspr** and **mfspr** instructions.
Implementation-Specific Registers	
Only MPC7445-, MPC7447-, MPC7447A-, MPC7448-, MPC7455-, and MPC7457-specific: additional BAT registers (IBAT4U–IBAT7U, IBAT4L–IBAT7L, DBAT4U–DBAT7U, and DBAT4L–DBAT7L)[1]	There are 16 additional BAT registers for the MPC7445, MPC7447, MPC7447A, MPC7448, MPC7455, and the MPC7457, organized as four pairs of instruction BAT registers (IBAT4U–IBAT7U paired with IBAT4L–IBAT7L) and four pairs of data BAT registers (DBAT4U–DBAT7U paired with DBAT4L–DBAT7L). These are special-purpose registers that are accessed by the **mtspr** and **mfspr** instructions.
SPRG4–SPRG7	The SPRG4–7 provide additional registers to be used by system software for software table searching.
TLBMISS[2]	When software table searching is enabled (HID0[STEN] = 1), and a TLB miss exception occurs, the effective address (EA[0–30]) of the instruction or data access that requires the table search is saved in the TLBMISS register.
PTEHI[2]	When software table searching is enabled (HID0[STEN] = 1), and a TLB miss exception occurs, the fields of the PTEHI register are loaded automatically with the corresponding SR[VSID] information, and the API of the missed address. The PTEHI register is also used by the **tlbli** and **tlbld** instructions.
PTELO[2]	When software table searching is enabled (HID0[STEN] = 1), and a TLB miss exception occurs, software determines the lower 32 bits of the PTE and places those bits in the PTELO register. The PTELO register is also used by the **tlbli** and **tlbld** instructions.

[1] Only MPC7445/MPC7455- and MPC7447/MPC7457-specific

[2] These registers are MPC7441/MPC7451-, MPC7445/MPC7455-, MPC447/MPC7457-specific.

5.2 Real Addressing Mode

Real addressing is used when either MSR[IR] = 0 or MSR[DR] = 0, and an instruction or data access occurs, respectively. In this case, 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. Note also, that the G bit must be set to ensure that the accesses are strongly ordered. For instruction accesses, the default memory access mode bits (WIMG) are also 0b0011. That is, instruction accesses are considered cacheable (I = 0), and the memory is guarded. Again, instruction accesses are considered cacheable even if the instruction cache is disabled in the HID0 register (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 Section 2.4.2.4, “Synchronization,” in this book, and “Synchronization Requirements for Special Registers and for Lookaside Buffers” in Chapter 2, “Register Set,” in the *Programming Environments Manual.*

5.2.1 Real Addressing Mode—32-Bit Addressing

If address translation is disabled (MSR[IR] = 0 or MSR[DR] = 0) and extended addressing is disabled (HID0[XAEN] = 0), for a particular access, the effective address is treated as the 32-bit physical address and is passed directly to the memory subsystem as described in the “Real Addressing Mode” section in Chapter 7, “Memory Management,” of the *Programming Environments Manual*. In this case only PA[4–35] bit are used and the PA[0–3] bit are cleared.

5.2.2 Real Addressing Mode—Extended Addressing

When address translation is disabled (MSR[IR] = 0 or MSR[DR] = 0) and extended addressing is enabled (HID0[XAEN] = 1), the 36-bit physical address is generated by having the system software add 4 leading zeros to the 32-bit effective address. Figure 5-6 shows how an effective address is converted to a 36-bit physical address for real addressing mode address translation.

5.3 Block Address Translation

The block address translation (BAT) mechanism in the OEA provides a way to map ranges of effective addresses larger than a single page into contiguous areas of physical memory. Such areas can be used for data that is not subject to normal virtual memory handling (paging), such as a memory-mapped display buffer or an extremely large array of numerical data.

Block address translation in the MPC7450 is described in the “Block Address Translation” section in Chapter 7, “Memory Management,” of the *Programming Environments Manual* for a 32-bit physical address. However, the information that is modified to allow for 36-bit physical addressing is described in the following sections.

The MPC7450 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 explicitly cleared before setting any BAT area for the first time and before enabling translation. Also, note that software must avoid overlapping blocks while updating a BAT area or areas. Even if translation is disabled, multiple BAT area hits (with the valid bits set) can corrupt the remaining portion (any bits except the valid bits) of the BAT registers.

Thus multiple BAT hits (with valid bits set) are considered a programming error whether translation is enabled or disabled, and can lead to unpredictable results if translation is enabled, (or if translation is disabled, when translation is eventually enabled). For the case of unused BATs (if translation is to be enabled) it is sufficient precaution to simply clear the valid bits of the unused BAT entries.

5.3.1 BAT Register Implementation of BAT Array—Extended Addressing

The BAT array is comprised of four entries used for instruction accesses and four entries used for data accesses. The BAT array maintains the address translation information for 8 blocks of memory. When using the MPC7445, MPC7447, MPC7447A, MPC7448, MPC7455, or MPC7457 because of the additional 8 BAT registers, the BAT array maintains address translation information for 16 blocks of memory. Each BAT array entry consists of a pair of BAT registers—an upper and a lower BAT register for each entry. The BAT registers are accessed with the **mtspr** and **mfspr** instructions and are only accessible to supervisor-level programs. See Appendix F, “Simplified Mnemonics,” in the *Programming Environments Manual* for a list of simplified mnemonics for use with the BAT registers. The block is defined by a pair of SPRs (upper and lower BAT registers) that contain the effective and physical addresses for the block.

The format and bit definitions of the upper and lower BAT registers for extended addressing are shown in Figure 5-10 and Figure 5-11, respectively. The upper BAT register format is the same as that for 32-bit addressing as shown in Figure 5-9. When using the MPC7445, MPC7447, MPC7447A, MPC7448, MPC7455, or MPC7457, the extended block length (XBL) for the BATs replaces BATU[15–18] reserved field, as shown in Figure 5-10. When extended addressing is used, the lower BAT contains the new BXPN and BX fields that comprise the extended physical page number.

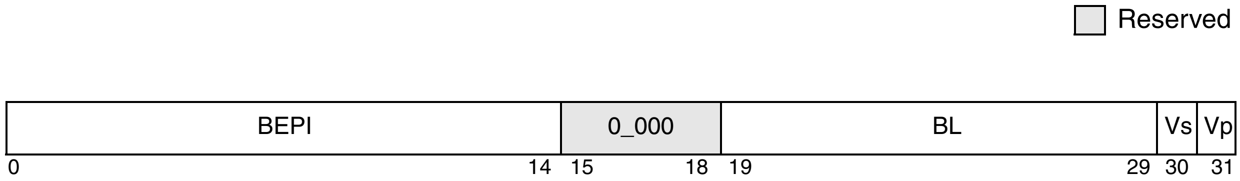


Figure 5-9. Format of Upper BAT Register (BATU)—Extended Addressing for the MPC7441, MPC7450, and MPC7451

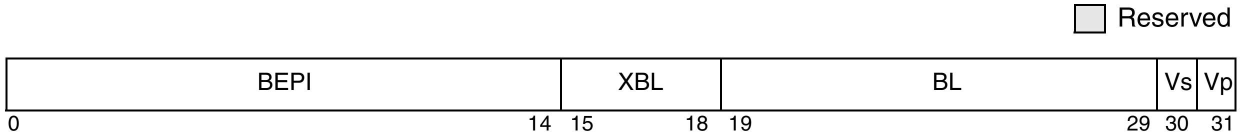
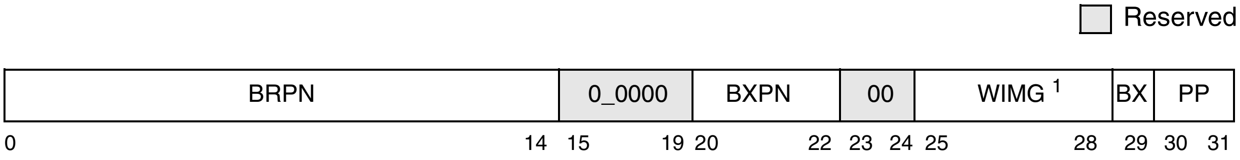


Figure 5-10. Format of Upper BAT Register (BATU)—Extended Block Size for the MPC7445, MPC7447, MPC7447A, MPC7448, MPC7455, or MPC7457



[1] W and G bits are not defined for IBAT registers. Attempting to write to these bits causes boundedly undefined results.

Figure 5-11. Format of Lower BAT Register (BATL)—Extended Addressing

The BAT registers contain the effective-to-physical address mappings for blocks of memory. This mapping information includes the effective address bits that are compared with the effective address of the access, the memory/cache access mode bits (WIMG), and the protection bits for the block. In addition, the size of the block and the starting address of the block are defined by the block physical page number (BRPN) and block size mask (BL) fields.

Table 5-7 describes the bits in the upper and lower BAT registers. Note that the W and G bits are defined for BAT registers that translate data accesses (DBAT registers); attempting to write to the W and G bits in IBAT registers causes boundedly undefined results.

Table 5-7. BAT Registers—Field and Bit Descriptions for Extended Addressing

Upper/Lower BAT[1]	Bits	Name	Description
Upper BAT Register (BAT*n*U)	0–14	BEPI	Block effective page index. This field is compared with high-order bits of the effective address to determine if there is a hit in that BAT array entry.
	15–18	—	Reserved on the MPC7441, MPC7450, and the MPC7451.
		XBL[2]	Extended block length. This XBL field is used only by the MPC7445, MPC7447, MPC7447A, MPC7448, MPC7455, and the MPC7557 to lengthen the block size. 0 When HID0[XBBSEN] is cleared at startup, BATnU[15–18] are always cleared, (0b0000), and extended BAT block size translation does not occur. 1 When HID0[XBBSEN] is set at startup, the extended BAT block size is enabled and bits BATU[15–18] become the 4 MSBs of the extended 15-bit BL field (BATU[15–29]). This allows for extended BAT block sizes of 512 MB, 1 GB, 2 GB, and 4 GB. If HID0[XBBSEN] is set at startup and then cleared after startup, the XBL bits will not clear but stay the same as they were set at startup. Values for the extended block length mask are listed in Table 5-9.
	19–29	BL	Block length. BL is a mask that encodes the size of the block. Values for this field are listed in Table 5-8.
	30	Vs	Supervisor mode valid bit. This bit interacts with MSR[PR] to determine if there is a match with the effective address. For more information, see the section, “Recognition of Addresses in BAT Arrays,” in the *Programming Environments Manual*.
	31	Vp	User mode valid bit. This bit also interacts with MSR[PR] to determine if there is a match with the effective address. For more information, see the section, “Recognition of Addresses in BAT Arrays,” in the *Programming Environments Manual*.

Table 5-7. BAT Registers—Field and Bit Descriptions for Extended Addressing (continued)

Upper/Lower BAT[1]	Bits	Name	Description
Lower BAT Register (BAT*n*L)	0–14	BRPN	Block physical page number. This field is used in conjunction with the BL field to generate high-order bits of the physical address of the block.
	15–19	—	Reserved
	20–22	BXPN [3]	Block extended physical page number (BXPN). This field comprises bits 0–2 of the physical address.
	23–24	—	Reserved
	25–28	WIMG	Memory/cache access mode bits W Write-through I Caching-inhibited M Memory coherence G Guarded Attempting to write to the W and G bits in IBAT registers causes boundedly-undefined results. For detailed information about the WIMG bits, see Section 3.3.1, “Memory/Cache Access Attributes (WIMG Bits).”
	29	BX[3]	Block extended physical page number (BX). This field comprises bit 3 of the physical address.
	30–31	PP	Protection bits for block. This field determines the protection for the block as described in the section, Block Memory Protection,” in the *Programming Environments Manual*.

[1] A context synchronizing instruction must follow a mtspr.

[2] Specific bits are only for the MPC7445, MPC7447, MPC7447A, MPC7448, MPC7455, and MPC7457.

[3] MPC7450-, MPC7441-/MPC7451-, MPC7445-/MPC7455-specific bits.

BAT*n* registers can be accessed with **mtspr** and **mfspr**. For synchronization requirements on the BAT*n* registers see Table 2-46.

The BL field in the upper BAT register is a mask that encodes the size of the block. Table 5-8 defines the bit encoding for the BL field of the upper BAT register (the same as for 32-bit physical addressing on the MPC7441, MPC7450, and the MPC7451).

Table 5-8. Upper BAT Register Block Size Mask Encoding

Block Size	BATU[BL] Encoding
128 Kbytes	000 0000 0000
256 Kbytes	000 0000 0001
512 Kbytes	000 0000 0011
1 Mbyte	000 0000 0111
2 Mbytes	000 0000 1111
4 Mbytes	000 0001 1111
8 Mbytes	000 0011 1111

Table 5-8. Upper BAT Register Block Size Mask Encoding (continued)

Block Size	BATU[BL] Encoding
16 Mbytes	000 0111 1111
32 Mbytes	000 1111 1111
64 Mbytes	001 1111 1111
128 Mbytes	011 1111 1111
256 Mbytes	111 1111 1111

Only the values shown in Table 5-8 are valid for BL. An effective address is determined to be within a BAT area if the appropriate bits (determined by the BL field) of the effective address match the value in the BEPI field of the upper BAT register and if the appropriate valid bit (Vs or Vp) is set. Note that for an access to occur, the protection bits (PP bits) in the lower BAT register must be set appropriately, as described and defined in Chapter 7, “Memory Management,” in the *Programming Environments Manual*.

The number of zeros in the BL field determines the bits of the effective address that are used in the comparison with the BEPI field to determine if there is a hit in that BAT array entry. The right-most bit of the BL field is aligned with bit 14 of the effective address; bits of the effective address corresponding to ones in the BL field are then cleared to zero for the comparison.

The value loaded into the BL field determines both the size of the block and the alignment of the block in both effective address space and physical address space. The values loaded into the BEPI and BRPN fields must have at least as many low-order zeros as there are ones in BL. Otherwise, the results are undefined.

5.3.2 Block Physical Address Generation—Extended Addressing

When extended addressing is enabled (HID0[XAEN] = 1) and the block protection mechanism validates the access, then a 36-bit physical address is formed as shown in Figure 5-12. Bits in the effective address corresponding to ones in the BL field, concatenate with the 17 low-order bits of the effective address, and form the offset within the block of memory defined by the BAT array entry. Bits in the effective address corresponding to zeros in the BL field are then logically ORed with the corresponding bits in the BRPN field to form the next high-order bits of the physical address. The highest-order four bits of the BRPN field (BATL[0–3]) form bits 4–7 of the physical address (PA[4–7]). Finally, the four extended address bits from BATL[BXPN] and BATL[BX] are concatenated to form the highest-order four bits of the physical address (PA[0–2] and PA[3], respectively).

Figure 5-12 shows how a block physical address is generated for extended addressing.

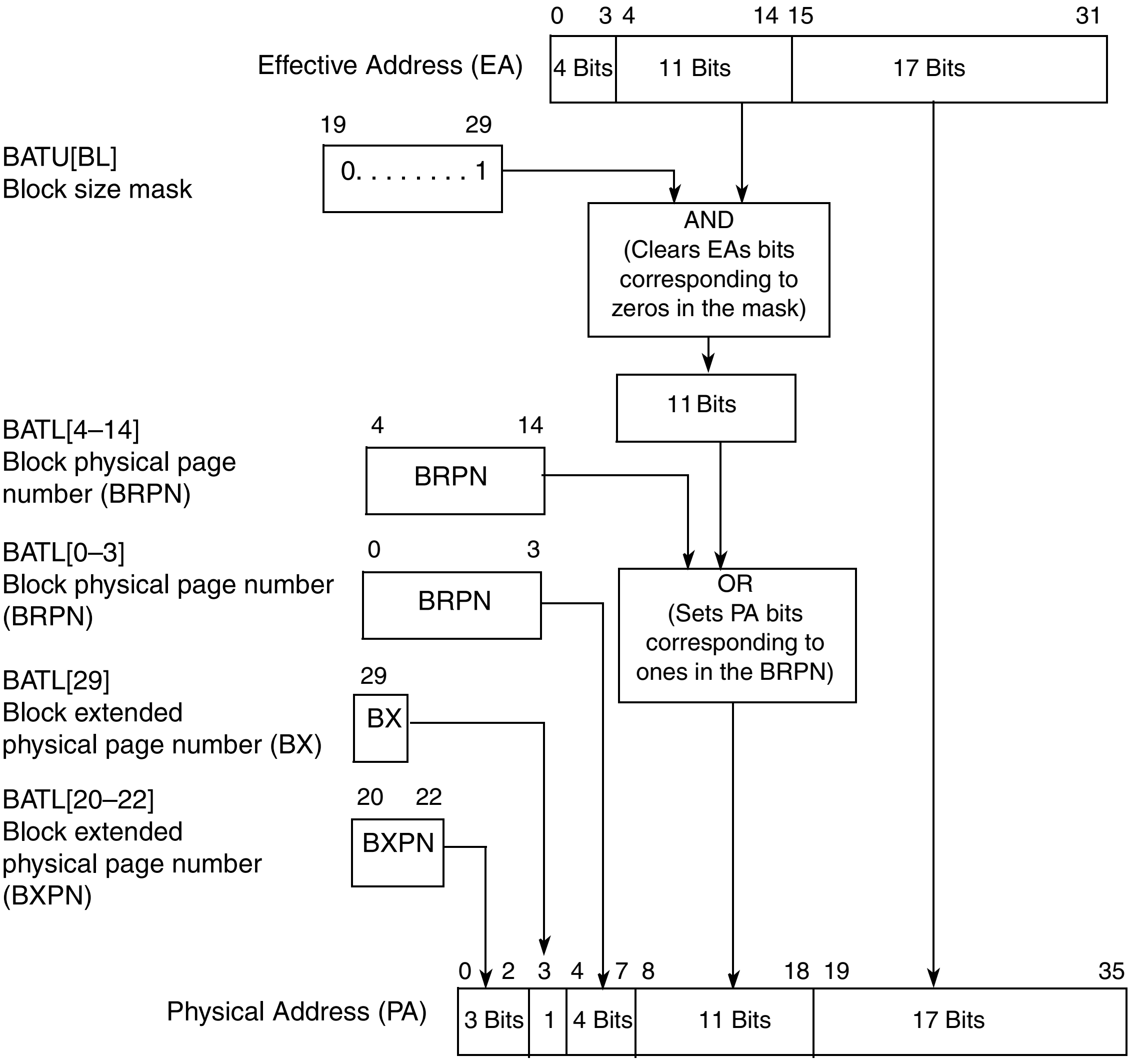


Figure 5-12. Block Physical Address Generation—Extended Addressing

5.3.2.1 Block Physical Address Generation with an Extended BAT Block Size

On the MPC7445, MPC7447, MPC7447A, MPC7455, MPC7457 and MPC7448, when the extended BAT block size is enabled (HID0[XBBSEN]=1) the BAT block size is increased through the XBL field in the Upper BAT register, as shown in Figure 5-10. This allows for extended BAT block sizes of 512MB, 1 GB, 2GB, and 4 GB. If HID0[XBBSEN] is set at startup and then cleared after startup, the XBL bits do not clear but stay the same as they were set at startup. The BL field is extended to 15 bits, with the XBL bits becoming the 4 most significant bits (MSBs) for the block size. The encoding for the extended BL field is shown in Table 5-9.

Table 5-9. Upper BAT Register Block Size Mask Encoding when the Extended Block Size is Enabled (HID0[XBBSEN] = 1)

Block Size	BATU[XBL + BL] Encoding
128 Kbytes	000 0000 0000 0000
256 Kbytes	000 0000 0000 0001
512 Kbytes	000 0000 0000 0011
1 Mbyte	000 0000 0000 0111
2 Mbytes	000 0000 0000 1111
4 Mbytes	000 0000 0001 1111
8 Mbytes	000 0000 0011 1111
16 Mbytes	000 0000 0111 1111
32 Mbytes	000 0000 1111 1111
64 Mbytes	000 0001 1111 1111
128 Mbytes	000 0011 1111 1111
256 Mbytes	000 0111 1111 1111
512 Mbytes	000 1111 1111 1111
1 Gbytes	001 1111 1111 1111
2 Gbytes	011 1111 1111 1111
4 Gbytes	111 1111 1111 1111

Only the values shown in Table 5-9 are valid for an extended 15-bit BL field. An effective address is determined to be within a BAT area if the appropriate bits (determined by the XBL and BL fields) of the effective address match the value in the 15-bit BEPI field of the upper BAT register and if the appropriate valid bit (Vs or Vp) is set.

The number of zeros in the extended BL field determines the bits of the effective address that are used in the comparison with the BEPI field to determine if there is a hit in that BAT array entry. The right-most bit of the BL field is still aligned with bit 14 of the effective address; bits of the effective address corresponding to ones in the BL field are then cleared to zero for the comparison.

The value loaded into the BL field determines both the size of the block and the alignment of the block in both effective address space and physical address space. The values loaded into the BEPI and BRPN fields must have at least as many low-order zeros as there are ones in BL, otherwise the results are undefined.

Figure 5-13 shows how a block physical address is generated for an extended block size with extended addressing (36-bit physical address).

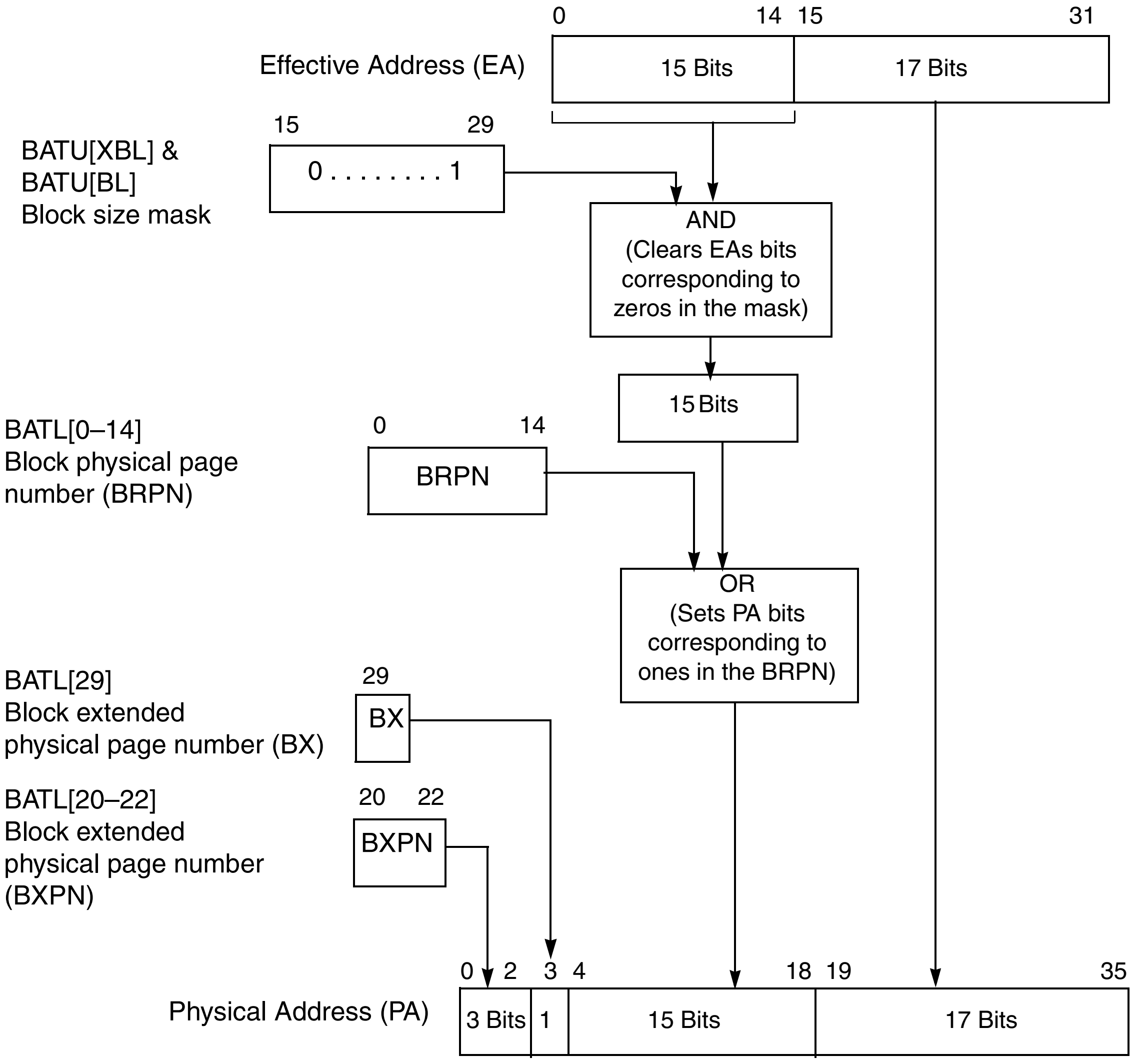


Figure 5-13. Block Physical Address Generation—Extended Block Size for a 36-Bit Physical Address

5.3.3 Block Address Translation Summary—Extended Addressing

Figure 5-14 is an expansion of the ‘BAT Array Hit’ branch of Figure 5-7 and shows the translation of address bits when extended addressing is enabled (HID0[XAEN] = 1) so that a 36-bit physical address is generated. Extended address bits from the lower BAT register are concatenated to the highest order bits of the physical address. Note that the figure does not show when many of the exceptions in Table 5-3 are detected or taken as this is implementation-specific. For further details on memory protection violations see the section, “Block Memory Protection,” of the *Programming Environments Manual.*

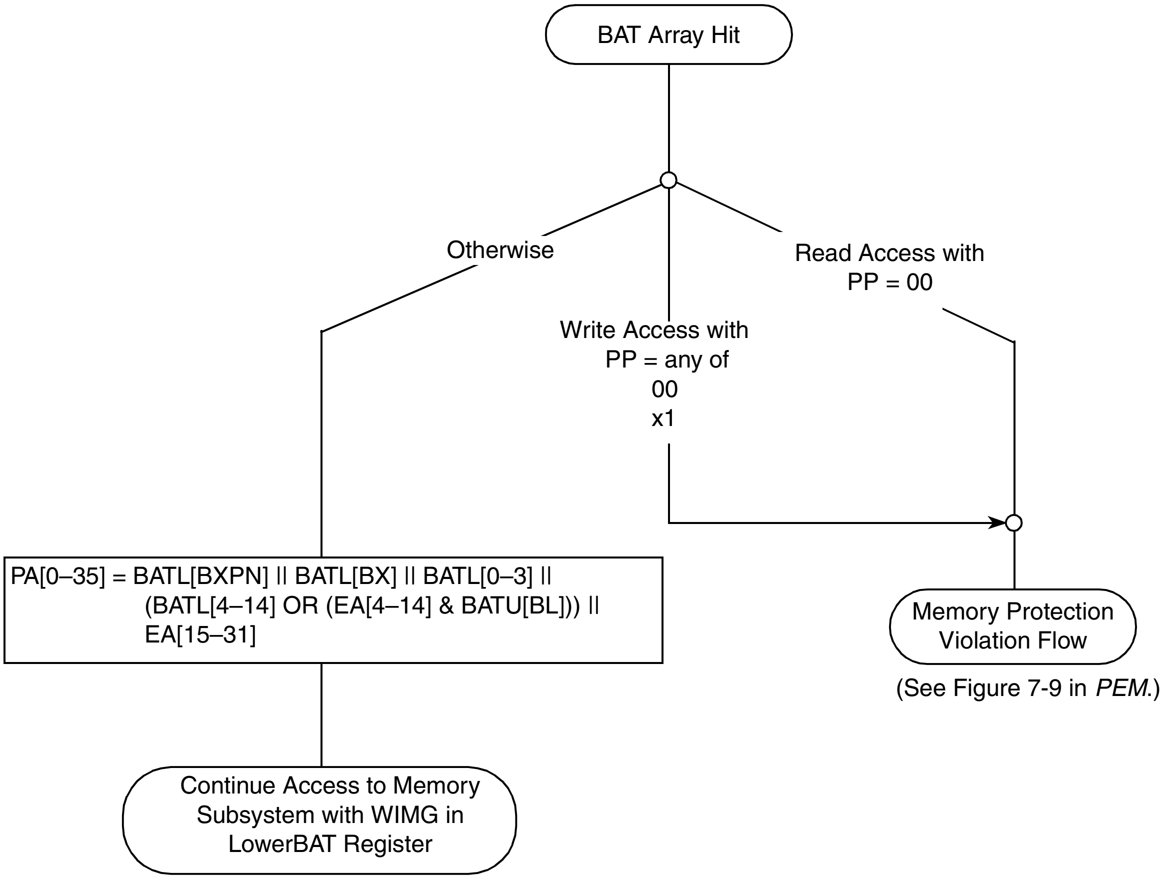


Figure 5-14. Block Address Translation Flow—Extended Addressing

In the MPC7445, MPC7447, MPC7447A, MPC7448, MPC7455, and MPC7457, Figure 5-15 shows translation of address bits when the extended block size is enabled (HID0[XBBSEN]=1) and extended addressing is enabled (HID0[XAEN] = 1). In this case all 15 bits of the effective address are compared with the BEPI field to determine if there is a hit in the BAT array. Once a match has been found, the physical address is generated by using all the bits in the effective address that correspond to zeros in the BL field. The result is then logically ORed with the BRPN field to form bits 4–18 of the physical address.

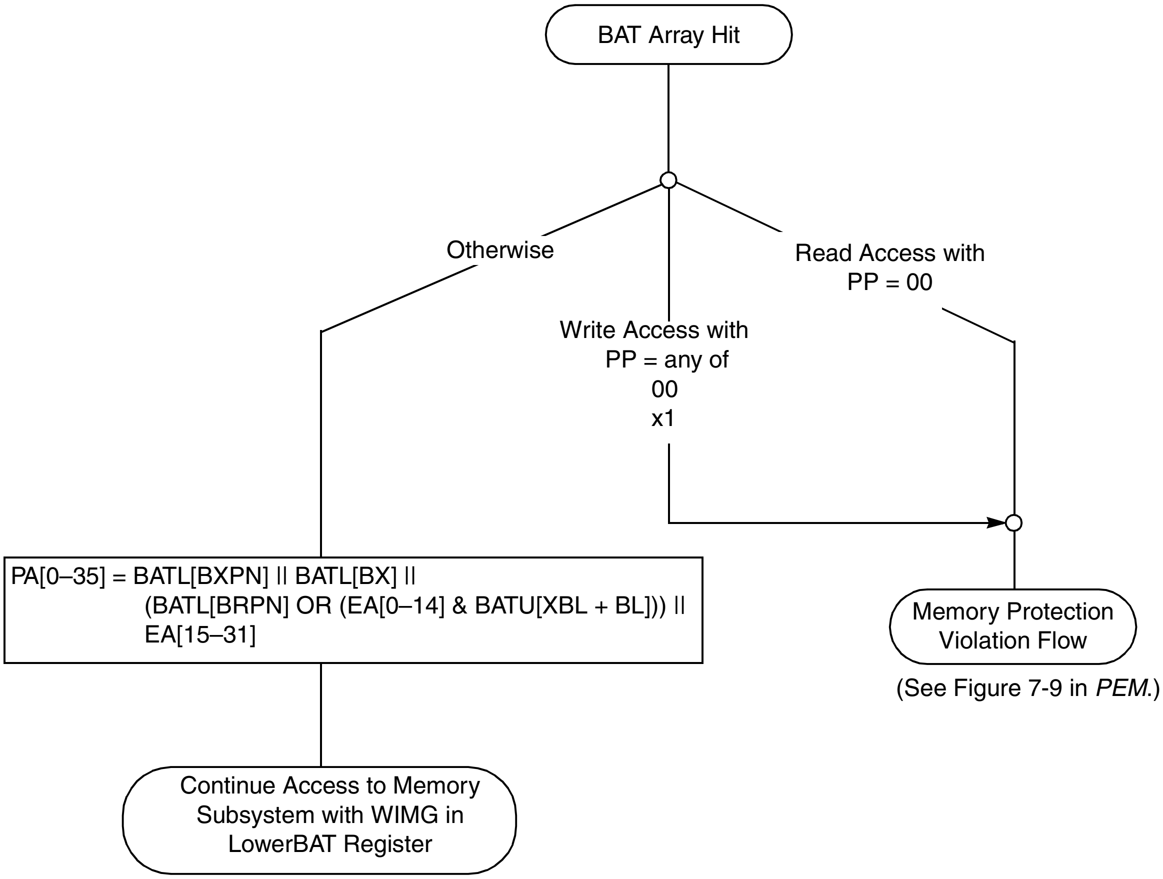


Figure 5-15. Block Address Translation Flow—Extended Block Size for a 36-Bit Physical Address

5.4 Memory Segment Model

The MPC7450 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 there is not a BAT hit, the page address translation proceeds in the following two steps:

1. From effective address to the virtual address (that 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)
2. From virtual address to physical address

The following subsections highlight those areas of the memory segment model defined by the OEA that are specific to the MPC7450 as well as modifications that apply for extended 36-bit physical addressing. The memory segment model for 32-bit physical addressing is as described in Chapter 7, “Memory Management,” in the *Programming Environments Manual*.

5.4.1 Page Address Translation Overview

A page address translation overview for 32-bit physical addresses is provided in the section, "Page Address Translation Overview," of the *Programming Environments Manual*. The following sections highlight the differences for 36-bit physical addressing. The general flow for page address translation is as shown in Figure 5-16. The effective address, EA[0–3], is used to find the correct segment descriptor in the segment registers. The segment descriptor is then used to generate the 52-bit virtual address (VA). The MMU then fetches the page table entry (PTE) from the virtual address. If the PTE is not found in the tables then a hardware or software page table search is performed. The following subsections describe the details of how page address translation is performed for an extended 36-bit physical address.

The translation of an effective address to an extended physical address is shown in Figure 5-16. Note that in the process of translating the physical address, a 52-bit virtual address is generated and that is used to find the PTE in the on-chip TLB or through a hardware or software table search operation. The physical address translation is as follows:

- Bits 0–3 of the effective address comprise the segment register number used to select a segment descriptor, from which the virtual segment ID (VSID) is extracted.
- Bits 4–19 of the effective address bits correspond to the page number within the segment. EA[4–9] define the abbreviated page index (API), and EA[10–13] define the extended API (EAPI) bits in the PTE. EA[4–19] are concatenated with the VSID from the segment descriptor to form the virtual page number (VPN). The VPN is used to search for the PTE in either an on-chip TLB or the page table. The PTE then provides the physical page number (RPN) and the extended page number bits (XPN and X). The XPN and X fields of the page table entry (PTE) provide the extra bits for the extended physical page number. These become the most significant bits of the 36-bit physical address (PA[0–3]).
- Bits 20–31 of the effective address are the byte offset within the page; these are bits 24–35 of the physical address used to access memory.

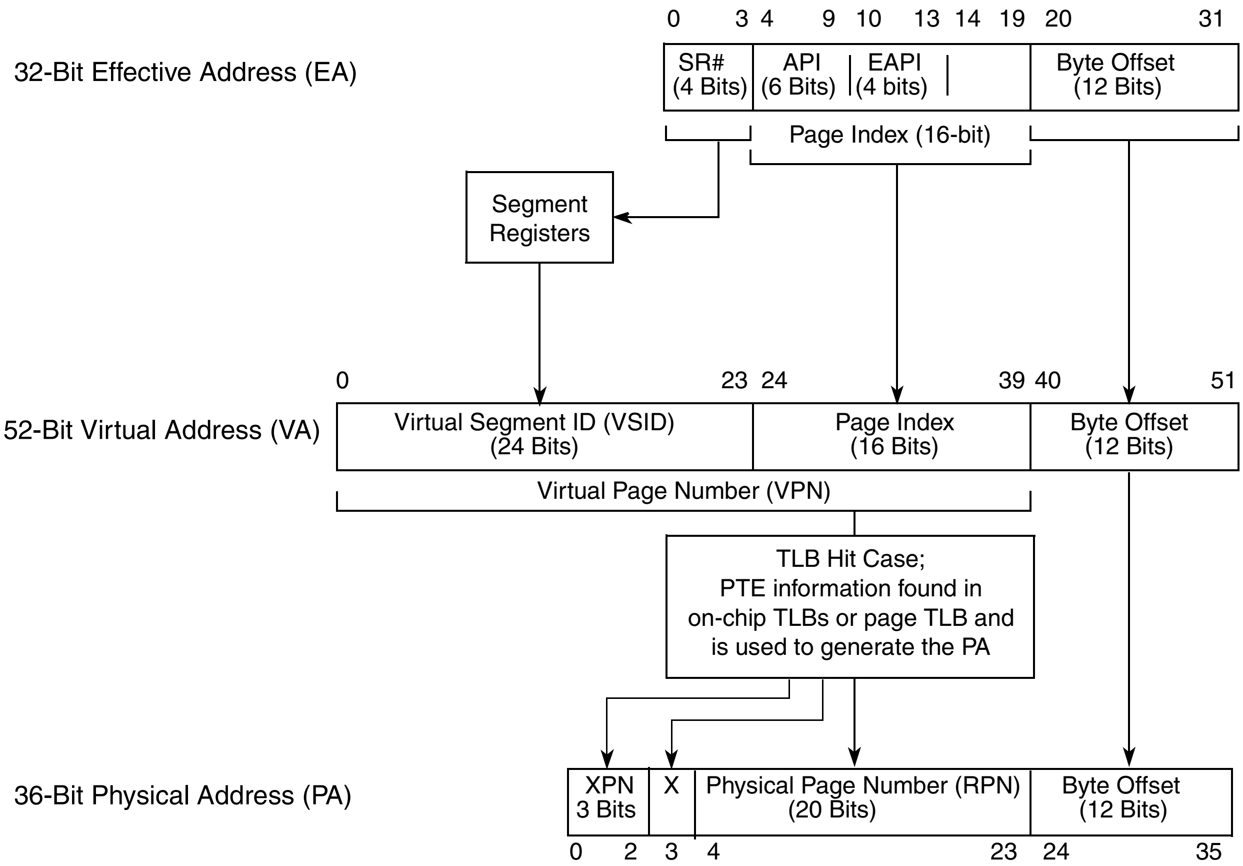


Figure 5-16. Generation of Extended 36-Bit Physical Address for Page Address Translation

5.4.1.1 Segment Descriptor Definitions

The segment registers are defined the same for both 32-and 36-bit physical addressing. See the description of the segment register format in the “Segment Descriptor Format” section of Chapter 7, “Memory Management,” in the *Programming Environments Manual*. The segment descriptors are 32 bits long and reside in one of the 16 on-chip segment registers. The fields in the segment register are interpreted differently depending on the value of the T bit. When T=1 (SR*n*[T] = 1), the segment descriptor defines a direct-store segment; however, the MPC7450 does not support the direct-store interface. When an access is determined to be to the direct-store interface space, the MPC7450 takes a DSI exception if it is a data access (see Section 4.6.3, “DSI Exception (0x00300)”), and takes an ISI exception if it is an instruction access (see Section 4.6.4, “ISI Exception (0x00400)”).

5.4.1.2 Page Table Entry (PTE) Definition—Extended Addressing

The definition of a page table entry for 32-bit physical addressing is as described in the section, “PTE Format,” of Chapter 7, “Memory Management,” in the *Programming Environments Manual*. The PowerPC OEA defines PTEs that are 64 bits in length. This section highlights the aspects of page address translation that are unique for 36-bit physical addresses.

Figure 5-17 shows the format of the two words that comprise a PTE for a 36-bit physical address (HID0[XAEN] = 1).

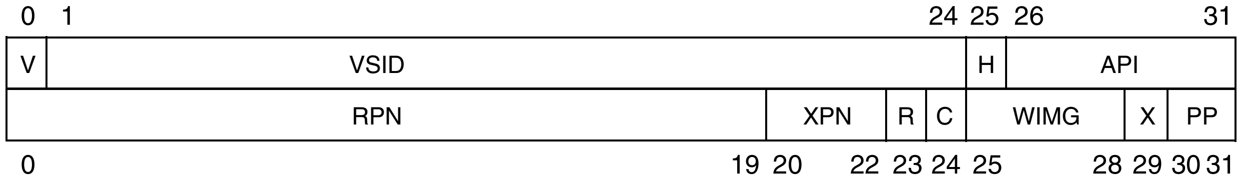


Figure 5-17. Page Table Entry Format—Extended Addressing

Table 5-10 lists the corresponding bit definitions for each word in a PTE as defined above.

Table 5-10. PTE Bit Definitions

Word	Bit	Name	Description
0	0	V	Entry valid (V = 1) or invalid (V = 0)
	1–24	VSID	Virtual segment ID
	25	H	Hash function identifier
	26–31	API	Abbreviated page index
1	0–19	RPN	Physical page number
	20–22	XPN	Extended page number provides physical address bits 0-2.
	23	R	Referenced bit
	24	C	Changed bit
	25–28	WIMG	Memory/cache control bits
	29	X	Extended page number provides physical address bit 3
	30–31	PP	Page protection bits

A PTE contains an abbreviated page index rather than the complete page index field because at least ten of the low-order bits of the page index are used in the hash function to select a PTEG address (PTEG addresses define the location of PTE). Therefore, these ten low-order bits are not repeated in the 8 PTEs of that PTEG. The XPN and X fields have been added to form the extended page number. When extended addressing is not enabled (HID0[XAEN] = 0), the four most significant bits of the physical address are zeros, regardless of the XPN and X values of a PTE.

5.4.2 Page History Recording

Referenced (R) and changed (C) bits in each PTE keep history information about the page. When hardware table searching is enabled, the history bits are maintained by a combination of the MPC7450 table search hardware and the system software. When software table searching is enabled, the history bits are maintained by a combination of the following:

- Table search software provided by the exception
- Exception model

The operating system uses the information in each PTE to determine which areas of memory to write back to disk when new pages must be allocated in main memory

Referenced and changed recording is performed only for accesses made with page address translation and not for translations made with the BAT mechanism or for accesses that correspond to direct-store (T = 1) segments. Furthermore, R and C bits are maintained only for accesses made while address translation is enabled (MSR[IR] = 1 or MSR[DR] = 1).

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. Software table searching is optional in the MPC7450. When software table searching is enabled (HID0[STEN] = 1), the software table search routines are responsible for setting the R bit when a PTE is accessed. Additionally, the MPC7450 also causes an exception (to vector to the software table search routines) when the C bit in the corresponding TLB entry (and PTE entry) requires updating.

In the MPC7450, the referenced and changed bits are updated as follows:

- For TLB hits, the C bit is updated according to Table 5-11.
- 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 5-11. Table Search Operations to Update History Bits—TLB Hit Case

R and C bits in TLB Entry	Processor Action
00	Combination does not occur
01	Combination does not occur
10	Read: No special action 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 5-11 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.

In some previous implementations, the **dcbt** and **dcbtst** instructions execute only if there is a TLB/BAT hit or if the processor is in real addressing mode. In case of a TLB or BAT miss, these instructions are treated as no-ops and do not initiate a table search operation, and do not set either the R or C bits. In the MPC7450, the **dcbt**, **dcbtst**, and data stream touch instructions (**dst**[**t**] and **dstst**[**t**]) do cause a table search operation in the case of a TLB miss. However, they never cause the C bit to be set, and a failed table search operation does not cause an exception.

As defined by the PowerPC architecture, the referenced and changed bits are updated as if address translation were disabled (real addressing mode). If these update accesses hit any of the on-chip caches, they are not seen on the external bus. If they miss in the on-chip caches, they are performed as typical cache line fill accesses on the bus (if the data cache is enabled), or as discrete read and write accesses (if the data cache is disabled).

5.4.2.1 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 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 MPC7450 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 systems include the following:

- Fetching of instructions not subsequently executed
- A memory reference caused by a speculatively executed instruction that is mispredicted
- Accesses generated by an **lswx** or **stswx** instruction with a zero length
- Accesses generated by an **stwcx.** instruction when no store is performed because a reservation does not exist
- Accesses that cause exceptions and are not completed

5.4.2.2 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 MPC7450). 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 the C bit is already set, it is not updated. If the TLB changed bit is 0, the MPC7450 initiates a table search operation to set the C bit in the corresponding PTE in the page table. The MPC7450 then reloads the TLB (with the C bit set). This occurs automatically when hardware table searching is enabled. When software table searching is enabled, the MPC7450 takes a data TLB miss on store exception for this case so that the software can perform the table search operation to set the C bit. Refer to Section 5.5.5, “Implementation-Specific Software 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 the store is guaranteed to be in the execution path (unless an exception, other than those caused by the **sc**, **rfi**, or trap instructions, occurs). Furthermore, the following conditions 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.
- 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 the execution of the **dcbt**, **dcbtst** and data stream touch instructions (**dst**[**t**] and **dstst**[**t**]) never cause the C bit to be set.

5.4.2.3 Scenarios for Referenced and Changed Bit Recording

This section provides a summary of the model (defined by the PowerPC OEA) that is used by 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. Note that when the MPC7450 updates the R and C bits in memory, the accesses are performed as if MSR[DR] = 0 and G = 0 (that is, as nonguarded cacheable operations in which coherency is required—WIMG = 0010).

When software table searching is enabled, the MPC7450 does not maintain the R and C bits in hardware, and software assistance is required. In this case, the information in this section still applies, except that the software performing the updates is constrained to the rules described (that is, the software must set bits shown as guaranteed to be set and not set bits shown as guaranteed not to be set).

Table 5-12 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 MPC7450, the combination of the MPC7450 itself and the software used to search the page tables described in Section 5.5.5, “Implementation-Specific Software Table Search Operation,” is assumed. For more information, see “Page History Recording” of the *Programming Environments Manual*.

Table 5-12. Model for Guaranteed R and C Bit Settings

Priority	Scenario	Causes Setting of R Bit		Causes Setting of C Bit	
		OEA	MPC7450	OEA	MPC7450
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. Would be required by the sequential execution model in the absence of system-caused or imprecise exceptions, or of floating-point assist exception for instructions that would cause no other kind of precise exception.	Maybe[1]	No	No	No
5	All other out-of-order store operations	Maybe [1]	No	Maybe [1]	No
6	Zero-length load (**lswx**)	Maybe	No	No	No

Table 5-12. Model for Guaranteed R and C Bit Settings (continued)

Priority	Scenario	Causes Setting of R Bit		Causes Setting of C Bit	
		OEA	MPC7450	OEA	MPC7450
7	Zero-length store (**stswx**)	Maybe [1]	No	Maybe [1]	No
7.5	Store that triggers a precise exception (DSI, ALI)	Maybe	Yes	Maybe	Maybe
8	Store conditional (**stwcx.**) that does not store	Maybe [1]	Yes	Maybe [1]	Yes
9	In-order instruction fetch	Yes[2]	Yes	No	No
10	Load instruction or **eciwx**	Yes	Yes	No	No
11	Store instruction, **ecowx** or **dcbz** instruction	Yes	Yes	Yes	Yes
12	**icbi**, **dcbt**, or **dcbtst** instruction	Maybe	No	No	No
13	**dcbst** or **dcbf** instruction	Maybe	Yes	No	No
14	**dcbi** instruction	Maybe [1]	Yes	Maybe [1]	Yes
15	**dst** instruction	n/a	Yes	n/a	No

[1] If C is set, R is guaranteed to be set also.

[2] Includes the case in which the instruction is fetched out-of-order and R is not set (does not apply for MPC7450).

5.4.3 Page Memory Protection

The MPC7450 implements page memory protection as it is defined in the section, “Page Memory Protection,” of the *Programming Environments Manual*.

5.4.4 TLB Description

The MPC7450 implements separate 128-entry data and instruction TLBs to maximize performance. This section describes the hardware resources provided in the MPC7450 to facilitate page address translation. Note that the hardware implementation of the MMU is not specified by the architecture, and while this description applies to the MPC7450, it does not necessarily apply to other processors that implement the PowerPC architecture.

5.4.4.1 TLB Organization and Operation

Because the MPC7450 has two MMUs (IMMU and DMMU) that operate in parallel, some of the MMU resources are shared, and some are actually duplicated (shadowed) in each MMU to maximize performance. For example, although the architecture defines a single set of segment registers for the MMU, the MPC7450 maintains two identical sets of segment registers, one for the IMMU and one for the DMMU; when an instruction that updates the segment register executes, the MPC7450 automatically updates both sets.

The TLB entries contain on-chip copies of PTEs in the page tables in memory and are similar in structure. To uniquely identify a TLB entry as the required PTE, the TLB entry also contains four more bits of the page index, EA[10–13], called the extended API (EAPI) in addition to the API bits in the PTE.

Each TLB contains 128 entries organized as a two-way set-associative array with 64 sets as shown in Figure 5-18 for the DTLB (the ITLB organization is the same). When an address is being translated, a set of two TLB entries is indexed in parallel with the access to a segment register. If the address in one of the two TLB entries is valid and matches the 40-bit virtual page number, that TLB entry contains the translation. If no match is found, a TLB miss occurs.

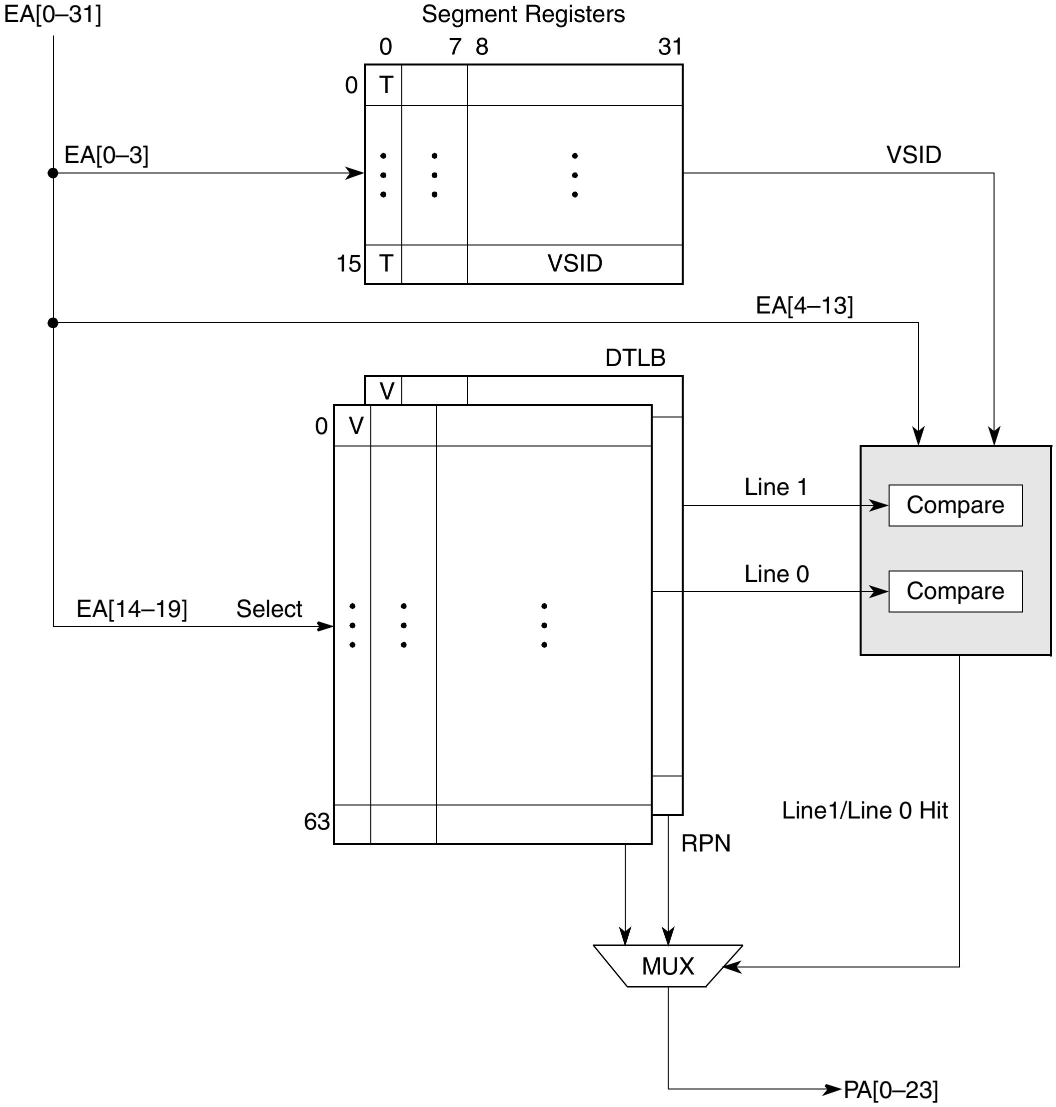


Figure 5-18. Segment Register and DTLB Organization

Unless the access is the result of an out-of-order access, when HID0[STEN] = 0, a hardware table search operation begins if there is a TLB miss. If the access is out-of-order, the table search operation is postponed until the access is required; at that point the access is no longer out-of-order. When the matching PTE is found in memory, it is loaded into the TLB entry selected by the least-recently-used (LRU) replacement algorithm, and the translation process begins again, this time with a TLB hit.

A software table search is initiated when HID0[STEN] = 1 and a TLB miss occurs. In this case, MPC7450 causes an exception when the TLB and BAT both miss for an access. There are separate exception vectors for instruction fetches, data loads, and data stores. Refer to Section 5.5.5, “Implementation-Specific Software Table Search Operation,” for more information on the loading of the TLBs in this case.

Each set of TLB entries has one associated LRU bit. The LRU bit for a set is updated any time either entry is used, even if the access is speculative. Invalid entries are always the first to be replaced.

Although both MMUs can be accessed simultaneously (both sets of segment registers and TLBs can be accessed in the same clock), only one exception condition is reported at a time. Exceptions are processed in strict program order, and a particular exception is processed when the instruction that caused it is the next instruction to be retired. When a particular instruction causes an instruction MMU exception, that exception is processed before that instruction can cause a data MMU exception.

ITLB miss conditions 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 the next instruction to be retired. In the case that both an ITLB and DTLB miss are reported in the same clock, the DTLB miss takes precedence and is handled first. Refer to Chapter 6, “Instruction Timing,” for more detailed information about the internal pipelines and the reporting of exceptions.

Although address translation is disabled on a soft or hard reset condition, the valid bits of TLB entries are not automatically cleared. Thus TLB entries must be explicitly cleared by the system software (with a series of **tlbie** instructions) before 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.

5.4.4.2 TLB Invalidation

The MPC7450 implements the optional **tlbie** and **tlbsync** instructions, that are used to invalidate TLB entries.

The **tlbia** instruction is not implemented on the MPC7450 and when its opcode is encountered, an illegal instruction program exception is generated. To invalidate all entries of both TLBs, 64 **tlbie** instructions must be executed, incrementing the value in EA[14–19] by one each time. See Chapter 8, “Instruction Set,” in the *Programming Environments Manual* for architecture information about the **tlbie** instruction.

5.4.4.2.1 tlbie Instruction

The execution of the **tlbie** instruction always invalidates four entries—for each ITLB set and each DTLB set, one entry in each of the two ways is indexed by EA[14–19]. The **tlbie** instruction executes regardless of the setting of the MSR[DR] and MSR[IR] bits.

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. When an MPC7450 processor executes a **tlbie** instruction, it always broadcasts this operation on the system bus as a global (M = 1) TLBIE address-only transaction (TT[0–4] = 11000) with the 32-bit effective (not physical) address reflected on the address bus. Figure 5-19 shows the flow of events caused by execution of the **tlbie** instruction as well as the actions taken by the MPC7450 when a TLBIE transaction is detected on the processor bus.

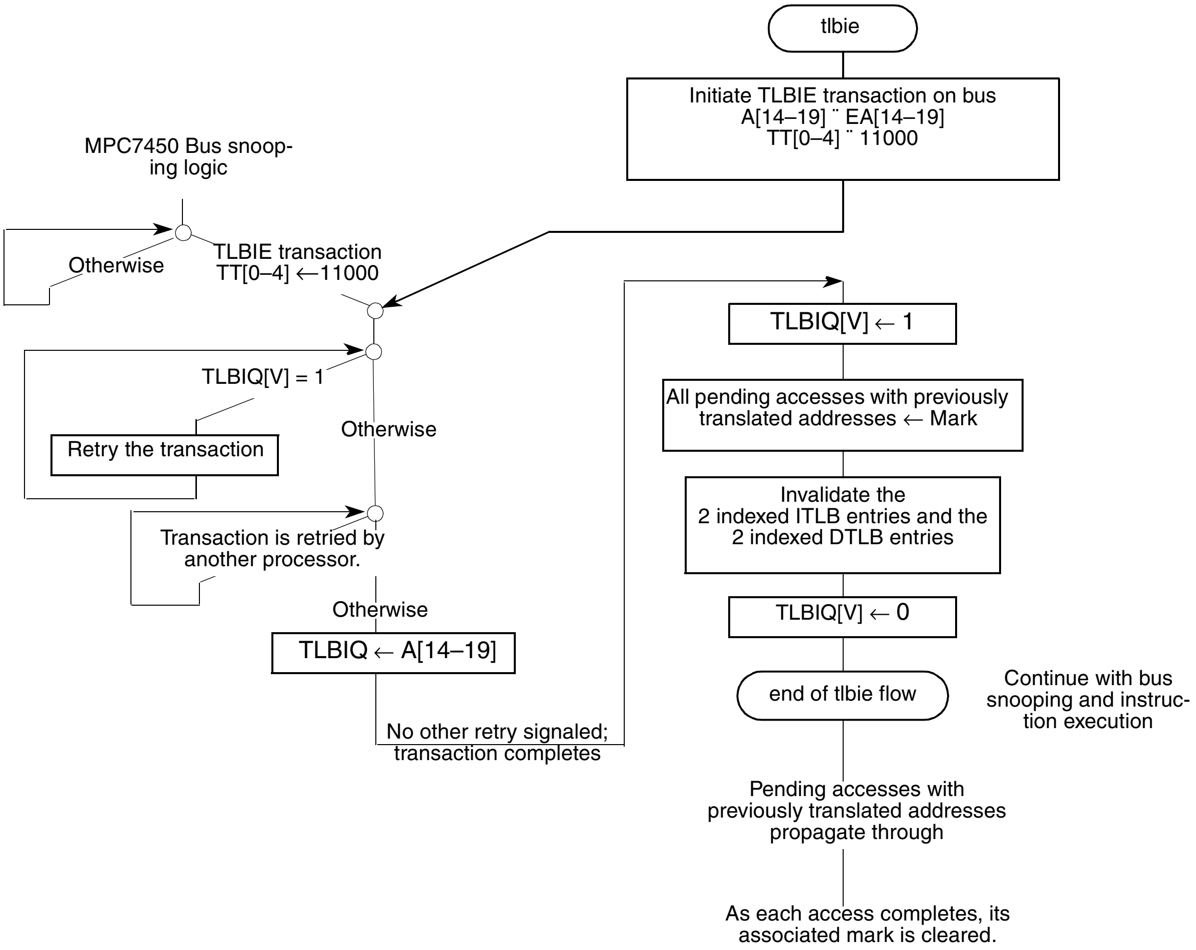


Figure 5-19. tlbie Instruction Execution and Bus Snooping Flow

The execution of the **tlbie** instruction is performed as if the TLBIE operation was snooped from the system bus by loading a single-entry TLBIQ that contains EA[14–19] and a valid bit. When the invalidation of the TLBs is complete, the TLBIQ is invalidated. Also, all valid queues in the machine that contain a previously translated address (physical address) are internally marked because these queues could contain references to addresses from the just invalidated TLB entries. These references propagate through to completion, but are marked for the purposes of synchronizing multiple TLB invalidations in multiple processors. See Section 5.4.4.2.2, “tlbsync Instruction,” for more information on the use of these internal marks.

When another processor on the system bus performs a TLBIE address-only transaction, the MPC7450 snoops the transaction and checks the status of its internal TLBIQ. If the TLBIQ is valid (that is, the processor is in the process of performing a TLB invalidation), it causes a retry of the transaction until the TLBIQ empties. If the TLBIQ is invalid and the transaction is not retried by any other processor, the MPC7450 loads the TLBIQ with EA[14–19] and sets the TLBIQ valid bit. This causes the MPC7450 to invalidate the four TLB entries (both the ITLB and DTLB entries indexed by EA[14–19]), and internally mark all accesses with previously translated addresses.

The **tlbie** instruction does not affect the instruction fetch operation—that is, the prefetch buffer is not purged and the machine does not cause these instructions to be refetched.

5.4.4.2.2 tlbsync Instruction

The **tlbsync** instruction ensures that all previous **tlbie** instructions executed by the system have completed. Specifically, **tlbsync** causes a global (M = 1) TLBSYNC address-only transaction (TT[0–4] = 01001) on the bus if that processor has completed all previous **tlbie** instructions and any memory operations based on the contents of those invalidated TLB entries have propagated through to completion.

Execution of a **tlbsync** instruction affects outstanding VTQ operations in the same way as a **sync** instruction, (see Chapter 7, “AltiVec Technology Implementation”) with the following additional effect: an outstanding table search operation for a VTQ-initiated access is cancelled when **tlbsync** is dispatched to the LSU, possibly causing a line fetch skip as described in Section 5.5.2, “Page Table Search Operations—Implementation.”

The **tlbsync** instruction does not complete until it is the oldest instruction presented to the on-chip memory subsystem. This occurs when all of the following conditions exist:

- The **tlbsync** instruction is the oldest instruction in the store queue
- The instruction and data cache reload tables are idle
- There are no outstanding table search operations (note that a table search operation for a VTQ-initiated access may have been cancelled as described above)

Figure 5-20 shows the flow of events caused by execution of the **tlbsync** instruction as well as the actions taken by the MPC7450 when a TLBSYNC transaction is detected on the processor bus.

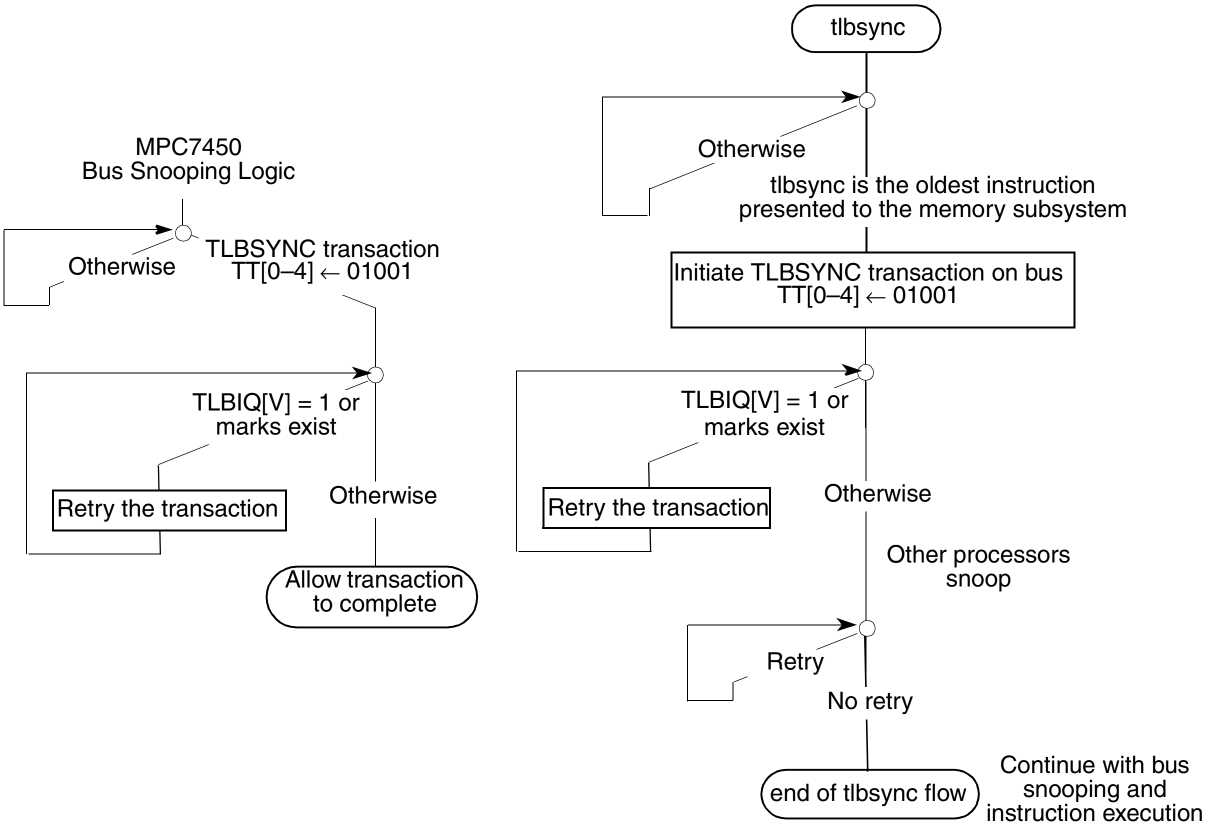


Figure 5-20. tlbsync Instruction Execution and Bus Snooping Flow

When an MPC7450 processor detects a TLBSYNC broadcast transaction, it causes a retry of that transaction until all pending TLB invalidate operations have completed. In this snoop process, the MPC7450 checks its TLBIQ and any pending marks for previously translated addresses. If the queue is valid or if any marks exist, the TLBSYNC transaction is retried, until the queue is invalid (idle) and no marks exist.

5.4.4.2.3 Synchronization Requirements for tlbie and tlbsync

In order to guarantee that a particular MPC7450 processor executing a **tlbie** instruction has completed the operation, a **sync** instruction must be placed after the **tlbie** instruction. A **tlbsync** instruction can also be used instead of the **sync** instruction for this purpose, but a **sync** will suffice for that processor. However, in order to guarantee that all MPC7450 processors in a system have coherently invalidated their respective TLB entries due to a **tlbie** instruction executing on any one of those processors, a **tlbsync** instruction is required.

The PowerPC architecture requires that when a **tlbsync** instruction has been executed by a processor, a **sync** instruction must be executed by that processor before a **tlbie** or **tlbsync** instruction is executed by another processor. If this requirement is not met, a livelock situation may occur in a system with multiple MPC7450 processors. Specifically, if more than one processor executes **tlbie** or **tlbsync** instructions simultaneously, it is likely that these processors will cause a system livelock.

5.4.5 Page Address Translation Summary—Extended Addressing

A detailed description of page address translation for a 32-bit physical address is provided in the section, “Page Address Translation Summary,” of Chapter 7, “Memory Management,” in the *Programming Environments Manual*. The following section highlights the differences for 36-bit physical addressing.

Figure 5-21 provides the detailed flow for the page address translation mechanism when using extended addressing.

When an instruction or data access occurs, the effective address is routed to the appropriate MMU. EA0–EA3 select 1 of the 16 segment registers, and the remaining effective address bits and the VSID field from the segment register are passed to the TLB. EA[14–19] then select two entries in the TLB; the valid bits are checked, and the 40-bit virtual page number (24-bit VSID concatenated with EA[4–19]) must match the VSID, EAPI, and API fields of the TLB entries. If one of the entries hits, the PP bits are checked for a protection violation. If these bits do not cause an exception, the C bit is checked. If the C bit must be updated, a table search operation is initiated. If the C bit does not require updating, the RPN value with the XPN and X extensions is passed to the memory subsystem and the WIMG bits are then used as attributes for the access.

Figure 5-21 includes the checking of the N bit in the segment descriptor and then expands on the ‘TLB Hit’ branch of Figure 5-8. The detailed flow for the ‘TLB Miss’ branch of Figure 5-8 is described in Section 5.5.2, “Page Table Search Operations—Implementation.” Note that as in the case of block address translation, if an attempt is made to execute a **dcbz** instruction to a page marked either write-through or caching-inhibited (W = 1 or I = 1), an alignment exception is generated. The checking of memory protection violation conditions is described in Chapter 7, “Memory Management,” in the *Programming Environments Manual*.

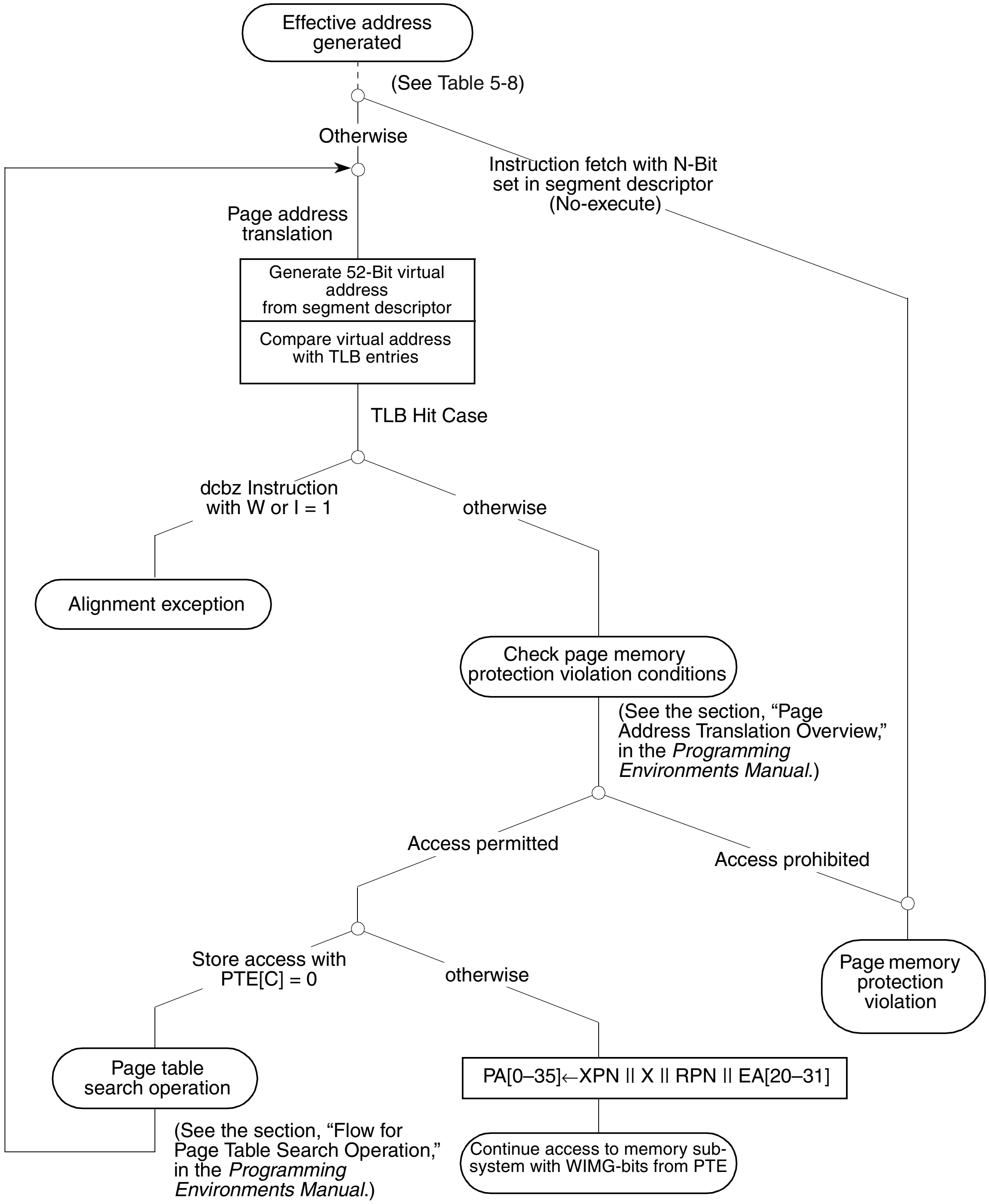


Figure 5-21. Page Address Translation Flow—TLB Hit—Extended Addressing

5.5 Hashed Page Tables—Extended Addressing

If a copy of the PTE corresponding to the VPN for an access is not resident in a TLB (corresponding to a miss in the TLB, provided a TLB is implemented), the processor must search (in hardware or software) for the PTE in the page tables set up by the operating system in main memory.

The algorithm specified by the architecture for accessing the page tables in hardware includes a hashing function on some of the virtual address bits. Thus the addresses for PTEs are allocated more evenly within the page tables and the hit rate of the page tables is maximized. This algorithm must be synthesized by the operating system for it to correctly place the page table entries in main memory.

When page table search operations are performed automatically by the hardware, they are performed using physical addresses and as if the memory access attribute bit M = 1 (memory coherency enforced in hardware). If the software performs the page table search operations, the accesses must be performed in real addressing mode (MSR[DR] = 0); this additionally guarantees that M = 1.

The section, “Hashed Page Tables,” in the *Programming Environments Manual* describes the format of the page tables and the algorithm used to access them for a 32-bit physical address. Section 5.4.1.2, “Page Table Entry (PTE) Definition—Extended Addressing,” describes the PTE format for extended addressing. The following subsections highlight the differences when translating for 36-bit physical addresses. In addition, the constraints imposed on the software in updating the page tables and the software table searching exception handlers (and other MMU resources) are described.

5.5.1 SDR1 Register Definition—Extended Addressing

The SDR1 register definition for 32-bit physical addressing is as described in Chapter 7, “Memory Management,” in the *Programming Environments Manual*. The SDR1 register contains the control information for the page table structure in that it defines the high-order bits for the physical base address of the page table and it defines the size of the table. Note that there are certain synchronization requirements for writing to SDR1 that are described in the section, “Synchronization Requirements for Special Registers and for Lookaside Buffers,” in the *Programming Environments Manual*. The format of the SDR1 register for extended addressing is described in the following sections. The SDR1 register has been modified for the MPC7450 to support extended 36-bit physical addresses (for when HID0[XAEN = 1]).

Figure 5-22 shows the format of the SDR1 register in the bottom half of the figure; the top half shows how the physical address generated corresponds to SDR1 fields.

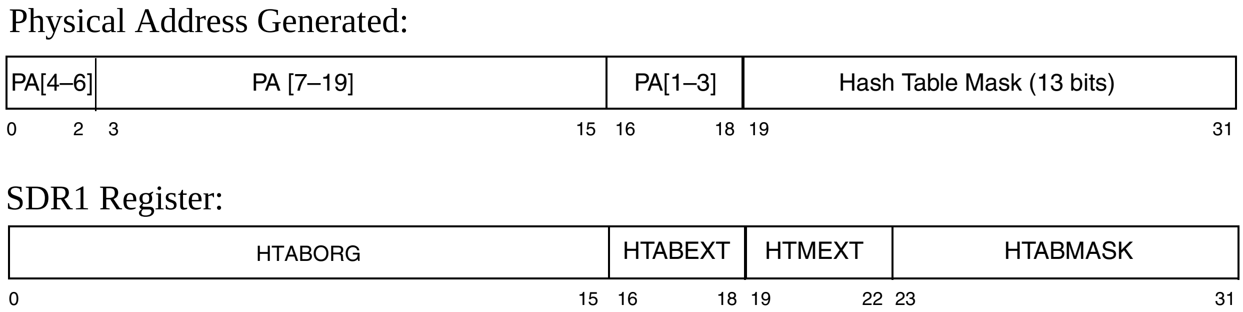


Figure 5-22. SDR1 Register Format—Extended Addressing

Bit settings for the SDR1 register are described in Table 5-13.

Table 5-13. SDR1 Register Bit Settings—Extended Addressing

Bits	Name	Description
0–15	HTABORG	Physical base address of page table If HID0[XAEN] = 1, field contains physical address [4–19] If HID0[XAEN] = 0, field contains physical address [0–15]
16–18	HTABEXT[1]	Extension bits for physical base address of page table If HID0[XAEN] = 1, field contains physical address [1–3] (and PA0 = 0) If HID0[XAEN] = 0, field is reserved
19–22	HTMEXT [1]	Hash table mask extension bits If HID0[XAEN] = 1, field contains hash table mask [0–3] If HID0[XAEN] = 0, field is reserved
23–31	HTABMASK	Mask for page table address If HID0[XAEN] = 1, field contains hash table mask [4–12] If HID0[XAEN] = 0, field contains hash table mask [0–7]

[1] MPC7441-/MPC7451-,MPC7445-/MPC7455-, MPC7447-/MPC7457-specific bits.

SDR1 can be accessed with **mtspr** and **mfspr** using SPR 25. For synchronization requirements on the register see Section 2.4.2.4, “Synchronization.”

When extended addressing is disabled (HID0[XAEN] = 0), then the SDR1[HTABORG] field contains the high-order 16 bits of the 32-bit physical address of the page table. That is, SDR1[0–15] comprise the physical base address of the page table. Therefore, the beginning of the page table lies on a 2^{16} byte (64 Kbyte) boundary at a minimum. If extended addressing is enabled (HID0[XAEN] = 1), then a leading zero is concatenated with the values in the SDR1[HTABEXT] and SDR1[HTABORG] fields to produce the physical base address of the page table. In this case, the beginning of the page table lies on a 2^{19} (512 Kbyte) boundary at a minimum.

When extended addressing is enabled, a page table can be any size 2^n bytes where $16 \leq n \leq 29$. The HTMEXT field concatenated with the HTABMASK field in SDR1 contains a mask value that determines how many bits from the output of the hashing function are used as the page table index. This mask must be of the form 0b00...011...1 (a string of 0 bits followed by a string of 1 bits). As the table size increases, more bits are used from the output of the hashing function to index into the table. The 1 bits in HTMEXT || HTABMASK determine how many additional bits (beyond the minimum of 10) from the hash are used in the index; the HTABORG field must have the same number of low-order bits equal to 0 as the HTMEXT || HTABMASK fields have low-order bits equal to 1.

The SDR1[HTABEXT] field is ignored when extended addressing is disabled (HID0[XAEN] = 0). If extended addressing is enabled (HID0[XAEN] = 1), then the SDR1[HTABEXT] field contains bits 1-3 of the physical address of the page table. Note that bit 0 of the physical address of the page table is always 0.

5.5.1.1 Page Table Size

The number of entries in the page table directly affects performance because it influences the hit ratio in the page table and thus the rate of page fault exception conditions. If the table is too small, not all virtual pages that have physical page frames assigned may be mapped via the page table. This can happen if more than 16 entries map to the same primary/secondary pair of PTEGs; in this case, many hash collisions may occur.

In a 32-bit implementation, the minimum size for a page table is 64 Kbytes (2^{10} PTEGs of 64 bytes each). However, it is recommended that the total number of PTEGs in the page table be at least half the number of physical page frames to be mapped. While avoidance of hash collisions cannot be guaranteed for any size page table, making the page table larger than the recommended minimum size reduces the frequency of such collisions by making the primary PTEGs more sparsely populated, and further reducing the need to use the secondary PTEGs.

Table 5-14 shows some example sizes for total main memory with the MPC7450 using extended addressing. The recommended minimum page table size for these example memory sizes are then outlined, along with their corresponding HTABORG, HTMEXT, and HTABMASK settings in SDR1. Note that systems with less than 8 Mbytes of main memory may be designed with 32-bit processors, but the minimum amount of memory that can be used for the page tables in these cases is 64 Kbytes.

Table 5-14. Minimum Recommended Page Table Sizes—Extended Addressing

Total Main Memory	Recommended Minimum			Settings for Recommended Minimum	
	Memory for Page Tables	Number of Mapped Pages (PTEs)	Number of PTEGs	HTABORG (Maskable Bits 3–15)	HTMEXT || HTABMASK SDR1[19-31]
8 Mbytes (2^{23})	64 Kbytes (2^{16})	2^{13}	2^{10}	x xxxx xxxx xxxx	0 0000 0000 0000
16 Mbytes (2^{24})	128 Kbytes (2^{17})	2^{14}	2^{11}	x xxxx xxxx xxx0	0 0000 0000 0001
32 Mbytes (2^{25})	256 Kbytes (2^{18})	2^{15}	2^{12}	x xxxx xxxx xx00	0 0000 0000 0011
64 Mbytes (2^{26})	512 Kbytes (2^{19})	2^{16}	2^{13}	x xxxx xxxx x000	0 0000 0000 0111
128 Mbytes (2^{27})	1 Mbyte (2^{20})	2^{17}	2^{14}	x xxxx xxxx 0000	0 0000 0000 1111
256 Mbytes (2^{28})	2 Mbytes (2^{21})	2^{18}	2^{15}	x xxxx xxx0 0000	0 0000 0001 1111
512 Mbytes (2^{29})	4 Mbytes (2^{22})	2^{19}	2^{16}	x xxxx xx00 0000	0 0000 0011 1111

Table 5-14. Minimum Recommended Page Table Sizes—Extended Addressing (continued)

Total Main Memory	Recommended Minimum			Settings for Recommended Minimum	
	Memory for Page Tables	Number of Mapped Pages (PTEs)	Number of PTEGs	HTABORG (Maskable Bits 3–15)	HTMEXT || HTABMASK SDR1[19-31]
1 Gbytes (2^{30})	8 Mbytes (2^{23})	2^{20}	2^{17}	x xxxx x000 0000	0 0000 0111 1111
2 Gbytes (2^{31})	16 Mbytes (2^{24})	2^{21}	2^{18}	x xxxx 0000 0000	0 0000 1111 1111
4 Gbytes (2^{32})	32 Mbytes (2^{25})	2^{22}	2^{19}	x xxx0 0000 0000	0 0001 1111 1111
8 Gbytes (2^{33})	64 Mbytes (2^{26})	2^{23}	2^{20}	x xx00 0000 0000	0 0011 1111 1111
16 Gbytes (2^{34})	128 Mbytes (2^{27}	2^{24}	2^{21}	x x000 0000 0000	0 0111 1111 1111
32 Gbytes (2^{35})	256 Mbytes (2^{28})	2^{25}	2^{22}	x 0000 0000 0000	0 1111 1111 1111
64 Gbytes (2^{36})	512 Mbytes (2^{29})	2^{26}	2^{23}	0 0000 0000 0000	1 1111 1111 1111

As an example, if the physical memory size is 2^{35} bytes (32 Gbyte), then there are $2^{35} – 2^{12}$ (4 Kbyte page size) = 2^{23} (8 Mbyte) total page frames. If this number of page frames is divided by 2, the resultant minimum recommended page table size is 2^{22} PTEGs, or 2^{28} bytes (256 Mbytes) of memory for the page tables.

5.5.1.2 Page Table Hashing Functions

The MMU uses two different hashing functions, a primary and a secondary, in the creation of the physical addresses used in a page table search operation. These hashing functions distribute the PTEs within the page table, in that there are two possible PTEGs where a given PTE can reside. Additionally, there are eight possible PTE locations within a PTEG where a given PTE can reside. If a PTE is not found using the primary hashing function, the secondary hashing function is performed, and the secondary PTEG is searched. Note that these two functions must also be used by the operating system to set up the page tables in memory appropriately.

The address of a PTEG is derived from the HTABORG field of the SDR1 register, and the output of the corresponding hashing function (primary hashing function for primary PTEG and secondary hashing function for a secondary PTEG). The values in the HTMEXT and HTABMASK fields determine how many of the high-order hash value bits are masked and how many are used in the generation of the physical address of the PTEG.

Figure 5-23 depicts the hashing functions used by the MPC7450 to generate a 36-bit physical table entry group address. The inputs to the primary hashing function are the low-order 23 bits of the VSID field of the selected segment register (VA[1–23]), and the page index field of the effective address (VA[24–39]) concatenated with seven zero high-order bits. The XOR of these two values generates the output of the primary hashing function (hash value 1).

When the secondary hashing function is required, the output of the primary hashing function is complemented with one’s complement arithmetic, to provide hash value 2.

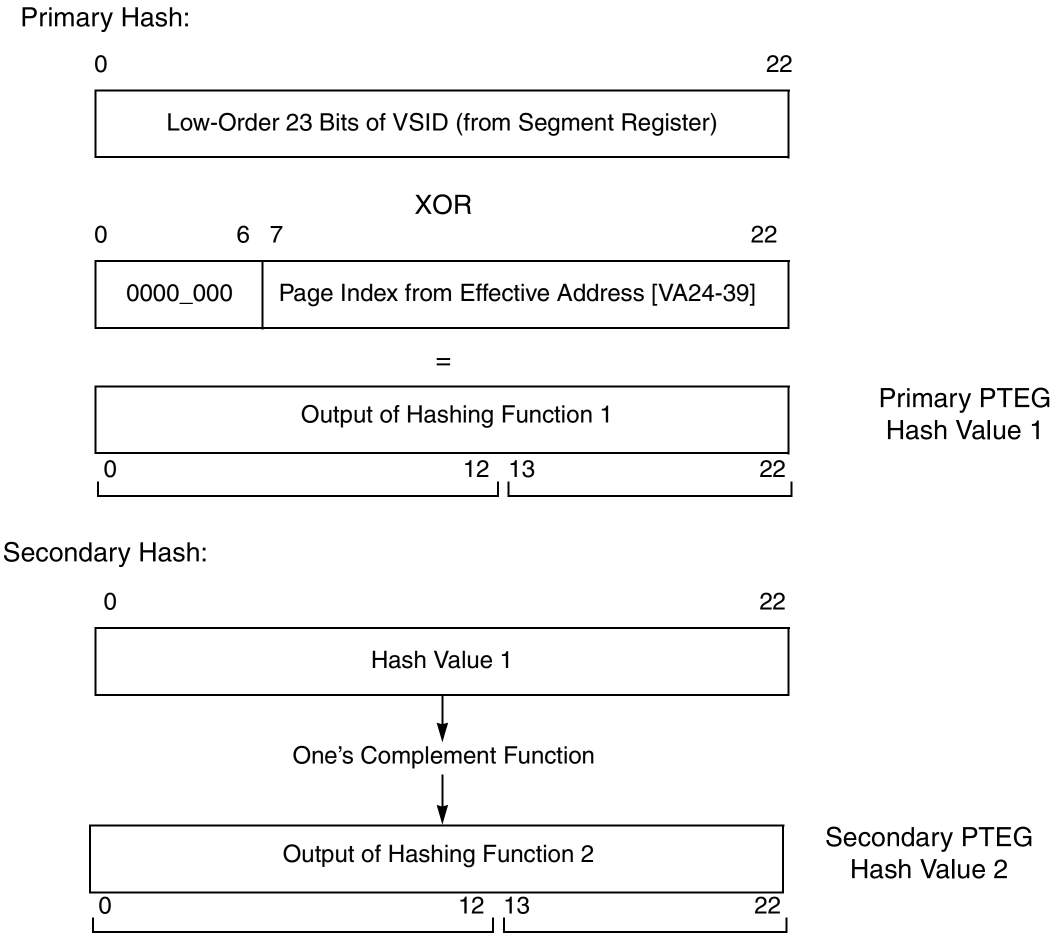


Figure 5-23. Hashing Functions for Page Table Entry Group Address

5.5.1.3 Page Table Address Generation

The following sections illustrate the generation of the addresses used for accessing the hashed page tables. As stated earlier, the operating system must synthesize the table search algorithm for setting up the tables. This process is as described in Chapter 7, “Memory Management,” in the *Programming Environments Manual*.

For extended addressing, PTEG[0] is zero and PTEG[1–3] is defined by the HTABEXT field of SDR1 (SDR1[16–18]) as shown in Figure 5-24. PTEG[4–6] is defined by the highest order bits of the HTABORG field (SDR1[0–2]). PTEG[7–19] are derived from the masking of the high-order bits of the hash value[0–12] with SDR1[HTABMASK] and SDR1[HTMEXT]. The value from the AND function is then concatenated with (implemented as an OR function) the high-order bits of the unmasked HTABORG bits SDR1[3–15]. PTEG[20–29] are the 10 low-order bits of the hash value. PTEG[30–35] are zeros. In the process of searching for a PTE, the processor checks up to eight PTEs located in the primary PTEG and up to eight PTEs located in the secondary PTEG, if required, searching for a match.

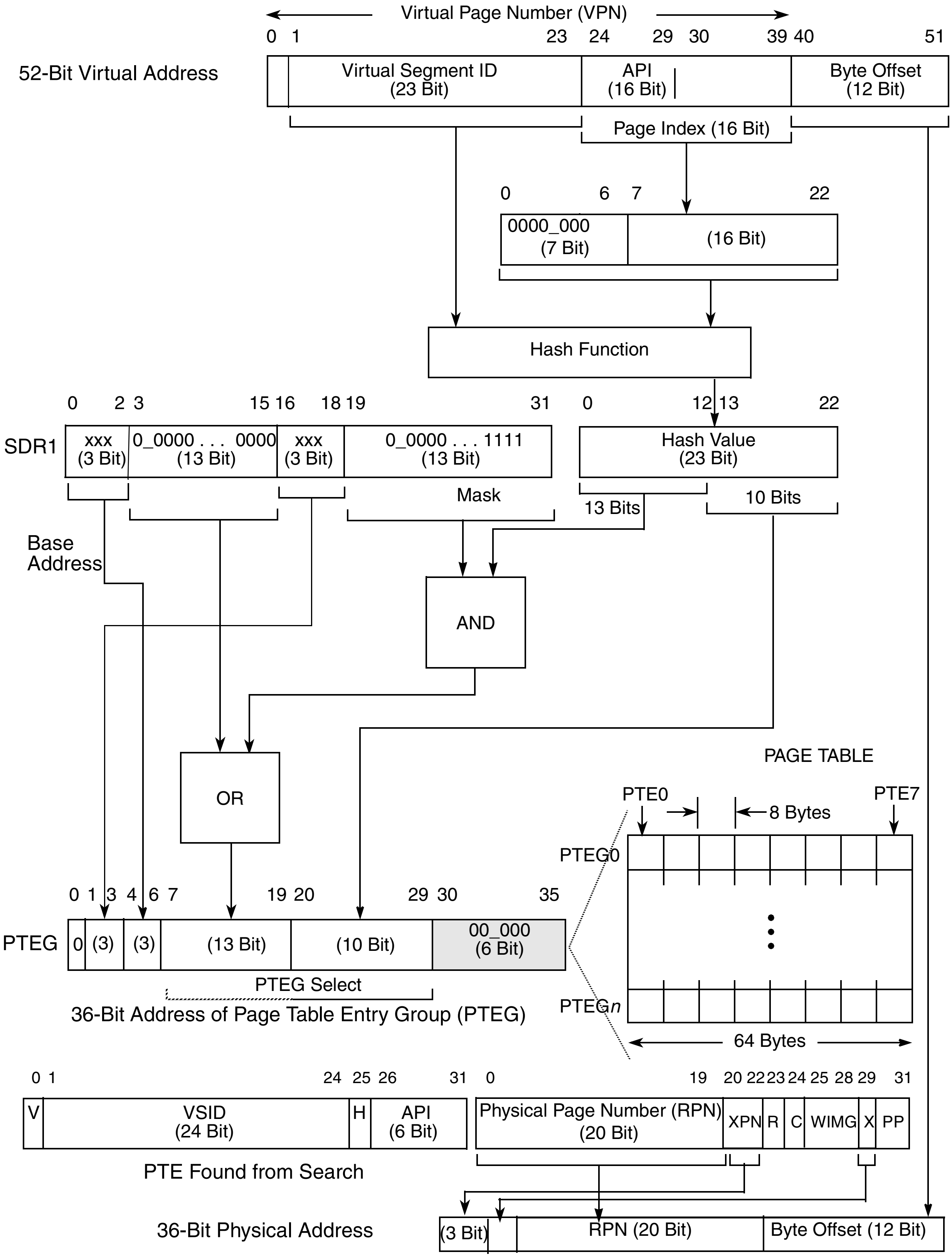


Figure 5-24. PTEG Address Generation for Page Table Search—Extended Addressing

5.5.1.4 Page Table Structure Example—Extended Addressing

Figure 5-25 shows the structure of an example page table. The base address of the page table is defined as shown in Figure 5-25. In this example, the address is identified by 0 || HTBEXT || HTABORG[0–13]; note that bits 14 and 15 of HTABORG must be zero because the low-order two bits of HTABMASK are ones. The addresses for individual PTEGs within this page table are then defined by bits 18–29 as an offset from bits 0–17 of this base address. Thus the size of the page table is defined as 4096 PTEGs.

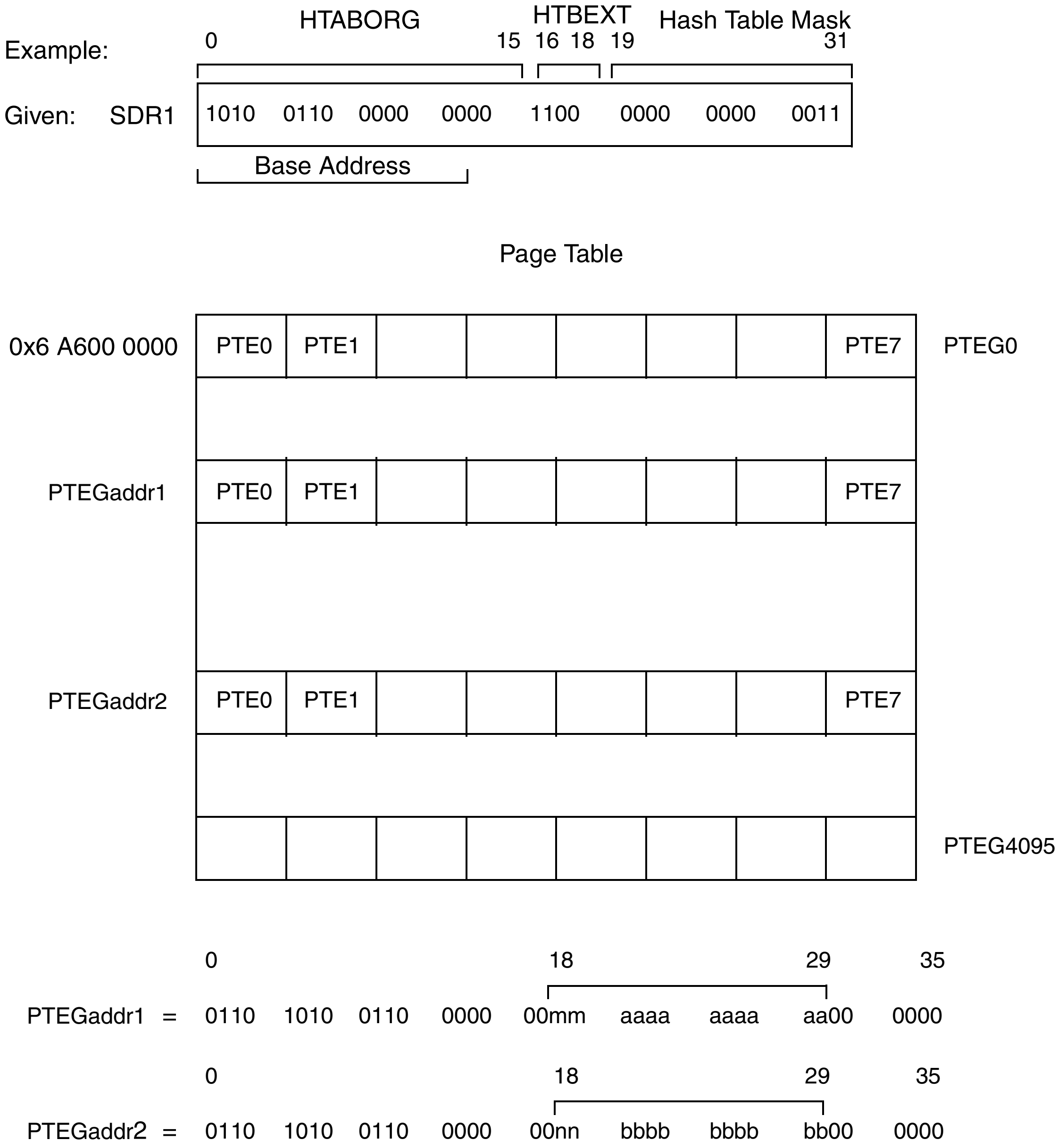


Figure 5-25. Example Page Table Structure—Extended Addressing

Two example PTEG addresses are shown in the figure as PTEGaddr1 and PTEGaddr2. Bits 18–29 of each PTEG address in this example page table are derived from the output of the hashing function (bits 30–35 are zero to start with PTE0 of the PTEG). In this example, the ‘b’ bits in PTEGaddr2 are the one’s complement of the ‘a’ bits in PTEGaddr1. The ‘n’ bits are also the one’s complement of the ‘m’ bits, but these two bits are generated from bits 11–12 of the output of the hashing function, logically ORed with bits 14–15 of the HTABORG field (that must be zero). If bits 18–29 of PTEGaddr1 were derived by using the primary hashing function, then PTEGaddr2 corresponds to the secondary PTEG.

Note, however, that bits 18–29 in PTEGaddr2 can also be derived from a combination of effective address bits, segment register bits, and the primary hashing function. In this case, PTEGaddr1 corresponds to the secondary PTEG. Thus while a PTEG may be considered a primary PTEG for some effective addresses (and segment register bits), it may also correspond to the secondary PTEG for a different effective address (and segment register value).

It is the value of the H bit in each of the individual PTEs that identifies a particular PTE as either primary or secondary (there may be PTEs that correspond to a primary PTEG and PTEs that correspond to a secondary PTEG, all within the same physical PTEG address space). Thus only the PTEs that have H = 0 are checked for a hit during a primary PTEG search. Likewise, only PTEs with H = 1 are checked in the case of a secondary PTEG search.

5.5.1.5 PTEG Address Mapping Examples—Extended Addressing

This section contains two examples of an effective address and how its address translation (the PTE) maps into the primary PTEG in physical memory. The examples illustrate how the processor generates PTEG addresses for a table search operation; this is also the algorithm that must be used by the operating system in creating page tables.

Figure 5-27 shows an example of PTEG address generation for extended addressing. In the example, the value in SDR1 defines a page table at address 0x4_0F98_0000 that contains 8192 PTEGs. The example effective address selects segment register 0 (SR0) with the highest-order four bits. The contents of SR0 are then used along with bits 4–31 of the effective address to create the 52-bit virtual address.

To generate the address of the primary PTEG, bits1–23, and bits 24–39 of the virtual address are then used as inputs into the primary hashing function (XOR) to generate hash value 1. The low-order 13 bits of hash value 1 are then concatenated with the high-order 13 bits of HTABORG and HTBEXT with an added leading zero. Finally the address is appended with six low-order 0 bits, defining the address of the primary PTEG (0x4_0F9F_F980).

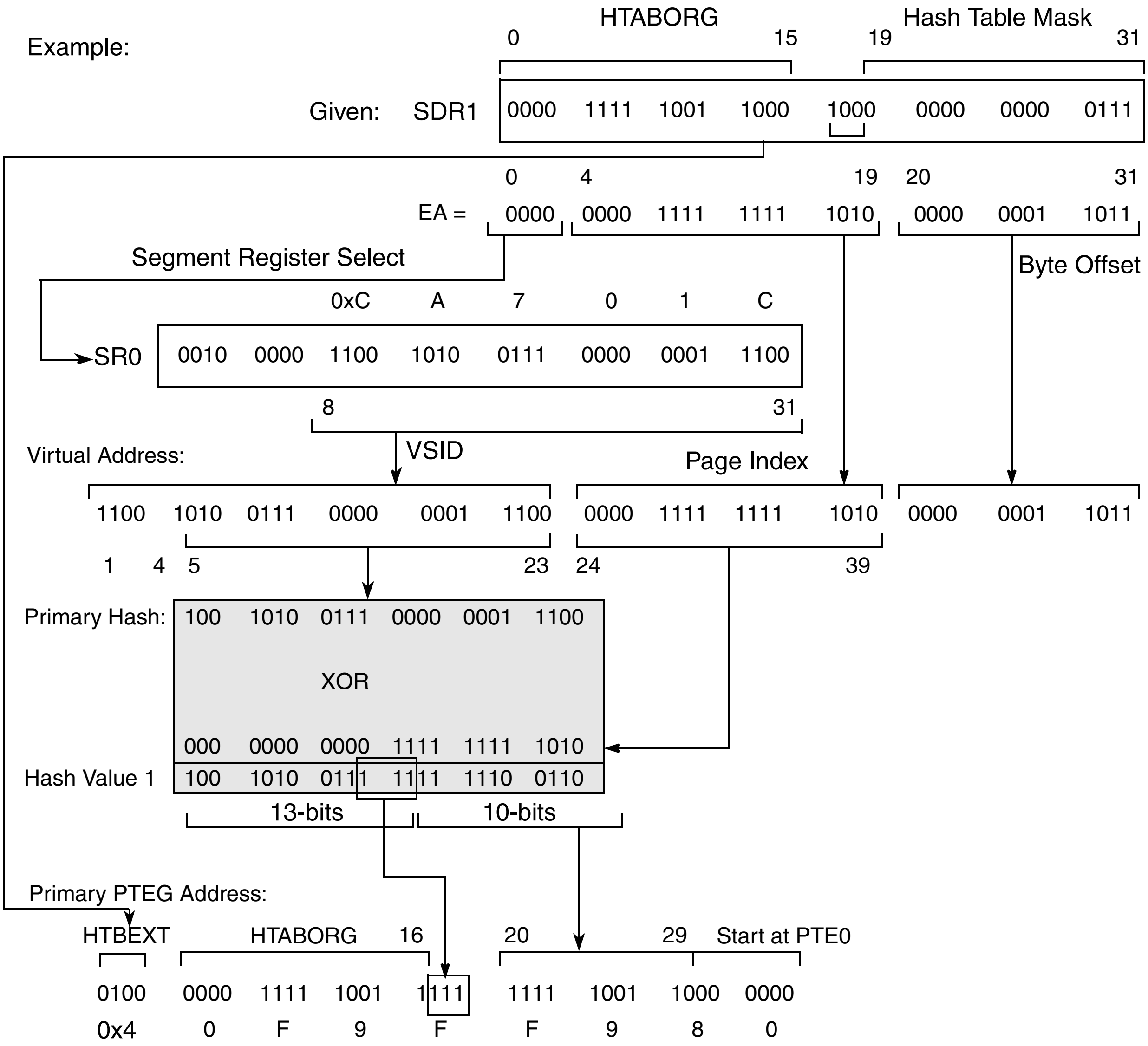


Figure 5-26. Example Primary PTEG Address Generation

Figure 5-27 shows the generation of the secondary PTEG address for this example. If the secondary PTEG is required, the secondary hash function is performed and the low-order 13 bits of hash value 2 are then ORed with the high-order 16 bits of HTABORG (bits 13–15 should be zero), and HTBEXT with an added leading zero. Finally, the address is appended with six low-order 0 bits, defining the address of the secondary PTEG (0x4_0F98_0640).

As described in Figure 5-24, the 10 low-order bits of the page index field are always used in the generation of a PTEG address (through the hashing function) for a 32-bit implementation. This is why only the abbreviated page index (API) is defined for a PTE (the entire page index field does not need to be checked). For a given effective address, the low-order 10 bits of the page index (at least) contribute to the PTEG address (both primary and secondary) where the corresponding PTE may reside in memory. Therefore, if the high-order 6 bits (the API field) of the page index match with the API field of a PTE within the specified PTEG, the PTE mapping is guaranteed to be the unique PTE required.

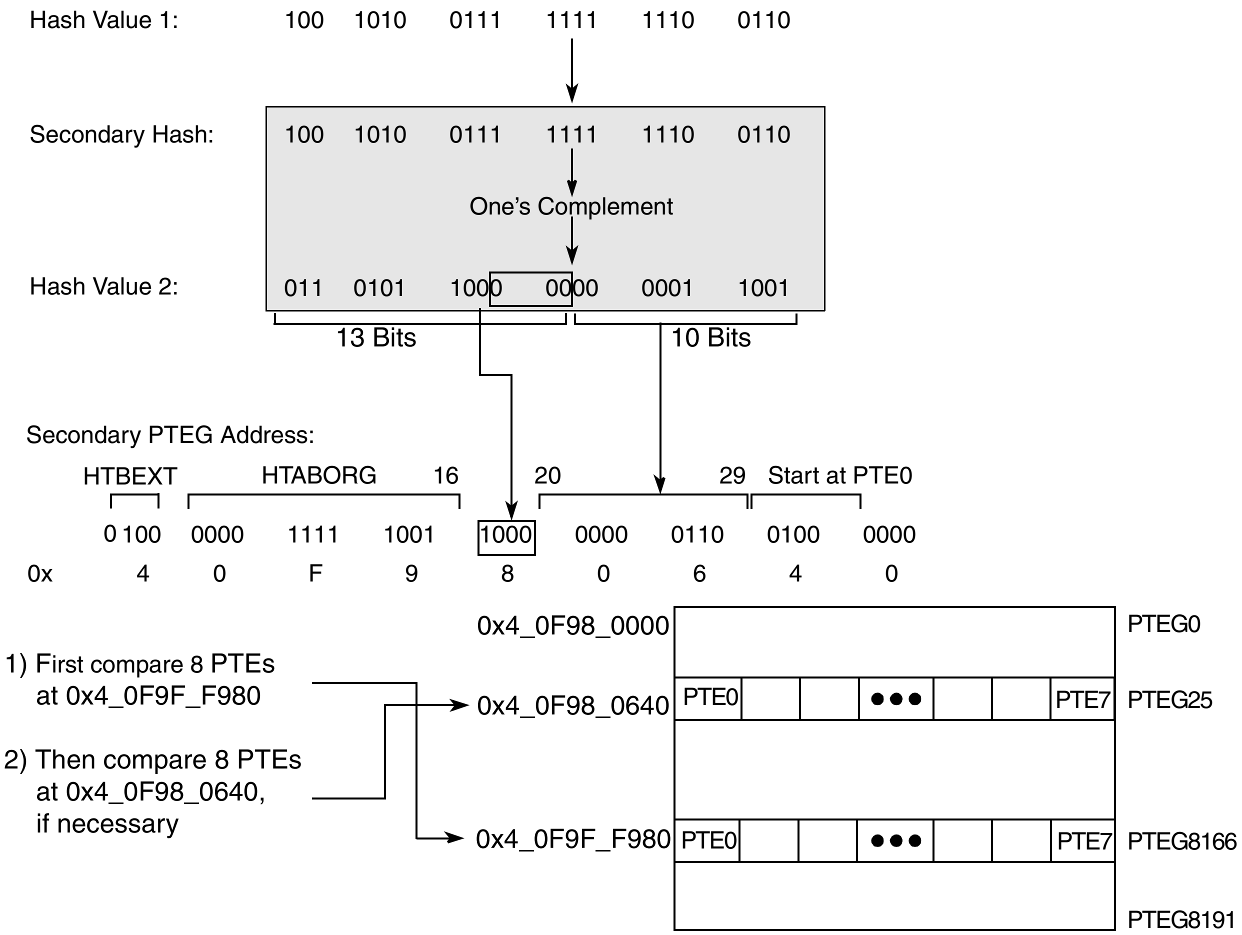


Figure 5-27. Example Secondary PTEG Address Generation

Note that a given PTEG address does not map back to a unique effective address. Not only can a given PTEG be considered both a primary and a secondary PTEG (as described in Section 5.5.1.4, "Page Table Structure Example—Extended Addressing"), but in this example, bits 24–26 of the page index field of the virtual address are not used to generate the PTEG address. Therefore, any of the eight combinations of these bits will map to the same primary PTEG address. (However, these bits are part of the API and are therefore compared for each PTE within the PTEG to determine if there is a hit.) Furthermore, an effective address can select a different segment register with a different value such that the output of the primary (or secondary) hashing function happens to equal the hash values shown in the example. Thus these effective addresses would also map to the same PTEG addresses shown.

5.5.2 Page Table Search Operations—Implementation

If the translation is not found in the TLBs (a TLB miss), the MPC7450 initiates a hardware or software table search operation as described in this section for 36-bit addressing. Formats for the PTEs used in 32-bit addressing are described in "PTE Format for 32-Bit Implementations," in Chapter 7, "Memory Management," of the *Programming Environments Manual.*

5.5.2.1 Conditions for a Page Table Search Operation

For instruction accesses, the MPC7450 processor does not initiate a table search operation for an ITLB miss until the completion buffer is empty and the completed store queue is empty. Also, the instruction buffer must be empty, there must be no other exceptions pending, there must be no branch processing in progress, and there must be no outstanding instruction cache misses.

Also, the MMU does not perform a hardware table search due to DTLB misses (or to modify the C bit) until the access is absolutely required by the program flow and there are no other exceptions pending.

In the MPC7450, a TLB miss (and subsequent page table search operation) occurs transparently to the program. Thus if a TLB miss occurs when a misaligned access crosses a translation boundary, the second portion of the misaligned access is completed automatically once the table search operation completes successfully. If the table search operation results in a page fault, an exception occurs and upon returning from the page fault handling routine, the entire misaligned access is restarted beginning with the first portion of the access.

Note that, as described in Chapter 6, “Instruction Timing,” store gathering does not occur while a page table search operation is in progress.

The AltiVec data stream touch instructions (**dst**[**t**] and **dstst**[**t**]) provide the ability to prefetch up to 128 Kbytes of data per instruction. As described in Chapter 6, “Instruction Timing,” a **dst**[**t**] or **dstst**[**t**] instruction can be retired from the completion buffer as soon as the instruction is loaded into the vector touch queue (VTQ). However, if a line fetch in the VTQ requires a table search operation before the instruction is retired, then the table search operation is delayed until the instruction is retired. If a line fetch in the VTQ requires a table search operation after the instruction has been retired, the table search operation is initiated immediately.

To further increase performance, the VTQ stream engines operate in parallel with the other execution units. Also, the TLBs are non-blocking, and are available to the instruction unit and LSU for both instruction and data address translation during a VTQ-initiated table search operation.

5.5.2.2 AltiVec Line Fetch Skipping

As described in Chapter 7, “AltiVec Technology Implementation,” there are many conditions (exceptions, etc.) that cause the stream fetch performed by a VTQ stream engine to abort. In the case of a VTQ-initiated table search operation, when an exception or interrupt condition occurs, the stream engine pauses, the line-fetch that caused the table search operation is effectively dropped, and no MMU exceptions are reported for this line-fetch. When the stream engine resumes operation, the next line fetch is attempted, causing a skip of one line fetch in the stream engine.

Also, when a **tlbsync** instruction is executed while a VTQ-initiated table search operation is in progress, that table search operation is aborted, potentially causing a line fetch skip.

5.5.2.3 Page Table Search Operation—Conceptual Flow

The following is a summary of the page table search process performed automatically by the MPC7450 when hardware table searching is enabled. A very similar flow occurs when the software table searching is enabled.

1. The 32-bit physical address of the primary PTEG is generated as described in Chapter 7, “Memory Management,” of the *Programming Environments Manual*. When extended addressing is enabled, the 36-bit address generation is described in Section 5.5.1.3, “Page Table Address Generation.”
2. The first PTE (PTE0) in the primary PTEG is read from memory. PTE reads occur with an implied WIM memory/cache mode control bit setting of 0b001. Therefore, they are considered cacheable and read (burst) from memory and placed in the cache. Because the table search operation is never speculative and is cacheable, the G-bit has no effect.
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 8 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 have a WIM bit combination of 0b001, an entire cache line is read into the on-chip cache. 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]
6. If a match is not found, step 6 is repeated for each of the other seven PTEs in the secondary PTEG. If it is never found, an exception is taken (step 9).
7. If a match is found, the PTE is written into the on-chip TLB and the R bit is updated in the PTE in memory (if necessary). If there is no memory protection violation, the C bit is also updated in memory (if the access is a write operation) and the table search is complete.
8. If a match is not found within the 8 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 MPC7450 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 hardware table search operations are performed as global (but not exclusive), cacheable operations, and can be loaded into the on-chip cache. These types of transactions should be generated when software table searching is enabled.

Figure 5-28 and Figure 5-29 show how the conceptual flow diagrams for the primary and secondary page table search operations, described in the section, “Page Table Search Operation,” in the *Programming Environments Manual*, are realized in the MPC7450. Recall that the architecture allows for implementations to perform the page table search operations automatically (in hardware), or software assistance may be allowed, as is an option with the MPC7450.

Figure 5-28 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 if memory protection is violated.

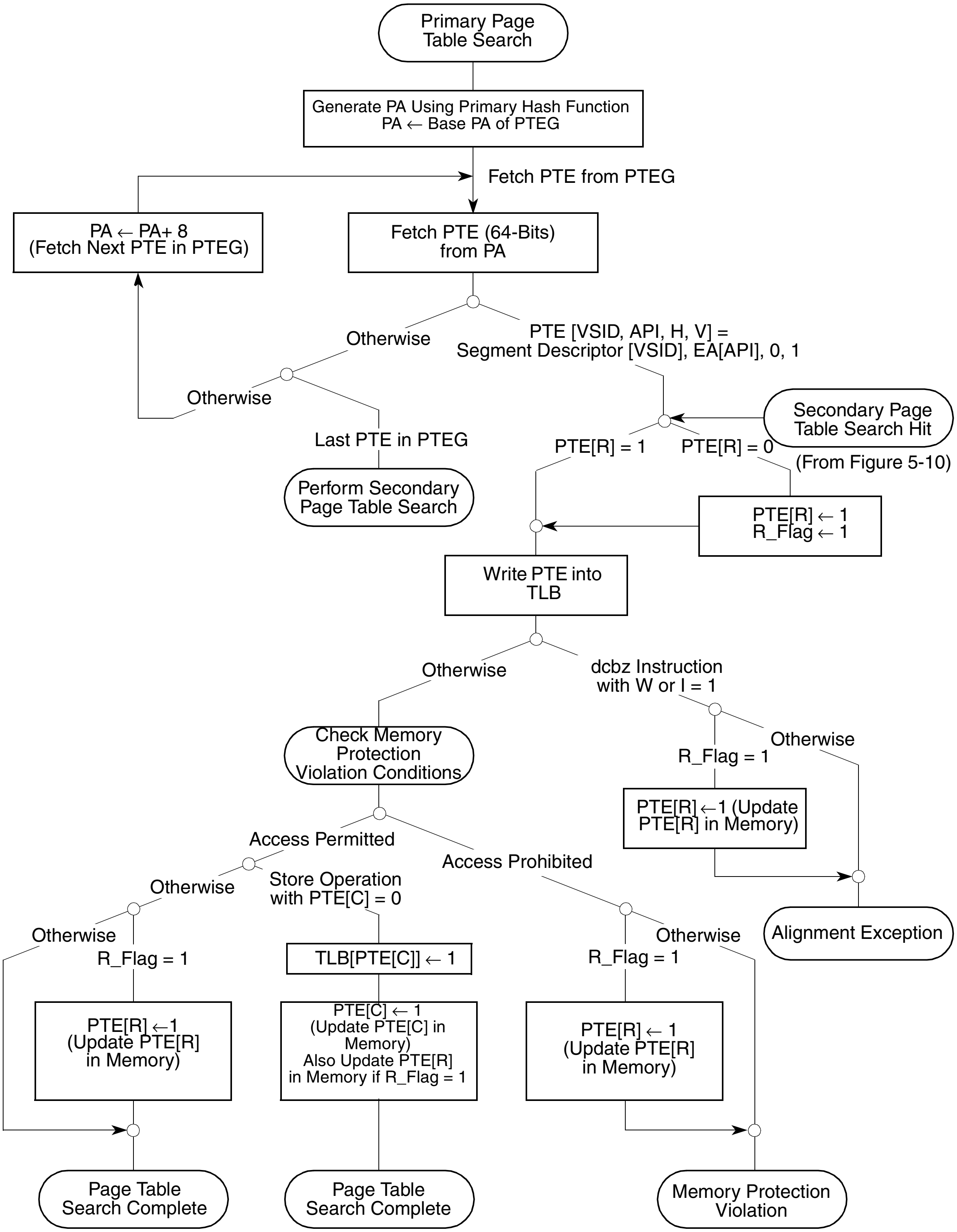


Figure 5-28. Primary Page Table Search—Conceptual Flow

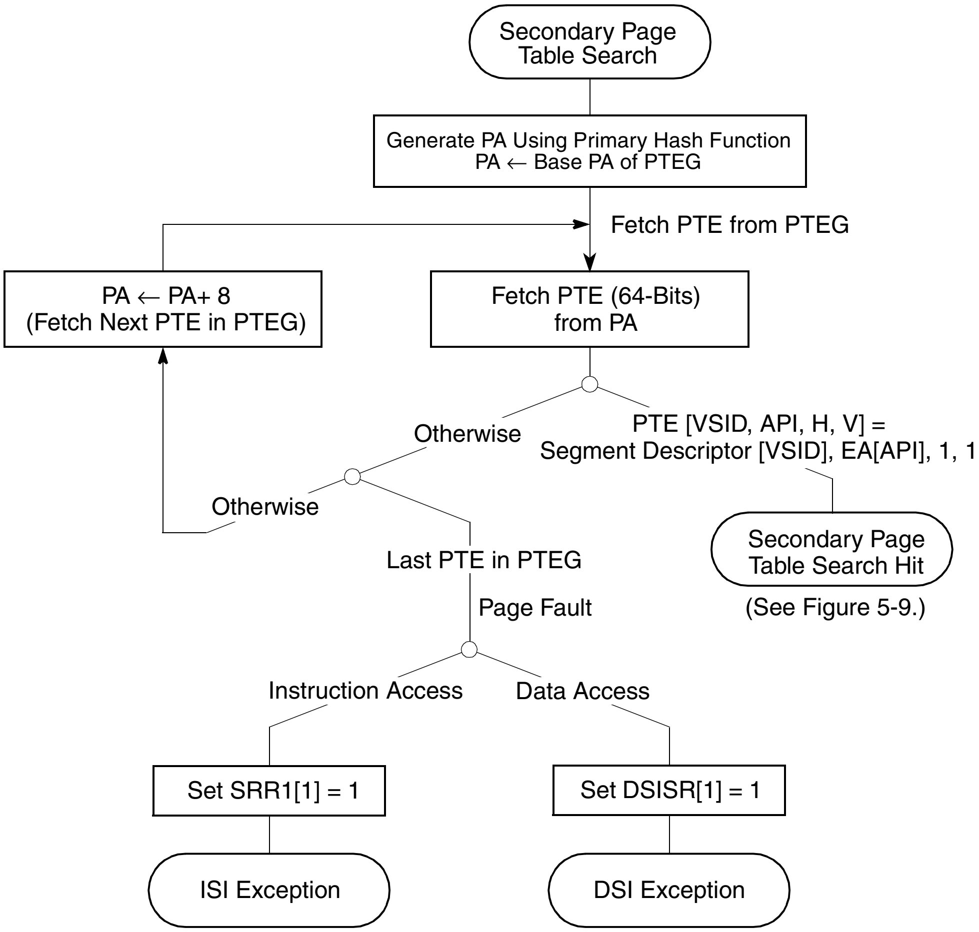


Figure 5-29. Secondary Page Table Search Flow—Conceptual Flow

5.5.3 Page Table Updates

When TLBs are implemented (as in the MPC7450) 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.

Chapter 7, "Memory Management," in the *Programming Environments Manual* describes some required sequences of instructions for modifying the page tables. In a multiprocessor MPC7450 environment, PTEs can only be modified by adhering to the procedure for deleting a PTE, followed by the procedure for adding a PTE.

Thus the following code should be used:

```
/* Code for Modifying a Page Table Entry */
/* First delete the current page table entry */
PTE_V <- 0/* (other fields don't matter) */
sync                /* ensure update completed */
tlbie(old_EA)       /* invalidate old translation */
eieio               /* order tlbie before tlbsync */
tlbsync             /* ensure tlbie completed on all processors */
sync                /* ensure tlbsync completed */
                    /* Then add new PTE over old */
PTE_RPN,R,C,WIMG,PP <- new values
eieio               /* order 1st PTE update before 2nd */
PTE_VSID,API,H,V <- new values (V=1)
sync               /* ensure updates completed */
```

Processors may write referenced and changed bits with unsynchronized, atomic byte store operations. Note that the V, R, and C bits each reside in a distinct byte of a PTE. Therefore, extreme care must be taken to use byte writes when updating only one of these bits.

Explicitly altering certain MSR bits (using the **mtmsr** instruction), or explicitly altering PTEs, or certain system registers, 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

Synchronization requirements for using the move to segment register instructions are described in “Synchronization Requirements for Special Registers and for Lookaside Buffers” in Chapter 2, “PowerPC Register Set,” in the *Programming Environments Manual*.

5.5.5 Implementation-Specific Software Table Search Operation

The MPC7540 has a set of implementation-specific registers, exceptions, and instructions that facilitate very efficient software searching of the page tables in memory for when software table searching is enabled (HID0[STEN] = 1). This section describes those resources and provides three example code sequences that can be used in a MPC7540 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 in this case—instruction TLB miss, data TLB miss on load, and data TLB miss on store exceptions.

5.5.5.1 Resources for Table Search Operations

When software table searching is enabled, the system software must set up the translation page tables in memory, and 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 5-15 provides a summary of the implementation-specific exceptions, registers, and instructions, that can be used by the TLB miss exception handler software in MPC7540 systems. Refer to Chapter 4, “Exceptions,” for more information about exception processing.

Table 5-15. Implementation-Specific Resources for Software Table Search Operations

Resource	Name	Description
Exceptions	ITLB miss exception (vector offset 0x1000)	No matching entry found in ITLB
	DTLB miss on load exception (vector offset 0x1100)	No matching entry found in DTLB for a load data access
	DTLB 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	TLBMISS	When either an instruction TLB miss, data TLB miss on load, and data TLB miss on store exception occurs, the TLBMISS register contains part of the effective address of the instruction or data access that caused the miss exception.
	PTEHI	When software table searching is enabled (HID0[STEN] = 1), and a TLB miss exception occurs, the fields of the PTEHI register are loaded automatically with the VSID information from the corresponding SR, and the API of the miss address. The PTEHI register is also used by the **tlbli** and **tlbld** instructions.
	PTELO	When software table searching is enabled (HID0[STEN] = 1), and a TLB miss exception occurs, software determines the lower 32 bits of the PTE and places those bits in the PTELO register. The PTELO register is also used by the **tlbli** and **tlbld** instructions.
	SPRG4–7[1]	For the MPC7445, MPC7447, MPC7447A, MPC7448, MPC7455, and the MPC7457, when software table searching is enabled (HID0[STEN] = 1), and a TLB miss exception occurs, the SPRGs provide additional registers to be used by system software for table software searching.
Instructions	**tlbli r**B	Loads the contents of the PTEHI and PTELO registers into the ITLB entry selected by <EA> where <EA> = bits 10–19 of **r**B. Way to be loaded is selected by **r**B[31] (LRU way bit).
	tlbld rB	Loads the contents of the PTEHI and PTELO registers into the DTLB entry selected by <EA> where <EA> = bits 10–19 of **r**B. Way to be loaded is selected by **r**B[31] (LRU way bit).

[1] Specific only to the MPC7445, MPC7447, MPC7447A, MPC7448, MPC7455, and MPC7457 registers.

In addition, the MPC7540 contains the following features that do not specifically control the MPC7540 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:

- TLBMISS[31] 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. In the case of a store hit with C = 0, TLBMISS[31] points to the way that missed on the store access (and not the entry that hit with C = 0). Therefore, software must toggle this bit before placing it into **r**B[31]. Then **tlbld r**B is executed by software, updating the entry that originally hit with C = 0.
- 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 5-16 summarizes the SRR1 bits updated whenever one of the three TLB miss exceptions occurs.

Table 5-16. Implementation-Specific SRR1 Bits

Bit Number	Name	Function
11	CEQ0	Set if the exception was caused by the a store to a page with PTE[C] = 0.
12	KEY	Key for TLB miss (either SR[Ks] or SR[Kp] from the segment register, depending on whether the access is a supervisor or user access)

The key bit saved in SRR1 is derived as shown in Figure 5-30.

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

Figure 5-30. 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 that 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 TLBMISS, PTEHI, and PTELO 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.

Software must ensure that a TLB lookup never results in a match on both ways of the same set. It is a programming error for multiple ways to match and it can produce unpredictable results. Software is required to keep track of the current contents of the TLBs.

In a multiprocessing system, software must take steps to ensure coherency during a software table search operation. If a processor executes a **tlbie** instruction while another processor is handling a software table search exception, coherency can be lost and the TLB could be corrupted. A semaphore mechanism should be used when performing a software table search operation in a multiprocessing environment to ensure that coherency is maintained.

5.5.5.1.1 TLB Miss Register (TLBMISS)

The TLBMISS register is automatically loaded by the MPC7450 when software searching is enabled (HID0[XAEN] = 1) and a TLB miss exception occurs. Its contents are used by the TLB miss exception handlers (the software table search routines) to start the search process. Note that the MPC7450 always loads a big-endian address into the TLBMISS register. This register is read-only. The TLBMISS register has the format shown in Figure 5-31.

Figure 5-31. TLBMISS Register

Table 5-17 described the bits in the TLBMISS register.

Table 5-17. TLBMISS Register—Field and Bit Descriptions

Bit Number	Name	Function
0–30	PAGE	Effective page address. Stores EA[0–30] of the access that caused the TLB Miss exception.
31	LRU	Least recently used way of the addressed TLB set. The LRU bit can be loaded into bit 31 of **r**B, prior to execution of **tlbli** or **tlbld** to select the way to be replaced for a TLB miss. However, this value should be inverted in **r**B prior to execution of **tlbli** or **tlbld** for a TLB miss exception caused by the need to update the C-bit.

5.5.5.1.2 Page Table Entry Registers (PTEHI and PTELO)

The PTEHI and PTELO registers are used by the **tlbld** and **tlbli** instructions to create a TLB entry when extended addressing is enabled (HID0[XAEN] = 1). When software table searching is enabled (HID0[STEN] = 1), and a TLB miss exception occurs, the bits of the page table entry (PTE) for this access are located by software and saved in the PTE registers. Figure 5-32 shows the format for two supervisor registers PTEHI and PTELO, respectively.

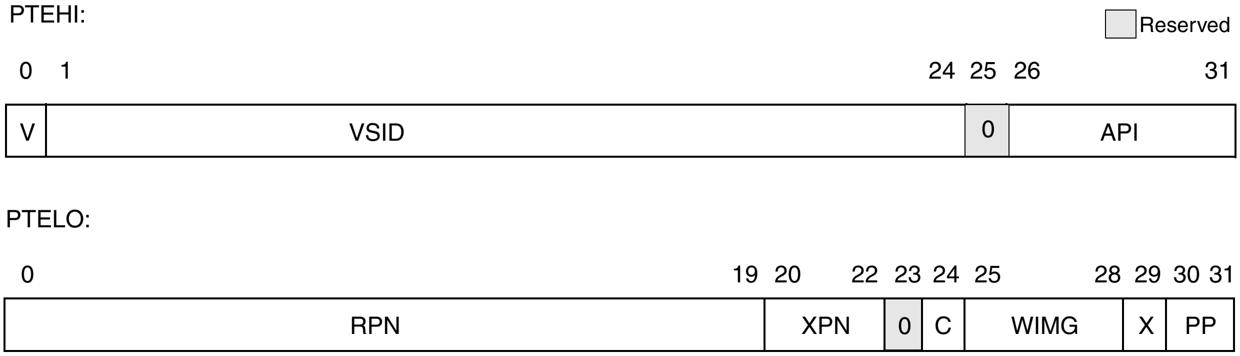


Figure 5-32. PTEHI and PTELO Registers—Extended Addressing

Note that the contents of PTEHI are automatically loaded when any of the three software table search exceptions is taken. PTELO is loaded by the software table search routines (the TLB miss exception handlers) based on the valid PTE located in the page tables prior to execution of **tlbli** or **tlbld** instruction.

Table 5-18 lists the corresponding bit definitions for the PTEHI and PTELO registers.

Table 5-18. PTEHI and PTELO Bit Definitions

Word	Bit	Name	Description
PTEHI	0	V	Entry valid (V = 1) or invalid (V = 0). Always set by the processor on a TLB miss exception.
	1–24	VSID	Virtual segment ID. The corresponding SR[VSID] field is copied to this field.
	25	—	Reserved
	26–31	API	Abbreviated page index. TLB miss exceptions will set this field with bits from TLBMISS[4–9] which are bits from the effective address for the access that caused the software table search operation. The **tlbld** and **tlbli** instructions will ignore the API bits in PTEHI register and get the API from instruction's operand**, r**B. However, for future compatibility, the API in **r**B should match the PTEHI[API].
PTELO	0–19	RPN	Physical page number
	20–22	XPN	Extended page number. The XPN field provides the physical address bits, PA[0–2].
	23	—	Reserved
	24	C	Changed bit
	25–28	WIMG	Memory/cache control bits
	29	X	Extended page number. The X field provides the physical address bit 3, PA[3].
	30–31	PP	Page protection bits

Note that PTELO[23] corresponds to the reference bit in a PTE. The reference bit is not stored in the page tables, so this bit is ignored in the PTELO register. All the other bits in PTELO correspond to the bits in the low word of the PTE. When extended addressing is not enabled, (HID0[XAEN] = 0), the PTELO[XPI] and PTELO[X] values should be zeros so that the four most significant bits of the physical address are zeros.

5.5.5.1.3 Special Purpose Registers (4–7)

Four additional SPRGs are provided on the MPC7445, MPC7447, MPC7447A, MPC7448, MPC7455, and the MPC7457. The registers are provided to assist in a software table search. For example, in the example code in Section 5.5.5.2.2, "Code for Example Exception Handlers," the register values are saved into the SPRGs to avoid any latency in storing the values out to memory. Thus using the additional SPRGs made the code faster and simpler.

5.5.5.2 Example Software Table Search Operation

When a TLB miss occurs, the instruction or data MMU loads the TLBMISS register with the effective address (EA[0–30]) 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.

The software uses whatever routine it implemented to generate the PTE. Then it places the upper and lower portions of the PTE into PTEHI and PTELO, respectively. Then it uses the **tlbli** or **tlbld** instructions to load the contents of the PTE into the selected TLB entry. The TLB entry is selected by bits 10–19 of **r**B and the way is selected by bit 31 of **r**B.

Note that a miss caused by a **dcbt**, **dst**, or **dstst** instruction while HID0[STEN] = 1 does not cause one of the software table searching exceptions; the **dcbt** in this case functions as a no-op and **dst**/**dstst** cause the stream to terminate.

If the PTE search algorithm does not produce a desired PTE, 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, that handles the page fault condition.

This section provides a flow diagram outlining an example software algorithm that mimics the hardware table search procedure used by the MPC7450 (and other processors that implement the PowerPC architecture). This software can be used to handle the three TLB miss exceptions. Some example assembly language that implements that flow is also provided. However, software can implement other types of page tables and PTE search algorithms using the same resources.

5.5.5.2.1 Flow for Example Exception Handlers

Figure 5-33 shows the flow for the example TLB miss exception handlers. Figure 5-34 shows the flow for how a PTEG address is generated. The flow shown is common for the three exception handlers. Also, in 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. Note that 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.

Figure 5-35 shows the flow for checking the R and C bits and setting them appropriately, Figure 5-36 shows the flow for synthesizing a page fault exception when no PTE is found. Figure 5-37 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 setup for these protection violation exceptions is very similar to that of page fault conditions (as shown in Figure 5-36) except that different bits in SRR1 (and DSISR) are set.

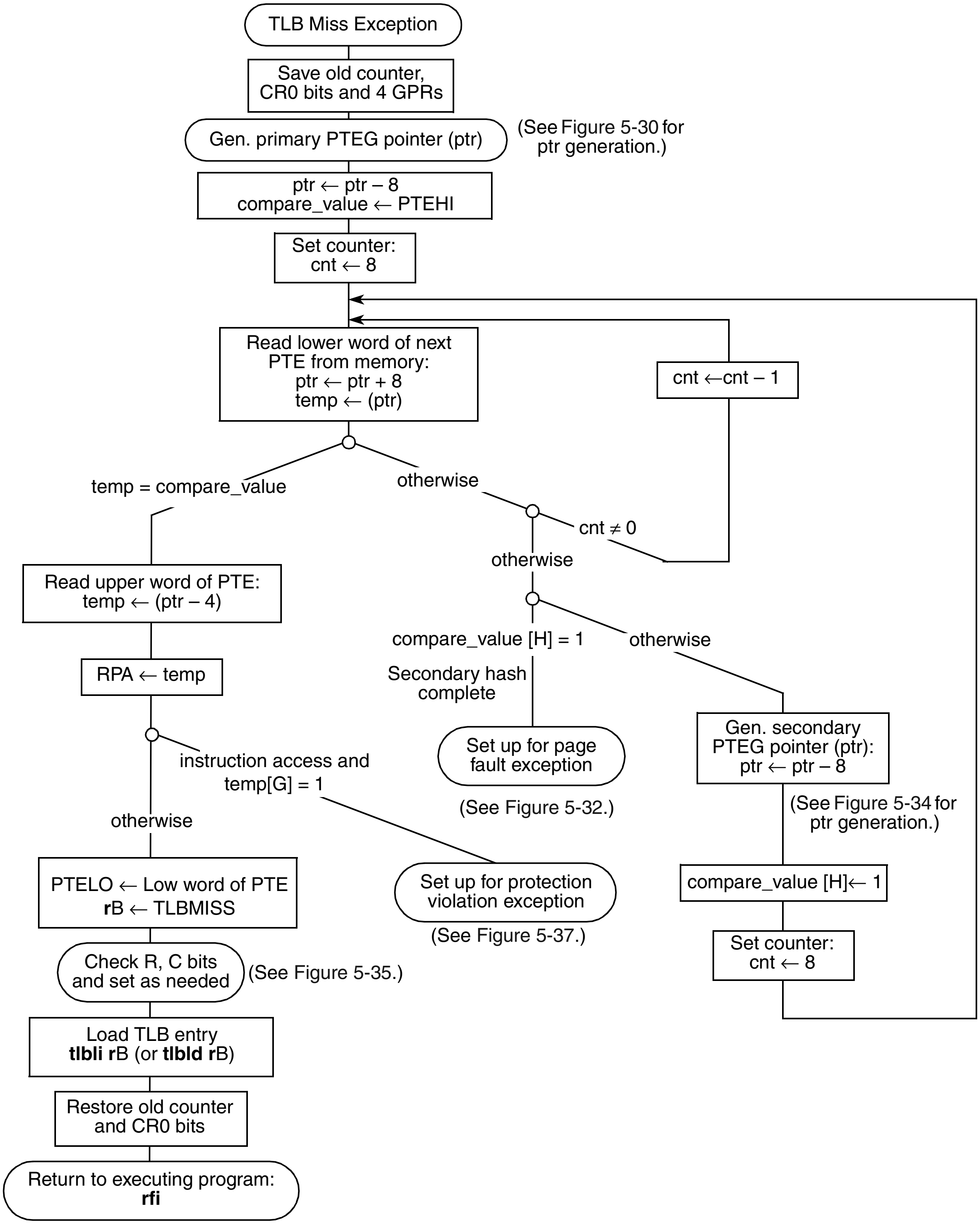


Figure 5-33. Flow for Example Software Table Search Operation

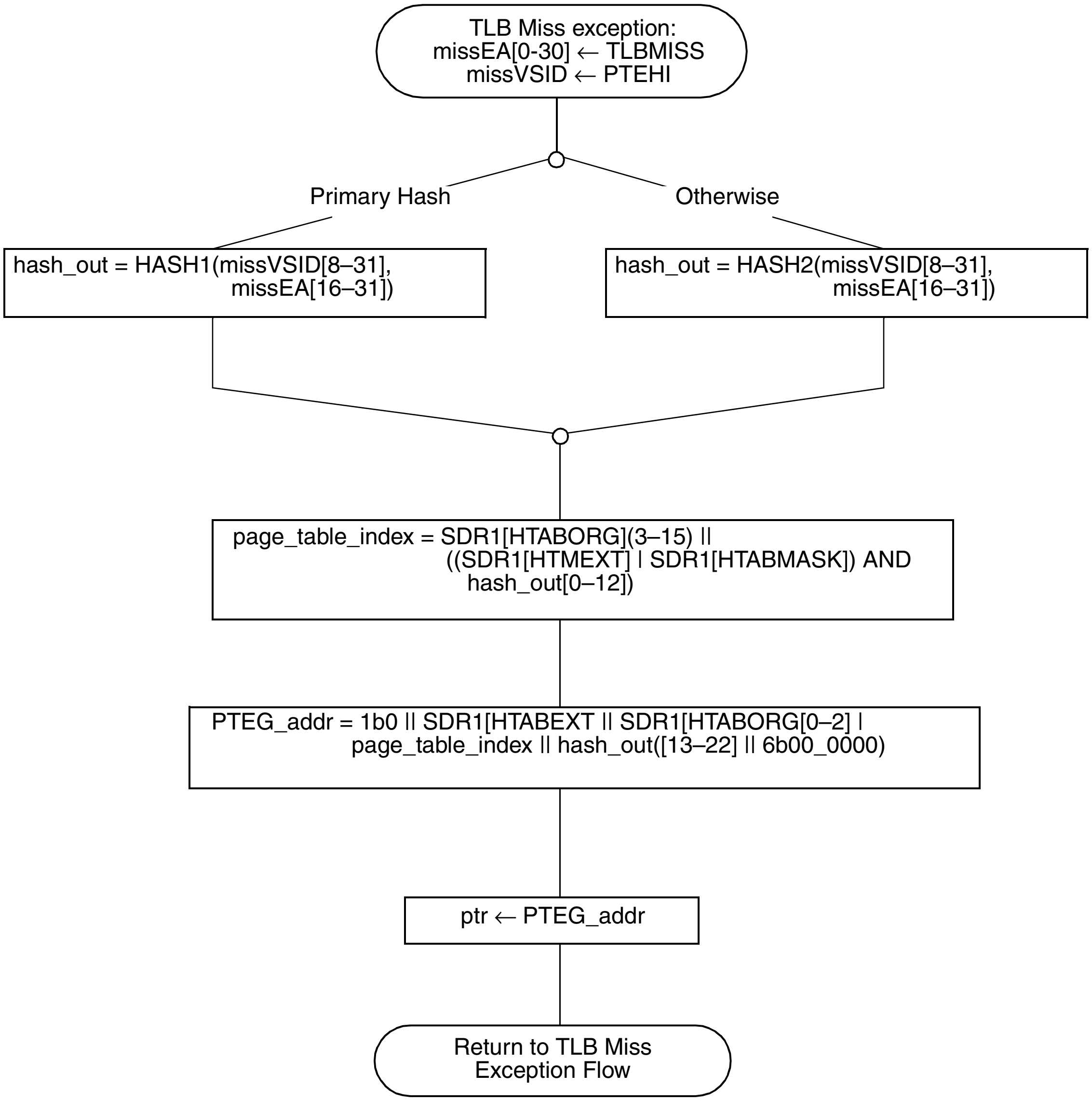


Figure 5-34. Flow for Generation of PTEG Address

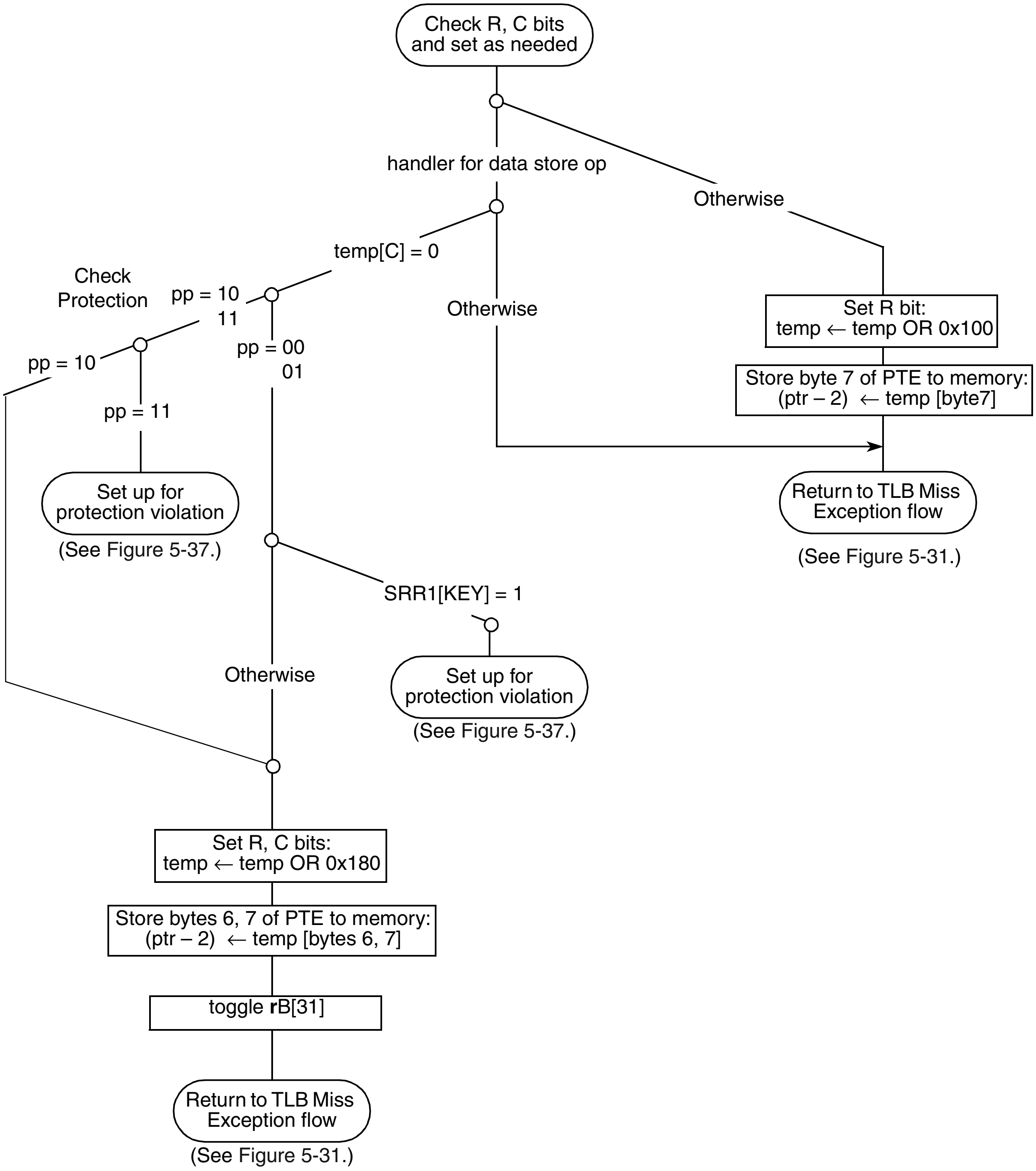


Figure 5-35. Check and Set R and C Bit Flow

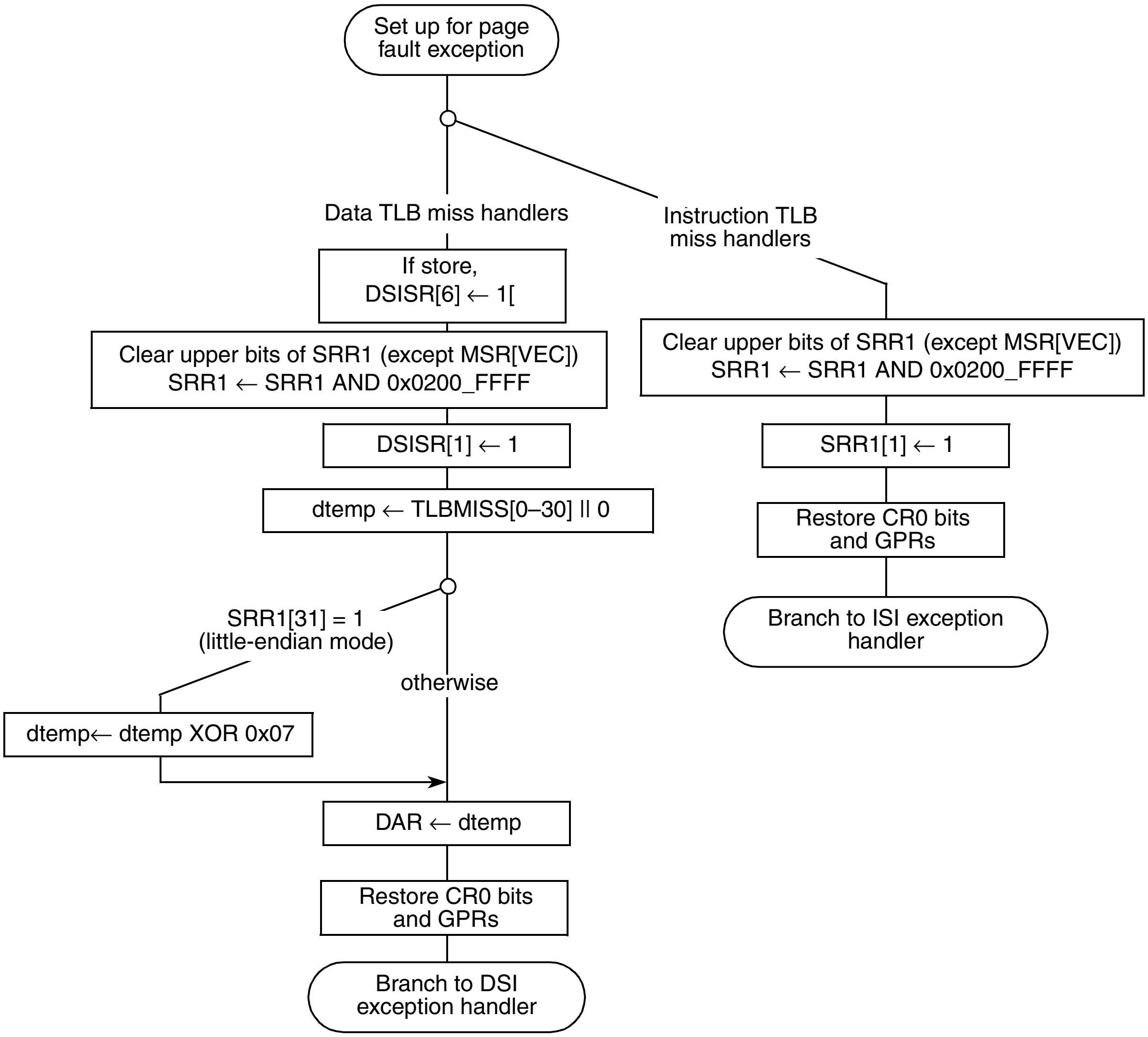


Figure 5-36. Page Fault Setup Flow

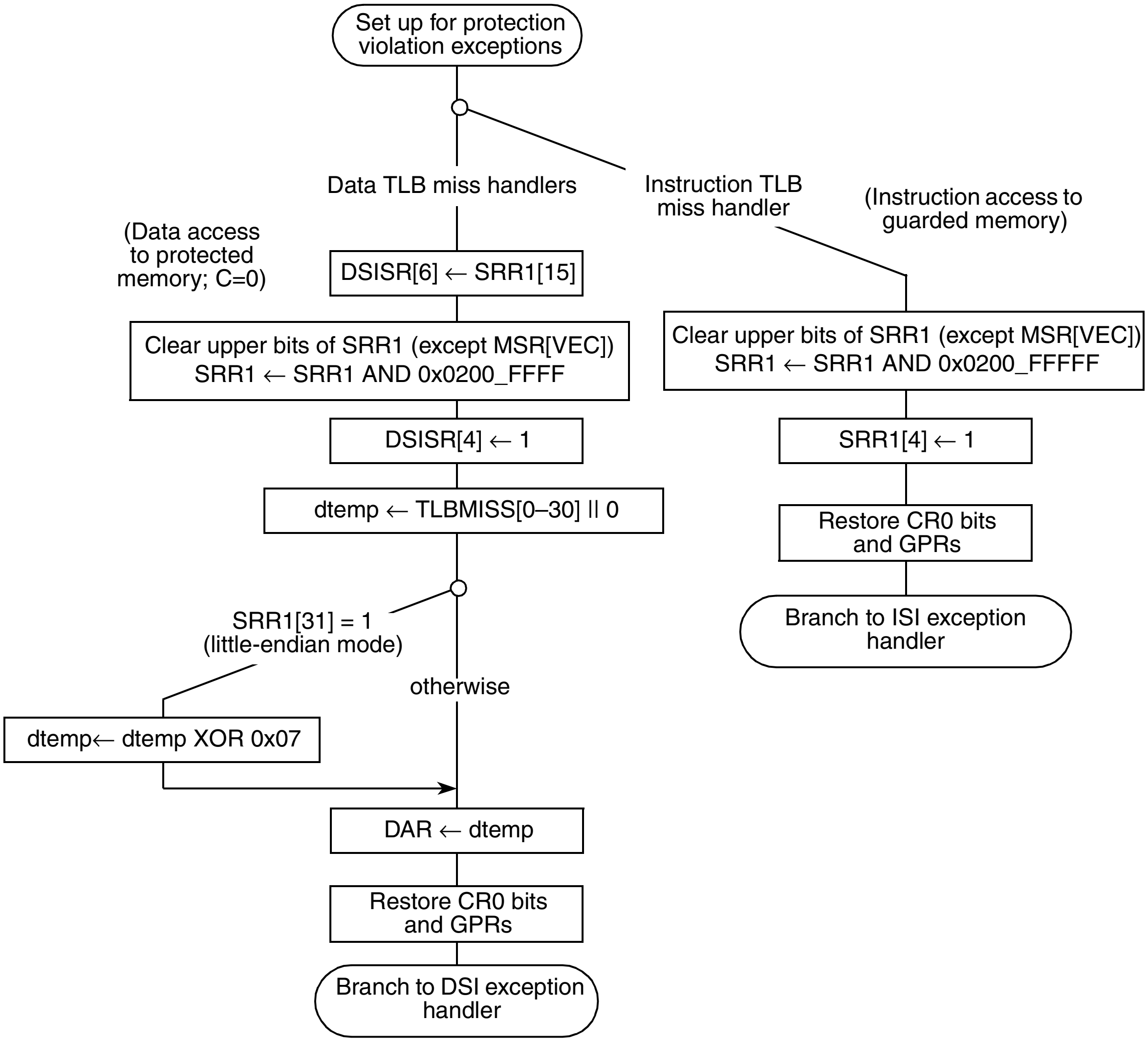


Figure 5-37. Setup for Protection Violation Exceptions

5.5.5.2.2 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 MPC7450

#
# New Instructions:
#       tlbld    - write the dtlb with the ptehi and ptelo values
#       tlbli    - write the itlb with the ptehi and ptelo values
# New SPRs
#       tlbmiss  - address of access that missed. Also contains LRU information
#       ptehi    - VSID of access that missed. Gets written to TLB on tlbld or tlbli
#       ptelo    - RPN value written to TLB on tlbld or tlbli
#
```

```
#
# 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_7450
# gpr r0..r3 are saved into SPR0-3
.set     r0, 0
.set     r1, 1
.set     r2, 2
.set     r3, 3
.set     tlbmiss, 980
.set     ptehi, 981
.set     ptelo, 982
.set     c0, 0
.set     dar, 19
.set     dsisr, 18
.set     srr0, 26
.set     srr1, 27
.set     sprg0, 272
.set     sprg1, 273
.set     sprg2, 274
.set     sprg3, 275
.
.csect   tlbmiss[PR]
vec0:
.globl   vec0

.org     vec0+0x300
vec300:
.org     vec0+0x400
vec400:
#+

# Exception vector serves as jump table
# Register usage:
#    Existing values of r0-r3 saved into sprg0-sprg3
# Note: It is assumed that the OS uses r31 as a pointer to the current top of stack.


.org     vec0+0x1000
# Instruction TLB Miss
         stwu r1,-4(r31)   # store r1 to stack
         mflr r1           # save link register
         stwu r1,-4(r31)   # store link register to stack
         bl   tlbInstrMiss # handler routine
         lwz  r1, 0(r31)   # link register from the stack
         addi r31,r31, 4   # pop stack
         mtlr r1           # restore the link register
         lwz  r1, 0(r31)   # r1 from the stack
         addi r31,r31, 4   # pop stack
         rfi
.org     vec0+0x1100
# Data TLB Miss
         stwu r1,-4(r31)   # store r1 to stack
```

```
        mflr r1           # save link register
        stwu r1,-4(r31)   # store link register to stack
        bl   tlbDataMiss  # handler routine
        lwz  r1, 0(r31)   # link register from the stack
        addi r31,r31, 4   # pop stack
        mtlr r1           # restore the link register
        lwz  r1, 0(r31)   # r1 from the stack
        addi r31,r31, 4   # pop stack
        rfi
.org    vec0+0x1200
# Data TLB Miss for Store
        stwu r1,-4(r31)   # store r1 to stack
        mflr r1           # save link register
        stwu r1,-4(r31)   # store link register to stack
        bl   tlbCeq0      # handler routine
        lwz  r1, 0(r31)   # link register from the stack
        addi r31,r31, 4   # pop stack
        mtlr r1           # restore the link register
        lwz  r1, 0(r31)   # r1 from the stack
        addi r31,r31, 4   # pop stack
        rfi

# Instruction TB miss flow
# Entry:
#       From Exception Vec = 1000
#       srr0 -> address of instruction that missed
#       srr1 -> 16:31 = saved MSR
#       tlbMiss-> ea that missed
#       tlbhi-> upper 32-bits of pte value
#       tlblo-> lower 32-bits of pte value
#
# Register usage:
#   r0-r3 used in the exception handler as follows
#       r0 is scratch pad
#       r1 is scratch pad
#       r2 is pointer to pteg
#       r3 is current compare value
        r31 pointer to top of stack
# Note: It is assumed that the OS uses r31 as a pointer to the current top of   #       stack.
.orig   0x2000
tlbInstrMiss:
        mtsprsprg0, r0    # save r0 into sprg0
        mtsprsprg1, r1    # save r1 into sprg1
        mtsprsprg2, r2    # save r2 into sprg2
        mtsprsprg3, r3    # save r3 into sprg3
# Save CTR and CR on stack.
        mfctrr0           # save counter
        stwu r0,-4(r31)   # store counter to stack
        mfcr r0           # save CR
        stwu r0,-4(r31)   # store CR to stack
        mfsprr0, tlbMiss  # EA of access that missed
        rlwinmr0,r0,20,16,31# Mask out lower 16 bits of EA
        mfsprr1, ptehi    # VSID of access that missed
        rlwinmr1,r1,25,8,31# Mask out uppder 23 bits of VSID
        xor  r1,r0,r1     # Primary HASH
        mfsprr3,sdr1      # SDR1 value
        rlwinmr0,r3,10,13,31# align HTMEXT and HTABMASK fields
```

```
        ori  r0,r0,0x3ff  # Mask out HTMEXT and HTABMASK
        and  r1,r0,r1     # and result
        rlwinmr0,r3,26,13,21
        or   r1,r0,r1     # or result

# 32-bit PTEG Address generation into r2
        andis.r2,r3,0xfe00
        rlwimir2,r1,6,7,25

        xor  r1,r1,r1     # zero out reg.
        addi r1, r0, 8    # load 8 for counter
        mfctrr0           # save counter
        mfsprr3, ptehi    # get first compare value
        addi r2, r2, -8   # pre dec the pointer
im0:
        mtctrr1           # load counter
im1:
        lwzu r1,8(r2)     # get next pte
        cmp  c0,r1,r3     # see if found pte
        bdnzfeq,im1       # dec count br if cmp ne and if count not zero
        bne  instrSecHash # if not found set up second hash or exit
        lwz  r1,4(r2)     # load tlb entry lower-word
        andi.r3,r1,8      # check G-bit
        bne  doISIp       # if guarded, take an ISI
        ori  r1,r1,0x100  # set reference bit
        mtsprptelo,r1     # put rpn into ptelo reg.
        mfsprr0, tlbmiss
        tlblir0           # load the itlb
        srwi r1,r1,8      # get byte 7 of pte
        stb  r1,6(r2)     # update page table
# Restore application values
        lwz  r0,0(r31)    # get counter value
        addi r31,r31,4    # pop stack
        mtctrr0           # restore counter
        lwz  r0,0(r31)    # get CR value
        addi r31,r31,4    # pop stack
        mtcrf0xff,r0      # restore CR
        mfsprr0, sprg0    # restore old value of r0
        mfsprr1, sprg1    # restore old value of r1
        mfsprr2, sprg2    # restore old value of r2
        mfsprr3, sprg3    # restore old value of r3
        blr               # return to jump table

#+
# 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
        mfsprr0,tlbMiss   # EA of access that missed
        rlwinmr0,r0,20,16,31# Mask out lower 16 bits of EA
        mfsprr1,ptehi     # VSID of access that missed
        rlwinmr1,r1,25,8,31# Mask out uppder 23 bits of VSID
```

```
         xor  r1,r0,r1     # Primary HASH
         mfsprr3,sdr1      # SDR1 value
         rlwinmr0,r3,10,13,31# align HTMEXT and HTABMASK fields
         ori  r0,r0,0x3ff  # Mask out HTMEXT and HTABMASK
         and  r1,r0,r1     # and result
         rlwinmr0,r3,26,13,21
         or   r1,r0,r1     # or result
# 32-bit PTEG Address generation into r2
         andis.r2,r3,0xfe00
         rlwimir2,r1,6,7,25

         ori  r3,r3,0x0040 # change the compare value
         addi r1, r0, 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:
         mfsprr3,srr1      # get srr1
         andi.r2,r3,0xFFFF # clean upper srr1
         addisr2,r2,0x0800 # or in srr<4> = 1 to flag prot violation
         b    isi1
doISI:
         mfsprr3,srr1      # get srr1
         andi r2,r3,0xFFFF # clean srr1
         addisr2,r2,0x4000 # or in srr1<1> = 1 to flag pte not found
isi1:
         mtctrr0           # restore counter
         mtsprsrr1,r2      # set srr1
         mtcrf0x80,r3      # restore CR0
         mfsprr0,sprg0     # restore old value of r0
         mfsprr1,sprg1     # restore old value of r1
         mfsprr2,sprg2     # restore old value of r2
         mfsprr3,sprg3     # restore old value of r3
         b    isiExc       # go to instr. access exception


+
# Data TLB miss flow
# Entry:
#    From Exception Vec = 1100
#        srr0 -> address of instruction that caused data tlb miss
#        srr1 -> store 16:31 = saved MSR
#        tlbMiss-> ea that missed
#        tlbhi-> upper 32-bits of pte value
#        tlblo-> lower 32-bits of pte value
#
# Register usage:
#        r0 is scratch pad
#        r1 is scratch pad
#        r2 is pointer to pteg
```

```
#        r3 is current compare value
         r31 pointer to top of stack
# Note: It is assumed that the OS has a stack for saving and restoring          #
       application variables. r31 serves as a pointer to the current top     #        of stack.
#-
.csect   tlbmiss[PR]

tlbDataMiss:
         mtsprsprg0,r0      # save r0 into sprg0
         mtsprsprg1,r1      # save r1 into sprg1
         mtsprsprg2,r2      # save r2 into sprg2
         mtsprsprg3, r3     # save r3 into sprg3
# Save CTR and CR on stack.
         mfctrr0            # save counter
         stwu r0,-4(r31)    # store counter to stack
         mfcr r0            # save CR
         stwu r0,-4(r31)    # store CR to stack
         mfsprr0,tlbMiss    # EA of access that missed
         rlwinmr0,r0,20,16,31# Mask out lower 16 bits of EA
         mfsprr1,ptehi      # VSID of access that missed
         rlwinmr1,r1,25,8,31# Mask out uppder 23 bits of VSID
         xor  r1,r0,r1      # Primary HASH
         mfsprr3,sdr1       # SDR1 value
         rlwinmr0,r3,10,13,31# align HTMEXT and HTABMASK fields
         ori  r0,r0,0x3ff   # Mask out HTMEXT and HTABMASK
         and  r1,r0,r1      # and result
         rlwinmr0,r3,26,13,21
         or   r1,r0,r1      # or result
# 32-bit PTEG Address generation into r2
         andis.r2,r3,0xfe00
         rlwimir2,r1,6,7,25

         xor  r1,r1,r1      # zero out reg.
         addi r1,r1,8       # load 8 for counter
         mfsprr3,ptehi      # get first compare value
         addi r2,r2,-8      # pre dec the pointer
dm0:
         mtctrr1            # load counter
dm1:
         lwzu r1,8(r2)      # get next pte
         cmp  c0,r1,r3      # see if found pte
         bdnzfeq,dm1        # dec count br if cmp ne and if count not zero
         bne  dataSecHash   # if not found set up second hash or exit
         lwz  r1,4(r2)      # load tlb entry lower-word
         ori  r1,r1,0x100   # set reference bit
         mtsprptelo,r1      # put rpn into ptelo reg.
         mfsprr0,tlbmiss
         tlbldr0            # load the dtlb
         srwi r1,r1,8       # get byte 7 of pte
         stb  r1,6(r2)      # update page table
# Restore application values
         lwz  r0,0(r31)     # get counter value
         addi r31,r31,4     # pop stack
         mtctrr0            # restore counter
         lwz  r0,0(r31)     # get CR value
         addi r31,r31,4     # pop stack
         mtcrf0xff,r0       # restore CR
```

```
        mfsprr0,sprg0     # restore old value of r0
        mfsprr1,sprg1     # restore old value of r1
        mfsprr2,sprg2     # restore old value of r2
        mfsprr3,sprg3     # restore old value of r3
        blr               # return to jump table

dataSecHash:
        andi.r1,r3,0x0040 # see if we have done second hash
        bne  doDSI        # if so, go to DSI exception
        mfsprr0,tlbMiss   # EA of access that missed
        rlwinmr0,r0,20,16,31# Mask out lower 16 bits of EA
        mfsprr1,ptehi     # VSID of access that missed
        rlwinmr1,r1,25,8,31# Mask out uppder 23 bits of VSID
        xor  r1,r0,r1     # Primary HASH
        mfsprr3,sdr1      # SDR1 value
        rlwinmr0,r3,10,13,31# align HTMEXT and HTABMASK fields
        ori  r0,r0,0x3ff  # Mask out HTMEXT and HTABMASK
        and  r1,r0,r1     # and result
        rlwinmr0,r3,26,13,21
        or   r1,r0,r1     # or result
# 32-bit PTEG Address generation into r2
        andisr2,r3,0xfe00
        rlwimir2,r1,6,7,25
        ori  r3,r3,0x0040 # change the compare value?
        addi r1,r0,8      # 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:
#       From Exception Vec = 1200
#       srr0 -> address of store that caused the exception
#       srr1 -> 16:31 = saved MSR
#       tlbMiss-> ea that missed
#       tlbhi-> upper 32-bits of pte value
#       tlblo-> lower 32-bits of pte value
#
# Register usage:
#       r0 is saved counter
#       r1 is junk
#       r2 is pointer to pteg
#       r3 is current compare value
        r31 pointer to top of stack
# Note: It is assumed that the OS has a stack for saving and restoring
#        application variables. r31 serves as a pointer to the current #        top of stack.
#-

.csect   tlbmiss[PR]

tlbCeq0:
        mtspr    sprg0,r0 # save r0 into sprg0
        mtspr    sprg1,r1 # save r1 into sprg1
        mtspr    sprg2,r2 # save r2 into sprg2
        mtspr    sprg3,r3 # save r3 into sprg3
# Save CTR reg and CR on stack.
        mfctr    r0       # save counter
        stwu     r0,-4(r31)# store counter to stack
```

```
         mfcr     r0       # save CR
         stwu     r0,-4(r31)# store CR to stack
         mfspr    r0,tlbMiss# EA of access that missed
         rlwinm   r0,r0,20,16,31# Mask out lower 16 bits of EA
         mfspr    r1,ptehi # VSID of access that missed
         rlwinm   r1,r1,25,8,31# Mask out uppder 23 bits of VSID
         xor      r1,r0,r1 # Primary HASH
         mfspr    r3,sdr1  # SDR1 value
         rlwinm   r0,r3,10,13,31# align HTMEXT and HTABMASK fields
         ori      r0,r0,0x3ff# Mask out HTMEXT and HTABMASK
         and      r1,r0,r1 # and result
         rlwinm   r0,r3,26,13,21
         or       r1,r0,r1 # or result
# 32-bit PTEG Address generation into r2
         andis.   r2,r3,0xfe00
         rlwimi   r2,r1,6,7,25

         xor      r1,r1,r1      # zero out reg.
         addi     r1,r1,8       # load 8 for counter
         mfspr    r3,ptehi      # 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
         bdnzf    eq, ceq1      # dec count br if cmp ne and if count not zero
         bne      cEq0SecHash   # if not found set up second hash or exit
         lwz      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:
         mtspr    ptelo,r1      # put rpn into ptelo reg.
         mfspr    r0,tlbmiss
         xori     r0,r0,0x01    # toggles lru bit
         tlbld    r0            # load the dtlb
# Restore application values
         lwz      r0,0(r31)     # get counter value
         addi     r31,r31,4     # pop stack
         mtctr    r0            # restore counter
         lwz      r0, 0(r31)    # get CR value
         addi     r31,r31,4     # pop stack
         mtcrf    0xff,r0       # restore CR
         mfspr    r0,sprg0      # restore old value of r0
         mfspr    r1,sprg1      # restore old value of r1
         mfspr    r2,sprg2      # restore old value of r2
         mfspr    r3,sprg3      # restore old value of r3
         blr                    # return to jump table
#+
# 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
        andi     r1,r3,0x004   # see if we have done second hash
        bne      doDSI         # if so, go to DSI exception
        mfspr    r0,tlbMiss    # EA of access that missed
        rlwinm   r0,r0,20,16,3 # Mask out lower 16 bits of EA
        mfspr    r1,pteh       # VSID of access that missed
        rlwinm   r1,r1,25,8,31 # Mask out uppder 23 bits of VSID
        xor      r1,r0,r1      # Primary HASH
        mfspr    r3,sdr1       # SDR1 value
        rlwinm   r0,r3,10,13,3 # align HTMEXT and HTABMASK fields
        ori      r0,r0,0x3f    # Mask out HTMEXT and HTABMASK
        and      r1,r0,r1      # and result
        rlwinm   r0,r3,26,13,21
        or       r1,r0,r1      # or result
# 32-bit PTEG Address generation into r2
        andis.   r2,r3,0xfe00
        rlwimi   r2,r1,6,7,25

         ori     r3,r3,0x004   # change the compare value
         addi    r1,r0,8       # 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)
        b        doDSIp        # else DSIp
chk2:
        ori      r1,r1,0x180   # set reference and change bit
        sth      r1,6(r2)      # update page table
        b        ceq2          # and back we go
        #
#+
# entry Not Found: synthesize a DSI exception
        # Entry:
#       r0 is saved counter
#       r1 is junk
#       r2 is pointer to pteg
```

```
#       r3 is current compare value
#
doDSI:
        mfspr    r3,srr1       # get srr1
        rlwinm   r1,r3,9,6,6   # get srr1<flag> to bit 6 for load/store, zero rest
        addis    r1,r1,0x4000  # or in dsisr<1> = 1 to flag pte not found
        b        dsi1
doDSIp:
        mfspr    r3, srr1      # get srr1
        rlwinm   r1, r3,9,6,6  # get srr1<flag> to bit 6 for load/store, zero rest
        addis    r1, r1, 0x0800# or in dsisr<4> = 1 to flag prot violation
dsi1:
        mtctr    r0            # restore counter
        andis.   r2,r3, 0x0200 # Keep the AltiVec Avail bit in r2
        andi     r3,r3,0xFFFF  # Zero out the upper bits of SRR1
        or       r2, r2, r3    # OR back in the lower bits into r2
        mtspr    srr1, r2      # set srr1
        mtspr    dsisr, r1     # load the dsisr
        mfspr    r1, tlbmiss   # get miss address
        rlwinm   r1,r1,0,0,30  # Clear the LRU bit
        rlwinm.  r2,r2,0,31,31 # test LE bit
        beq      dsi2          # if little endian then:
        xori     r1,r1,0x07    # de-mung the data address
dsi2:
        mtspr    dar,r1        # put in dar
        mtcrf    0x80,r3       # restore CR0?
        mfspr    r0,sprg0      # restore old value of r0
        mfspr    r1,sprg1      # restore old value of r1
        mfspr    r2,sprg2      # restore old value of r2
        mfspr    r3,sprg3      # restore old value of r3
        b        dsiExc        # branch to DSI exception
```

Chapter 6
Instruction Timing

This chapter describes how the MPC7450 microprocessor performs operations defined by instructions and how it reports the results of instruction execution. It gives detailed descriptions of how the MPC7450 execution units work and how these units interact with other parts of the processor, such as the instruction fetching mechanism, register files, and caches. It gives examples of instruction sequences, showing potential bottlenecks and how to minimize their effects. Finally, it includes tables that identify the unit that executes each instruction implemented on the MPC7450, the latency for each instruction, and other information useful to assembly language programmers.

AltiVec Technology and Instruction Timing

The AltiVec functionality in the MPC7450 affects instruction timing in the following ways:

- Execution units are provided for vector computations:
  - Vector permute unit (VPU). See Section 6.4.5.1.1, “AltiVec Permute Unit (VPU) Execution Timing.”
  - Short-latency vector integer unit 1 (VIU1). See Section 6.4.5.1.2, “Vector Simple Integer Unit (VIU1) Execution Timing.”
  - Long-latency vector complex integer unit (VIU2). See Section 6.4.5.1.3, “Vector Complex Integer Unit (VIU2) Execution Timing.”
  - Vector floating-point unit (VFPU). See Section 6.4.5.1.4, “Vector Floating-Point Unit (VFPU) Execution Timing.”
- The AltiVec technology defines data streaming instructions that allow automated loading of data for non-speculative accesses. These instructions can be identified as either static (likely to be reused) or transient (unlikely to be reused). See Chapter 7, “AltiVec Technology Implementation.”
- The AltiVec technology defines the difference between the instructions **lvxl** and **stvxl** with other AltiVec load and store instructions. See Section 6.4.4.3.1, “LRU Instructions.”

6.1 Terminology and Conventions

This section provides an alphabetical glossary of terms used in this chapter. These definitions offer a review of commonly used terms and point out specific ways these terms are used in this chapter.

NOTE

> Many of these definitions differ slightly from those used to describe previous processors that implement the PowerPC architecture, in particular with respect to dispatch, issue, finishing, retirement, and write back, so please read this glossary carefully.

- Branch prediction—The process of guessing the direction or target of a branch. Branch direction prediction involves guessing whether a branch will be taken. Target prediction involves guessing the target address of a **bclr** branch. The PowerPC architecture defines a means for static branch prediction as part of the instruction encoding.
- Branch resolution—The determination of whether a branch prediction was correct or not. If the prediction is correct, the instructions following the predicted branch that may have been speculatively executed can complete (*see* completion). If the prediction is incorrect, instructions on the mispredicted path and any results of speculative execution are purged from the pipeline and fetching continues from the correct path.
- Complete—An instruction is in the complete stage after it executes and makes its results available for the next instruction (*see* finish). At the end of the complete stage, the instruction is retired from the completion queue (CQ). When an instruction completes, it is guaranteed that this instruction and all previous instructions can cause no exceptions.
- Dispatch—The dispatch stage decodes instructions supplied by the instruction queue, renames any source/target operands, determines to which issue queue each non-branch instruction is dispatched, and determines whether the required space is available in both that issue queue and the completion queue.
- Fall-through folding (branch fall-through)—Removal of a not-taken branch. On the MPC7450, not-taken branch instructions that do not update LR or CTR can be removed from the instruction stream if the branch instruction is in IQ3–IQ7 the cycle after execution.
- Fetch—The process of bringing instructions from memory (such as a cache or system memory) into the instruction queue.
- Finish—An executed instruction finishes by updating the completion queue that execution is complete and results have been made available to subsequent instructions. For most execution units, finishing occurs at the end of the last cycle of execution; however, FPU, IU2, and VIU2 instructions finish at the end of a single-cycle finish stage after the last cycle of execution.
- Folding (branch folding)—The replacement with target instructions of a branch instruction and any instructions along the not-taken path when a branch is either taken or predicted as taken.
- Issue—The pipeline stage responsible for reading source operands from rename registers and register files. This stage also assigns and routes instructions to the proper execution unit.
- Latency— The number of clock cycles necessary to execute an instruction and make the results of that execution available to subsequent instructions.

- 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 can take many cycles to make results available (*see* latency), pipelining makes it possible to overlap processing so that the throughput (number of instructions processed per cycle) is greater than if pipelining were not implemented.
- Program order—The order of instructions in an executing program. More specifically, this term is used to refer to the original order in which program instructions are fetched into the instruction queue from the cache.
- Rename registers—Temporary buffers for holding results of instructions that have finished execution but have not completed
- Reservation station—A buffer between the issue and execute stages that allows instructions to be issued even though the results of other instructions on which the issued instruction may depend are not available.
- Retirement—Removal of a completed instruction from the CQ.
- Speculative instruction—Any instruction which is currently behind an older branch that has not been resolved yet.
- Stage—Used in two different senses, depending on whether the pipeline is being discussed as a physical entity or a sequence of events. In the latter case, a stage is an element in the pipeline during which certain actions are performed, such as decoding the instruction, performing an arithmetic operation, or writing back the results. Typically, the latency of a stage is one processor clock cycle. Some events, such as dispatch, write-back, and completion, happen instantaneously and may be thought to occur at the end of a stage.

  An instruction can spend multiple cycles in one stage. An integer multiply, for example, takes multiple cycles in the execute stage. When this occurs, subsequent instructions may stall.

  An instruction can also occupy more than one stage simultaneously, especially in the sense that a stage can be seen as a physical resource—for example, when instructions are dispatched they are assigned a place in the CQ at the same time they are passed to the issue queues.
- 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 execute stage at the same time.
- Throughput—The number of instructions that are processed per cycle. For example, a series of **mulli** instructions have a throughput of one instruction per clock cycle.
- Write-back—Write-back (in the context of instruction handling) occurs when a result is written into the architecture-defined registers (typically the GPRs, FPRs, and VRs). On the MPC7450, write back occurs in the clock cycle after the completion stage. Results in the write-back buffer cannot be flushed. If an exception occurs, results from previous instructions must write back before the exception is taken.

6.2 Instruction Timing Overview

The MPC7450 design minimizes the number of clock cycles it takes to fetch, decode, dispatch, issue, and execute instructions and to make the results available for a subsequent instruction. Some instructions, such as loads and stores, access memory and require additional clock cycles between the execute phase and the write-back phase. These latencies vary depending on whether the access is to cacheable or noncacheable memory, whether it hits in the L1, L2, or L3 cache, whether the cache access generates a write-back to memory, whether the access causes a snoop hit from another device that generates additional activity, and other conditions that affect memory accesses. Note that L3 cache is not supported on the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448.

To improve throughput, the MPC7450 implements pipelining, superscalar instruction issue, branch folding, removal of fall-through branches, three-level speculative branch handling, and multiple execution units that operate independently and in parallel.

As an instruction passes from stage to stage, the subsequent instruction can follow through the stages as the former instruction vacates them, allowing several instructions to be processed simultaneously. Although it may take several cycles for an instruction to pass through all the stages, when the pipeline is full, one instruction can complete on every clock cycle. Figure 6-1 represents a generic four-stage pipelined execution unit, which when filled has a throughput of one instruction per clock cycle.

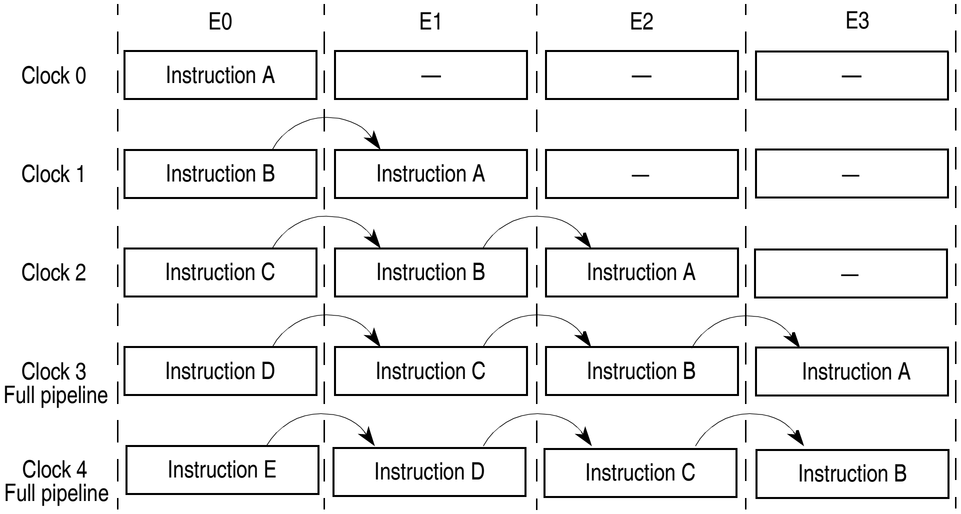


Figure 6-1. Pipelined Execution Unit

Figure 6-2 shows the entire path that instructions take through the fetch1, fetch2, decode/dispatch, execute, issue, complete, and write-back stages, which is considered the MPC7450's master pipeline. The FPU, LSU, IU2, VIU2, VFPU, and VPU are also multiple-stage pipelines.

The MPC7450 contains the following execution units:

- Branch processing unit (BPU)
- Three Single-cycle IUs (IU1a, IU1b, IU1c)—executes all integer (fixed-point) instructions except multiply, divide, and move to/from special-purpose register instructions. Note that all IU1 instructions execute in 1 cycle, except for some instructions like **tw**[i] and **sraw**[i][.], which take 2 cycles. See Table 6-5 for details.

- Multiple-cycle IU (IU2)—executes miscellaneous instructions including the CR logical operations, integer multiplication and division instructions, and move to/from special-purpose register instructions
- 64-bit floating-point unit (FPU)
- Load/store unit (LSU)
- The AltiVec unit contains the following four independent execution units for vector computations and the latencies are shown in Table 6-8:
  - Vector permute unit (VPU)
  - Vector simple integer unit (VIU1)
  - Vector complex integer unit (VIU2)
  - Vector floating-point unit (VFPU)

A maximum of two AltiVec instructions can be issued in order to any combination of AltiVec execution units per clock cycle. In the MPC7448, a maximum of two AltiVec instructions can be issued out-of-order to any combination of AltiVec execution units per clock cycle from the bottom two VIQ entries (VIQ1–VIQ0). This means an instruction in VIQ1 destined for VIU1 does not have to wait for an instruction in VIQ0 that is stalled behind an instruction waiting for operand availability. Moreover, the VIU2, VFPU, and VPU are pipelined, so they can operate on multiple instructions.

The MPC7450 can complete as many as three instructions on each clock cycle. In general, the MPC7450 processes instructions in seven stages—fetch1, fetch2, decode/dispatch, issue, execute, complete, and write-back as shown in Figure 6-2. Note that the pipeline example in Figure 6-1 is similar to the four-stage VFPU pipeline in Figure 6-2.

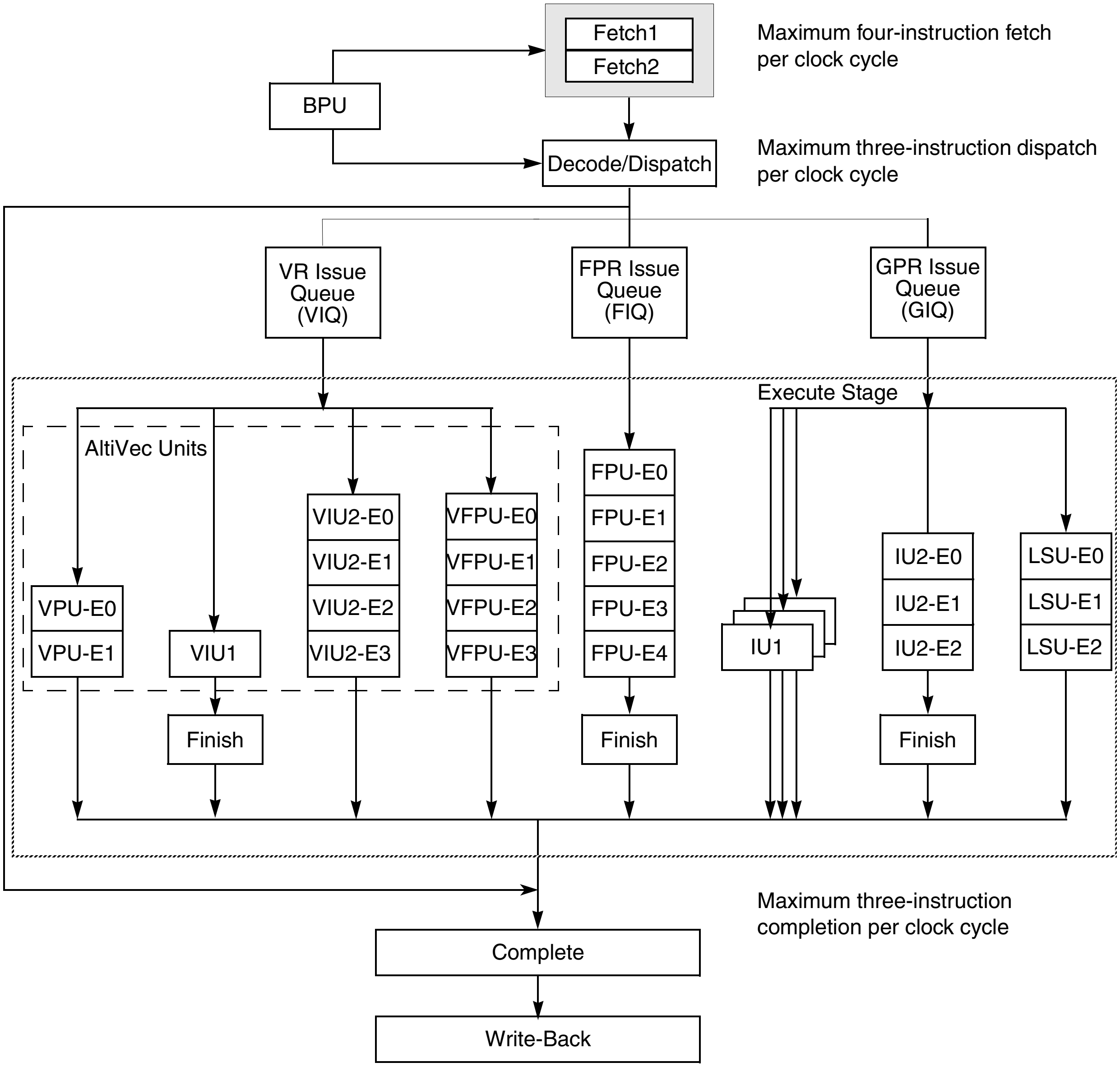


Figure 6-2. Superscalar/Pipeline Diagram

The instruction pipeline stages are described as follows:

- Instruction fetch—Includes the clock cycles necessary to request an instruction and the time the memory system takes to respond to the request. Instructions retrieved are latched into the instruction queue (IQ) for subsequent consideration by the dispatcher.

  Instruction fetch timing depends on many variables, such as whether an instruction is in the branch target instruction cache (BTIC), the on-chip instruction cache, or the L2 or L3 cache. Those factors increase when it is necessary to fetch instructions from system memory and include the processor-to-bus clock ratio, the amount of bus traffic, and whether any cache coherency operations are required.

Because there are so many variables, unless otherwise specified, the instruction timing examples in this chapter assume optimal performance and show the portion of the fetch stage in which the instruction is in the instruction queue. The fetch1 and fetch2 stages are primarily involved in retrieving instructions.

- The decode/dispatch stage fully decodes each instruction; most instructions are dispatched to the issue queues (branch, **isync**, **rfi**, and **sc** instructions do not go to issue queues).
- The three issue queues FIQ, VIQ, and GIQ can accept as many as one, two, and three instructions, respectively, in a cycle. Instruction dispatch requires the following:
  - — Instructions can be dispatched only from the three lowest IQ entries—IQ0, IQ1, and IQ2.
  - — A maximum of three instructions can be dispatched to the issue queues per clock cycle.
  - — Space must be available in the CQ for an instruction to dispatch (this includes instructions that are assigned a space in the CQ but not in an issue queue).

  In this chapter, dispatch is treated as an event at the end of the fetch stage. Dispatch dependencies are described in Section 6.7.2, “Dispatch Unit Resource Requirements.”

  The issue stage reads source operands from rename registers and register files and determines when instructions are latched into the execution unit reservation stations. The GIQ, FIQ, and VIQ (AltiVec) issue queues have the following similarities:
    - – Operand lookup in the GPRs, FPRs, and VRs, and their rename registers.
    - – Issue queues issue instructions to the proper execution units.
    - – Each issue queue holds twice as many instructions as can be dispatched to it in 1 cycle; the GIQ has six entries, the VIQ has four, and the FIQ has two.

  The three issue queues are described as follows:
  - — The GIQ accepts as many as three instructions from the dispatch unit each cycle. IU1, IU2, and all LSU instructions (including floating-point and AltiVec loads and stores) are dispatched to the GIQ.
  - — Instructions can be issued out-of-order from the bottom three GIQ entries (GIQ2–GIQ0). An instruction in GIQ1 destined for an IU1 does not have to wait for an instruction in GIQ0 that is stalled behind a long-latency integer divide instruction in the IU2.
  - — The VIQ accepts as many as two instructions from the dispatch unit each cycle. All AltiVec instructions (other than load, store, and vector touch instructions) are dispatched to the VIQ. As many as two instructions can be issued to the four AltiVec execution units, but unlike the GIQ, instructions in the VIQ cannot be issued out-of-order.
  - — The FIQ can accept one instruction from the dispatch unit per cycle. It looks at the first instruction in its queue and determines if the instruction can be issued to the FPU in this cycle.
- The execute stage accepts instructions from its issue queue when the appropriate reservation stations are not busy. In this stage, the operands assigned to the execution stage from the issue stage are latched.

  The execution unit executes the instruction (perhaps over multiple cycles), writes results on its result bus, and notifies the CQ when the instruction finishes. The execution unit reports any exceptions to the completion stage. Instruction-generated exceptions are not taken until the excepting instruction is next to retire.

Most integer instructions have a 1-cycle latency, so results of these instructions are available 1 clock cycle after an instruction enters the execution unit. The FPU, LSU, IU2, VIU2, VFPU, and VPU units are pipelined, as shown in Figure 6-3.

Note that AltiVec computational instructions are executed in the four independent, pipelined AltiVec execution units. The VPU has a two-stage pipeline, the VIU1 has a one-stage pipeline, and the VIU2 and VFPU have four-stage pipelines. As many as 10 AltiVec instructions can execute concurrently.

- The complete and write-back stages maintain the correct architectural machine state and commit results to the architecture-defined registers in the proper order. If completion logic detects an instruction containing an exception status, all following instructions are cancelled, their execution results in rename buffers are discarded, and the correct instruction stream is fetched.

  The complete stage ends when the instruction is retired. Three instructions can be retired per clock cycle. If no dependencies exist, as many as three instructions are retired in program order. Section 6.7.4, “Completion Unit Resource Requirements,” describes completion dependencies.

  The write-back stage occurs in the clock cycle after the instruction is retired.

Notation conventions used in the instruction timing examples in Figure 6-8 through Figure 6-17 are as follows:

▭ Fetch—Instructions are fetched from memory and placed in the 12-entry IQ. The latency associated with accessing an instruction depends on whether the instruction is in the BTIC, the on-chip caches, the off-chip L3 cache, or system memory (in which case latency is further affected by bus traffic, bus clock speed, and address translation issues). Therefore, in the examples in this chapter, the diagrams and fetch stage shown is for the common case of instructions hitting in the instruction cache.

⬒ Branch execute—The operations specified by a branch instruction are being performed by the BPU. In some cases, the branch direction or target may be predicted. The white stripe is a reminder that the branch instruction occupies an entry in the IQ.

▤ Dispatch—As many as three eligible instructions move, in order, from the IQ0–IQ2 to the appropriate issue queue. Note that branch, **isync**, **rfi**, and **sc** instructions do not go to issue queues. At the same time, the instruction is assigned an entry in the completion queue.

▩ Issue—Instructions are dispatched to issue queues from the instruction queue entries. At the end of the issue stage, instructions and their operands are latched into execution unit reservation stations. The black stripe is a reminder that the instruction occupies an entry in the CQ, described in Figure 6-3.

▬ Execute—The operations specified by an instruction are being performed by the appropriate execution unit. The black stripe is a reminder that the instruction occupies an entry in the CQ, described in Figure 6-3.

■ Finish (FPU, IU2, and VIU1 only)—The single-cycle finish stage is required for all FPU, IU2, and VIU1 instructions to notify the completion logic that an instruction has executed and its results have been made available to rename registers.

▬ Complete—Execution has finished. When all completion requirements are met, the instruction is retired from the CQ. The results are written back to architecture-defined registers in the clock cycle after retirement.

⬚ Write back—The instruction has retired and its results are written back to the architecture-defined registers.

The following events are associated with the stages described above:

- Dispatch—An instruction is dispatched to the appropriate issue queue at the end of the dispatch stage. At dispatch, the instruction passes to the issue pipeline stage by taking a place in the completion queue and in one of the three issue queues.
- Issue—The issue stage ends when the instruction is issued to the appropriate execution unit.
- Finish—An instruction finishes when the CQ is signalled that execution results are available to subsequent instructions. Architecture-defined registers are not updated until the instruction is retired. For FPU, IU2, and VIU2, finishing occurs at the end of a separate, one-cycle stage after the final execution stage.
- Retire—An instruction is retired when it has updated architecture-defined registers with its results and is removed from the completion queue.
- Write back—The results of a retired instruction are written back to the architecture-defined register.

Figure 6-3 shows the relationships between stages and events.

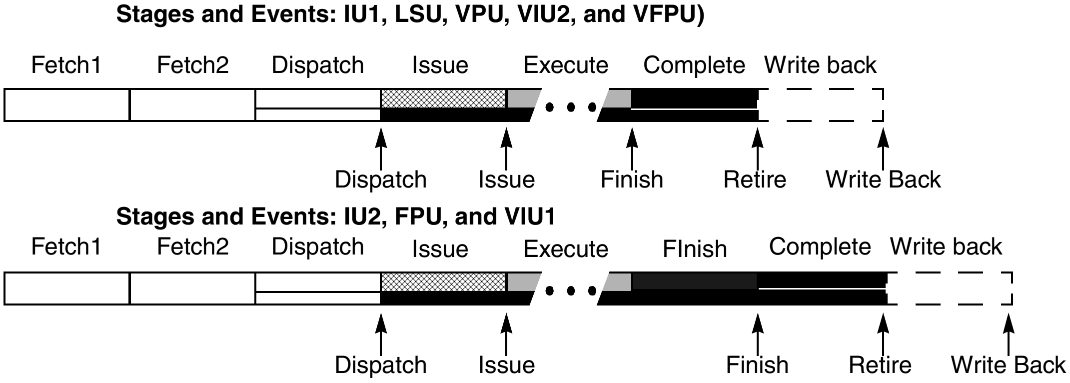
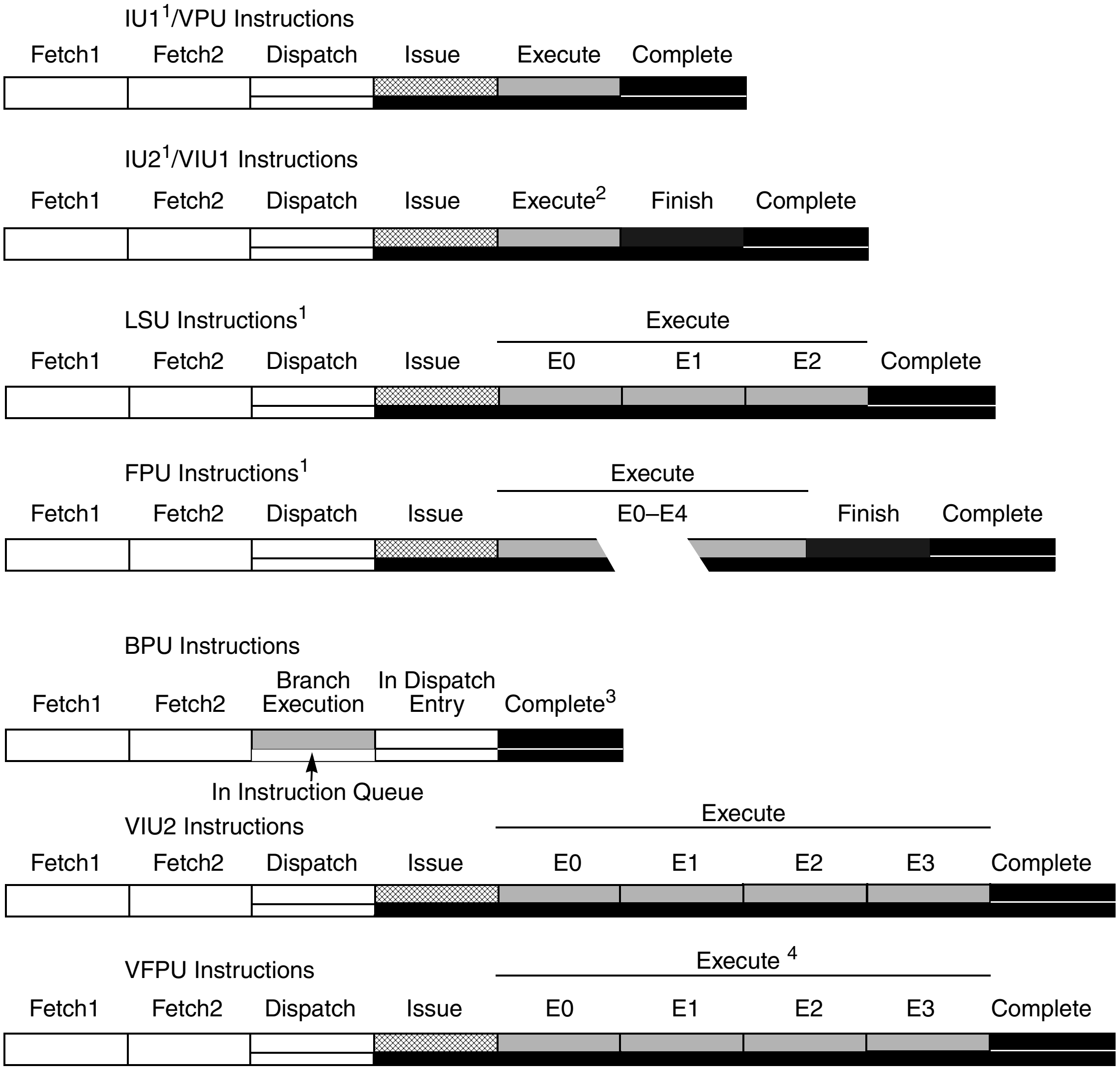


Figure 6-3. Stages and Events

Figure 6-4 shows the stages of MPC7450 execution units.



[1] Execution-serialized instructions require additional execution cycles (not shown).

[2] Several integer instructions, such as multiply and divide instructions, require multiple cycles in the execute stage.

[3] Branches that are not folded take an entry in the completion queue.

[4] Some VFPU instructions skip VFPU-E1 and VFPU-E2, which may increase the latency of other VFPU instructions. See Section 6.4.5.1.4, “Vector Floating-Point Unit (VFPU) Execution Timing.”

Figure 6-4. MPC7450 Microprocessor Pipeline Stages

6.3 Timing Considerations

When the fetch pipeline is full, as many as four instructions can be fetched to the IQ during each clock cycle. Three instructions can be dispatched to the issue queues per clock cycle.

The MPC7450 improves performance by executing multiple instructions at a time, using hardware to manage dependencies. When an instruction is issued, the register file or rename registers send the source data to the appropriate reservation station. Register files and rename registers have sufficient bandwidth to allow dispatch of three instructions per clock cycle under most conditions.

The BPU decodes and executes branches immediately after they reach the IQ. When a branch cannot be resolved due to a CR, CTR, or LR dependency, the branch may be predicted and that case execution continues from the predicted path. If the prediction is incorrect, the following steps are taken:

1. The IQ is purged and fetching continues from the correct path.
2. If mispredicted instructions have entered the CQ, any instructions older than the predicted branch are allowed to retire, after which remaining instructions are purged from the CQ and execution units.
3. Dispatching resumes from the correct path.

After an instruction executes, results are made available to subsequent instructions in the appropriate rename registers. The architecture-defined GPR, FPR, and VR registers are updated during the write-back stage. Branch instructions that update the LR or CTR write back their results in a similar fashion.

After instruction execution, results are made available to subsequent instructions in the appropriate GPR, FPR, or VR rename registers. Results are then stored into the correct GPR, FPR, or VR during the write-back stage. If a subsequent instruction needs the result as a source operand, the result is simultaneously made available to the appropriate execution unit, which allows a data-dependent instruction to be decoded and dispatched without waiting to read the data from the register file. Branch instructions that update either the LR or CTR write back their results in a similar fashion.

The following section describes this process.

6.3.1 General Instruction Flow

As many as four instructions can be fetched into the IQ during each clock cycle. An instruction fetch consists of two single-cycle fetch stages—fetch1 and fetch2.

As many as three instructions can be dispatched per clock cycle, within the limitations of individual issue queues—the GIQ can accept as many as three, the FIQ can accept one, and the VIQ can accept as many as two instructions. Likewise, the GIQ can issue at most three instructions, the FIQ can issue one instruction, and the VIQ can issue two instructions per clock cycle. The MPC7450 tries to keep the IQ full at all times, unless instruction cache throttling is enabled, as described in Chapter 10, “Power and Thermal Management.”

The number of instructions fetched in a clock cycle is determined by the number of vacant spaces in the IQ during the previous clock cycle. This is shown in the examples in this chapter. If IQ8–IQ11 are available, the fetcher tries to initiate a fetch every cycle. Because the two fetch stages are pipelined, as many as four instructions can reach the IQ every cycle. However, the fetcher normally initiates a fetch only if IQ8–IQ11 are empty.

Typically, instructions are fetched from the L1 instruction cache, but as many as four instructions from a targeted stream can be fetched on a BTIC hit in 2 clock cycles, allowing the next four instructions to be fetched from the instruction cache with no idle cycles.

Branch instructions in IQ0–IQ7 are identified and forwarded to the BPU directly for immediate execution.

If a branch is predicted as taken, all instructions are flushed from the IQ in the next clock cycle. If the branch is unconditional or if the specified conditions are already known, the branch can be resolved immediately. That is, the branch direction and target address are known and instruction fetching can continue from the correct location. Otherwise, the branch direction or branch target address must be predicted. The MPC7450 offers several resources to resolve branch instructions and to improve the accuracy of branch predictions. These include the following:

- Branch target instruction cache (BTIC)—The 128-entry, four-way-associative BTIC, shown in Figure 6-5, holds as many as four branch target instructions in each entry, so when a branch is encountered in a repeated loop, usually the first four instructions in the target stream can be fetched into the instruction queue on the next two clock cycles. The BTIC can be disabled and invalidated through bits in HID0.

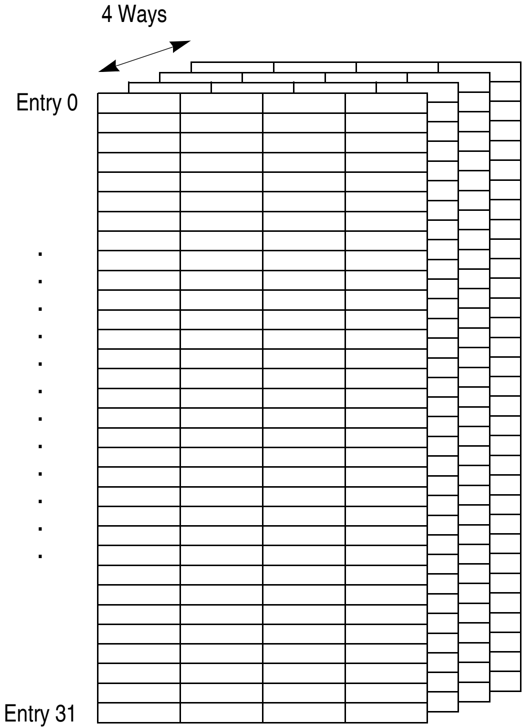


Figure 6-5. BTIC Organization

BTIC entries are indexed not from the address of the first target instruction but from the address of the branching instruction, so multiple branches sharing a target generate duplicate BTIC entries. Each entry can hold as many as four instructions, depending on where the first target instruction falls in the cache block.

As with other aspects of MPC7450 instruction timing, BTIC operation is optimized for cache-line alignment. If the first target instruction is one of the first five instructions in the cache block, the BTIC entry holds four instructions. If the first target instruction is the last instruction before the cache block boundary, it is the only instruction in the corresponding BTIC entry. If the next-to-last instruction in a cache block is the target, the BTIC entry holds two valid target instructions, as shown in Figure 6-6.

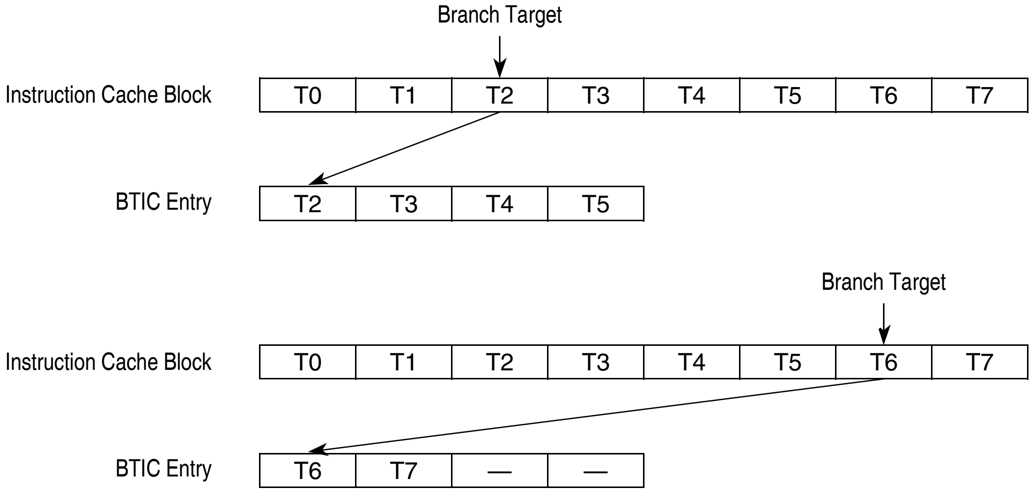


Figure 6-6. Alignment of Target Instructions in BTIC

BTIC ways are updated using a FIFO algorithm.

Note that the entire BTIC is invalidated if translation changes (for example, a **tlbie** instruction executes**,** a TLB or BAT is updated, or an exception puts the processor in real mode) or if an **icbi** instruction invalidates an instruction cache block.

- Dynamic branch prediction—The 2048-entry branch history table (BHT) is implemented with 2 bits per entry for four levels of prediction—not taken, strongly not, and strongly taken. Whether a branch instruction is taken or not taken can change the strength of the next prediction. Dynamic branch prediction is not defined by the PowerPC architecture.

  To reduce aliasing, only predicted branches update BHT entries. Dynamic branch prediction is enabled by setting HID0[BHT]; otherwise, static branch prediction is used.
- Static branch prediction—Static branch prediction is defined by the PowerPC architecture and involves encoding the branch instructions. See 6.4.1.3.1, “Static Branch Prediction.”
- Link stack registers—The MPC7450 also avoid stalls by implementing an eight-entry branch link stack. As many as eight levels of **bclr**/branch-and-link pairs can be held and the **bclr** target address can be predicted from the link stack rather than requiring a stall until the **ld**/**mtlr** subroutine restore sequence completes.

  The link register and rules required for correct use are as described in the *Programming Environments Manual*.

  Correct use of the link stack requires that computed GOTO statements consist of the **mtctr**/**bcctr** instruction pair rather than the **mtlr**/**bclr** pair, which lowers link stack performance by requiring an expensive mispredict drain/flush as well as clearing the link stack to its initial empty state.

> Attempting link stack and conditional branch prediction on the same instruction can affect performance. It may be necessary to avoid conditional **bclr** instructions because the BPU stalls execution until either the directional condition or the target address (LR) is resolved. Additionally, even if both are resolved when the conditional **bclr** is presented to the BPU, the conditional **bclr** takes 2 cycles to execute. Thus, except for code size, following a conditional branch (**bc**) with an unconditional **bclr** may be preferable to a conditional **bclr**.

Branch instructions that do not update the LR or CTR can be removed from the instruction stream, as described in Section 6.4.1.1, “Branch Folding and Removal of Fall-Through Branch Instructions.” Branch instructions that update the LR or CTR are treated as if they require dispatch, even though they are not dispatched to one of the issue queues. They must be assigned CQ entries to ensure that the CTR and LR are updated sequentially. The dispatch rate is affected by the serializing behavior of some instructions and on the availability of issue queues, execution units, rename registers, and CQ entries. Instructions are dispatched in program order; an instruction in IQ1 cannot be dispatched ahead of one in IQ0.

Figure 6-7 shows instruction paths.

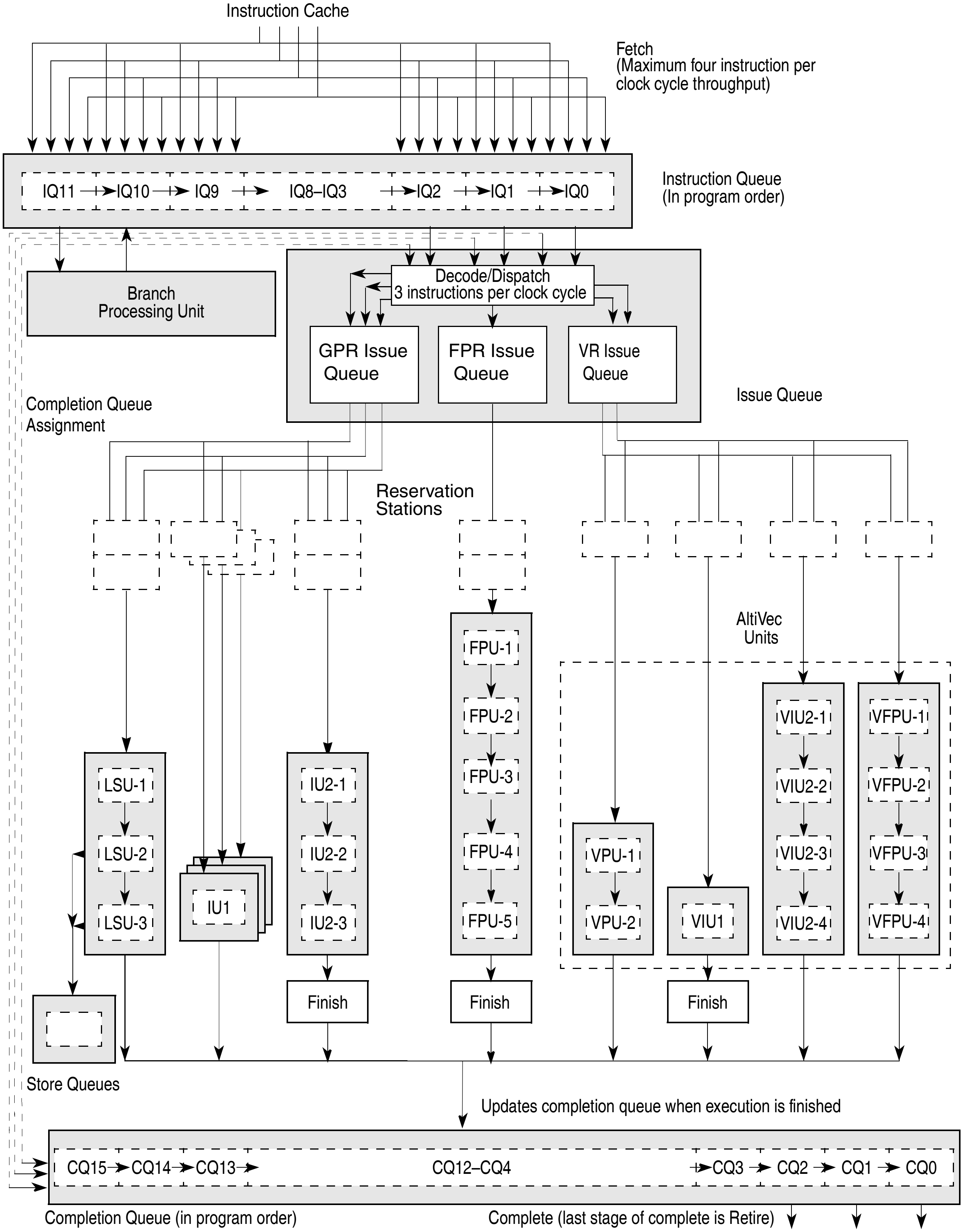


Figure 6-7. Instruction Flow Diagram

6.3.2 Instruction Fetch Timing

Instruction fetch latency depends on whether the fetch hits the BTIC, the instruction MMU, the L1 instruction cache, the on-chip L2-cache, or the off-chip L3 cache (if one is implemented). If no cache hit occurs, a memory transaction is required in which fetch latency is affected by bus traffic and bus clock speed. These issues are discussed further in the following sections.

6.3.2.1 Cache Arbitration

When the instruction fetcher requests instructions from the instruction cache, two things may happen. If the access hits the instruction cache and the cache is idle, the instructions arrive two clock cycles later. However, if the cache or MMU is busy due to a higher priority operation, such as a **tlbie**, **icbi**, or a cache line reload, instructions cannot be fetched until that operation completes.

6.3.2.2 Cache Hit

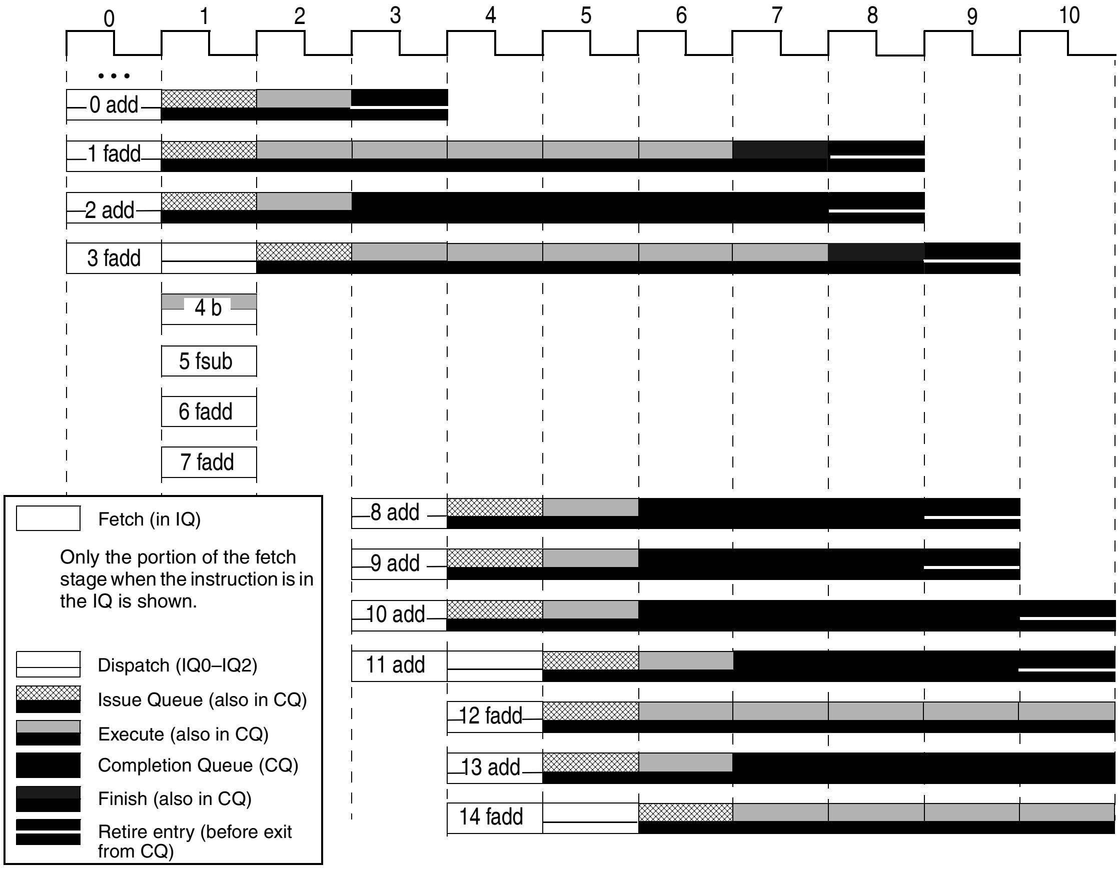
If the instruction fetch hits the instruction cache, it takes only two clock cycles after the request for as many as four instructions to enter the IQ. Note that the cache is not blocked to internal accesses during a cache reload (hits under misses). The critical quad word is written simultaneously to the cache and forwarded to the requesting unit, minimizing stalls due to load delays. Note that the cache allows a hit under one miss and is only blocked by a cache line reload for the cycle when the cache write happens. So, if a cache miss is discarded by a misprediction and a new fetch hits, the cache allows instructions to come back. As many as four instructions are pipelined from the cache per cycle until the original miss comes back and a cache reload is performed, which blocks the cache for 1 cycle.

Figure 6-8 shows a simple example of instruction fetching that hits in the on-chip cache. This example uses a series of integer add instructions and double-precision, floating-point add instructions to show the following:

- How the number of instructions to be fetched is determined
- How program order is maintained by the IQ and CQ
- How instructions are dispatched, issued and completed
- How the FPU and IU2 pipelines function

Note that for the following instruction sequence instruction 0 is assumed to be the first instruction in the cache block. Also, there are no critical dependencies between instructions.

```
0        add
1        fadd
2        add
3        fadd
4        b 8
5        fsub
6        fadd
7        fadd
8        add
9        add
10       add
11       add
12       fadd
13       add
14       fadd
15       .
16       .
17       .
```



Instruction Queue

IQ11											
•											
•											
IQ4		7									
IQ3	3	6		11	14						
IQ2	2	5		10	13						
IQ1	1	4		9	12						
IQ0	0	3		8	11	14					

Completion Queue

CQ15											
•											
•											
CQ9							14	14	14		
CQ8						13	13	13	13		
CQ7						12	12	12	12	14	
CQ6						11	11	11	11	13	
CQ5					10	10	10	10	10	12	
CQ4					9	9	9	9	9	11	14
CQ3			3	3	8	8	8	8	8	10	13
CQ2		2	2	2	3	3	3	3	3	9	12
CQ1		1	1	1	2	2	2	2	2	8	11
CQ0		0	0	0	1	1	1	1	1	3	10

Figure 6-8. Instruction Timing—Cache Hit

The instruction timing for this example is described cycle-by-cycle as follows:

0. In cycle 0, instructions 0–3 are fetched from the instruction cache and are available in the IQ. These instructions are placed in IQ0–IQ3. Instructions 0–2 are dispatched at the end of this cycle to the GIQ, FIQ, and GIQ, respectively. As they are dispatched, they are also allocated in the bottom three entries of the CQ.
1. Instructions 0 and 2 are issued to two of the IU1s and instruction 1 is issued to the FPU. Note that because these instructions were dispatched in the last cycle, they have been assigned sequential positions in the CQ. In this case, instructions 0–2 are in the bottom three entries of the CQ. Instruction 3 is in IQ0 from which it is dispatched to the FIQ at the end of this cycle. Instructions 4–7 have arrived from the instruction cache at the end of the last cycle, which are also allocated in the IQ1–IQ4. The eight-instruction cache block boundary falls between instructions 7 and 8. Instruction 4 (the unconditional branch) is executed in this cycle and is immediately resolved as taken and therefore can be folded from the IQ.
2. Instructions 0 and 2 execute in the single-stage IU1s. Instruction 1 proceeds to the first of the five FPU stages. Instruction 3 is in the fourth entry of the CQ and is issued from the FIQ to the FPU. Because instruction 4 was an unconditional taken branch, it is folded from the IQ and instructions 5–7 are discarded. No new instructions are available because of the latency of the BTIC and instruction cache fetching.
3. Instruction 0 is retired at the end of this cycle. Instruction 1 proceeds to the second FPU stage. Instruction 2 has finished executing but must remain in the CQ until instruction 1 retires. Instruction 3 replaces instruction 1 in the first FPU stage. Instructions 8–11 are arrived from the BTIC and are also allocated in the bottom four entries of the IQ. Instructions 8-10 are dispatched to the GIQ at the end of this cycle.
4. Instruction 0 is retired from the CQ, so the remaining instructions are shifted down in the CQ. Instructions 1 and 3 are in the third and second FPU stages, respectively. Instruction 2 is still waiting in the CQ for instruction 1 to retire. Instructions 8–10 take positions in CQ. Instructions 12–14, which arrived from the instruction cache at the end of last cycle, are now in the IQ1–IQ3. Instructions 11–13 are dispatched at the end of this cycle.
5. Instructions 1 and 3 occupy to the fourth and third FPU stages, respectively. Instruction 2 waits in the CQ for instruction 1 to retire. Instructions 8–10 execute in the IU1s. Instructions 11–13 are issued and occupy CQ6–CQ8. Instruction 14 is dispatched at the end of this cycle.
6. Instruction 1 reaches the last FPU stage, while instruction 3 advances through the FPU pipeline. Instructions 2, 8, 9, and 10 wait in the CQ. Instructions 11–13 begin executing. Instruction 14 is issued to the FPU and occupies CQ9.
7. Instruction 1 occupies the FPU finish stage, while instruction 3 moves to the last FPU stage. Instructions 2, 8–11, and 13 wait in the CQ. Instruction 12 advances through the FPU pipeline, while instruction 14 moves to the first FPU stage.
8. Instructions 1 and 2 are retired at the end of this cycle. Instruction 3 moves to the FPU finish stage. Instructions 8–11 and 13 wait in the CQ. Instructions 12 and 14 advance through the FPU.
9. Because instructions 1 and 2 were retired in the last clock cycle, the remaining instructions shift down in the CQ and instructions 3, 8, and 9 are retired at the end of this clock cycle. Instructions 10, 11, and 13 wait in the CQ. Instructions 12 and 14 continue through the FPU.

10. Instructions 3, 8, and 9 retired in the last clock cycle, so the remaining instructions shift down in the CQ. Instructions 10 and 11 are retired at the end of this clock cycle, while instruction 13 waits in the CQ. Instructions 12 and 14 continue through the FPU.

6.3.2.3 Cache Miss

Figure 6-9 uses a similar instruction sequence as in Section 6.3.2.2, “Cache Hit,” but here the fetch misses the caches and requires a bus access. Note that because the target instruction is not in the L1 cache, it cannot be in the BTIC. A 5:1 processor:bus clock ratio is used. The difference in the following code sample from that shown in Figure 6-8 is that the branch instruction 4 jumps to instruction 6 instead of instruction 8, and also cache block boundary lies between instructions 5 and 6. The following example assumes the minimum possible cycles for a bus transaction to occur, but any real system is likely to see further delays due to real DRAM latencies.

```
0        add
1        fadd
2        add
3        fadd
4        b 6
5        fsub
6        fadd
7        fadd
8        add
9        add
10       add
11       add
12       fadd
13       add
14       fadd
15       .
16       .
17       .
```

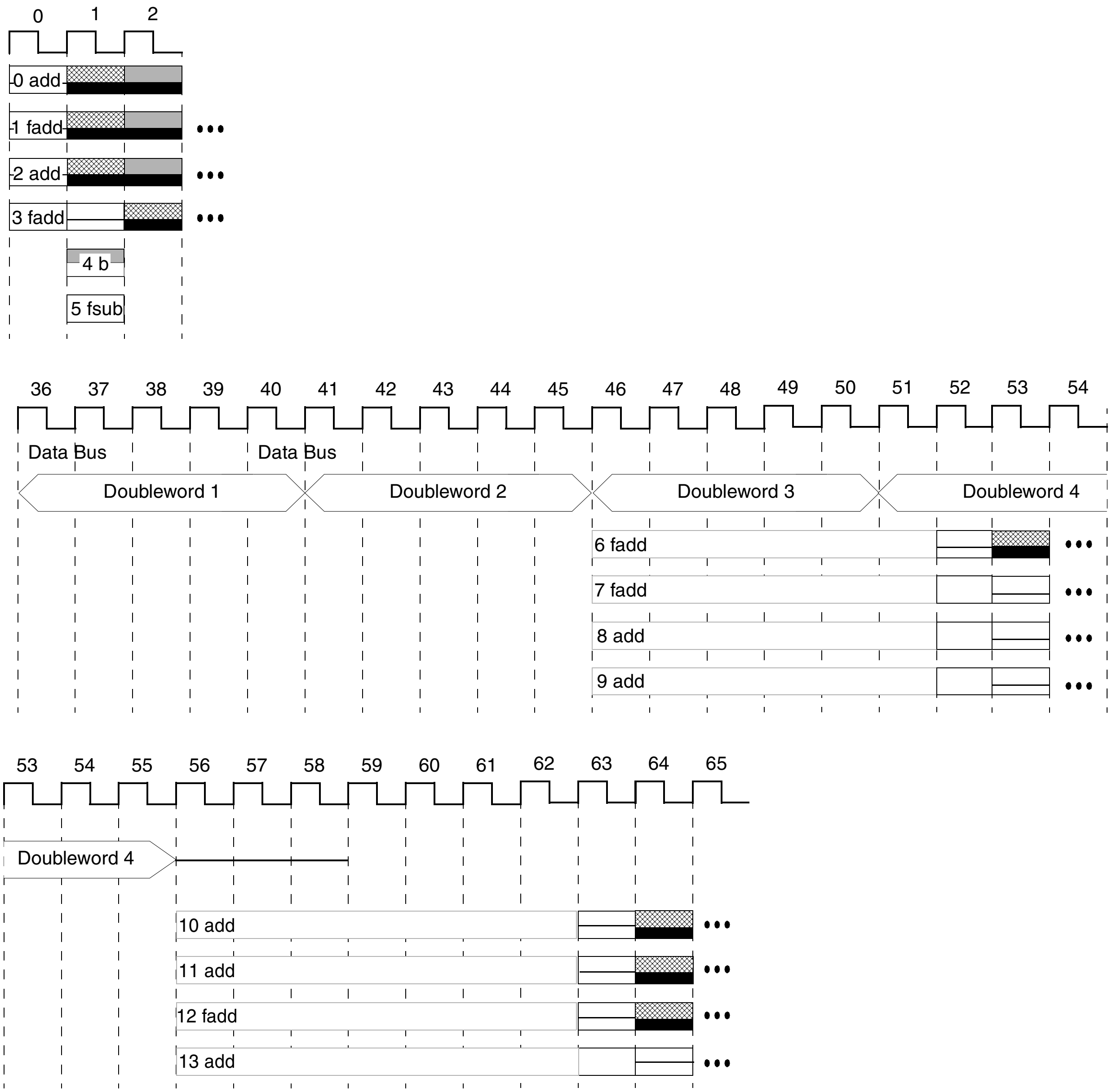


Figure 6-9. Instruction Timing—Cache Miss

A cache miss extends the latency of the fetch stage, so the fetch stage represents the time the instruction spends in the IQ and the time required for the instruction to be loaded from system memory, beginning in clock cycle 2.

The first four instructions follow the same pattern as in the cache hit example. The cache miss occurs in clock cycle 3, so the fetcher initiates a four-beat burst transaction to system memory. The critical quad word, which contains instructions 6–9, arrives in the first two beats and is forwarded to the IQ after the second beat. In clock cycle 52, these instructions are forwarded both to the instruction cache and to the instruction fetcher. Instructions 10–13 arrive 11 cycles later in clock cycle 63. Assuming that instructions 14–17 are also not in any of the caches, they are available to the IQ in cycle 102. The second half of this cache block (instructions 18–21) arrives 11 cycles later (cycle 113).

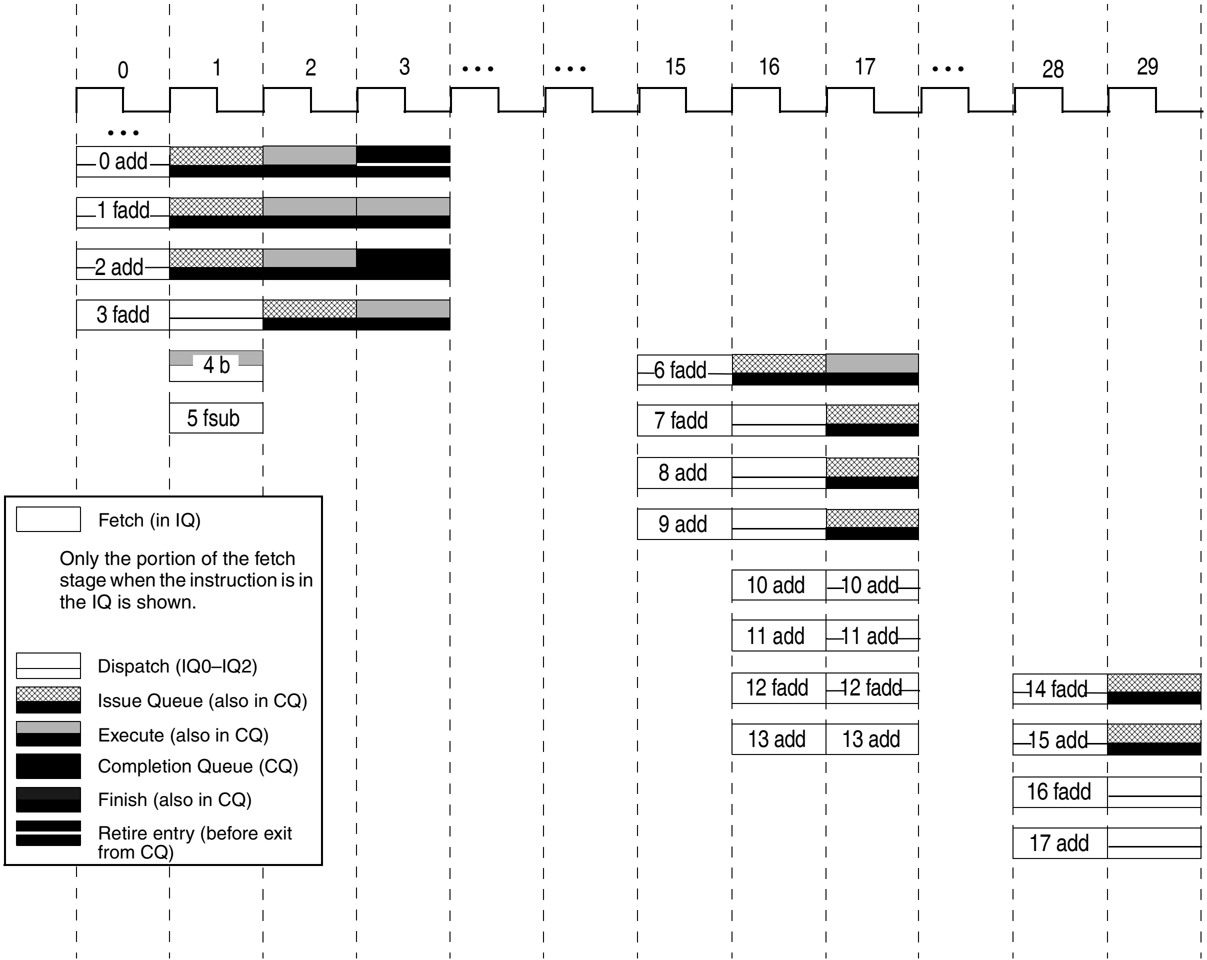
6.3.2.4 L2 Cache Access Timing Considerations

If an instruction fetch misses the BTIC and the on-chip instruction cache, the MPC7450 next looks in the L2 cache. If the requested instructions are there, they are burst into the MPC7450 in much the same way as shown in Figure 6-6.

The example shown is for the fastest possible L2 response. Other factors can effect this latency, including whether the L2 cache is busy with other operations, like servicing a load request from the LSU.

6.3.2.4.1 Instruction Cache and L2 Cache Hit

Full fetch latency (from fetch1 to arrival into IQ) for an instruction cache miss/L2 cache hit is 13 cycles. Note that an L2 hit provides a full cache line, so the instructions are fetched into the IQ back to back (see cycles 15 and 16 in Figure 6-10). This is the difference between the L3 cache and main memory bus, which use quad-word forwarding. Figure 6-10 shows the same code sequence as Figure 6-9.



Instruction Queue

IQ11												
•												
IQ6								13				
IQ5								12				
IQ4								11				
IQ3	3						9	10	13		17	
IQ2	2	5					8	9	12		16	
IQ1	1	4					7	8	11		15	17
IQ0	0	3					6	7	10		14	16

Completion Queue

CQ15												
•												
CQ4												
CQ3			3	3					9			
CQ2		2	2	2	3				8			
CQ1		1	1	1	2				7			15
CQ0		0	0	0	1			6	6			14

Figure 6-10. Instruction Timing—Instruction Cache Miss/L2 Cache Hit

6.3.2.4.2 Instruction Cache Miss/L3 Cache Hit

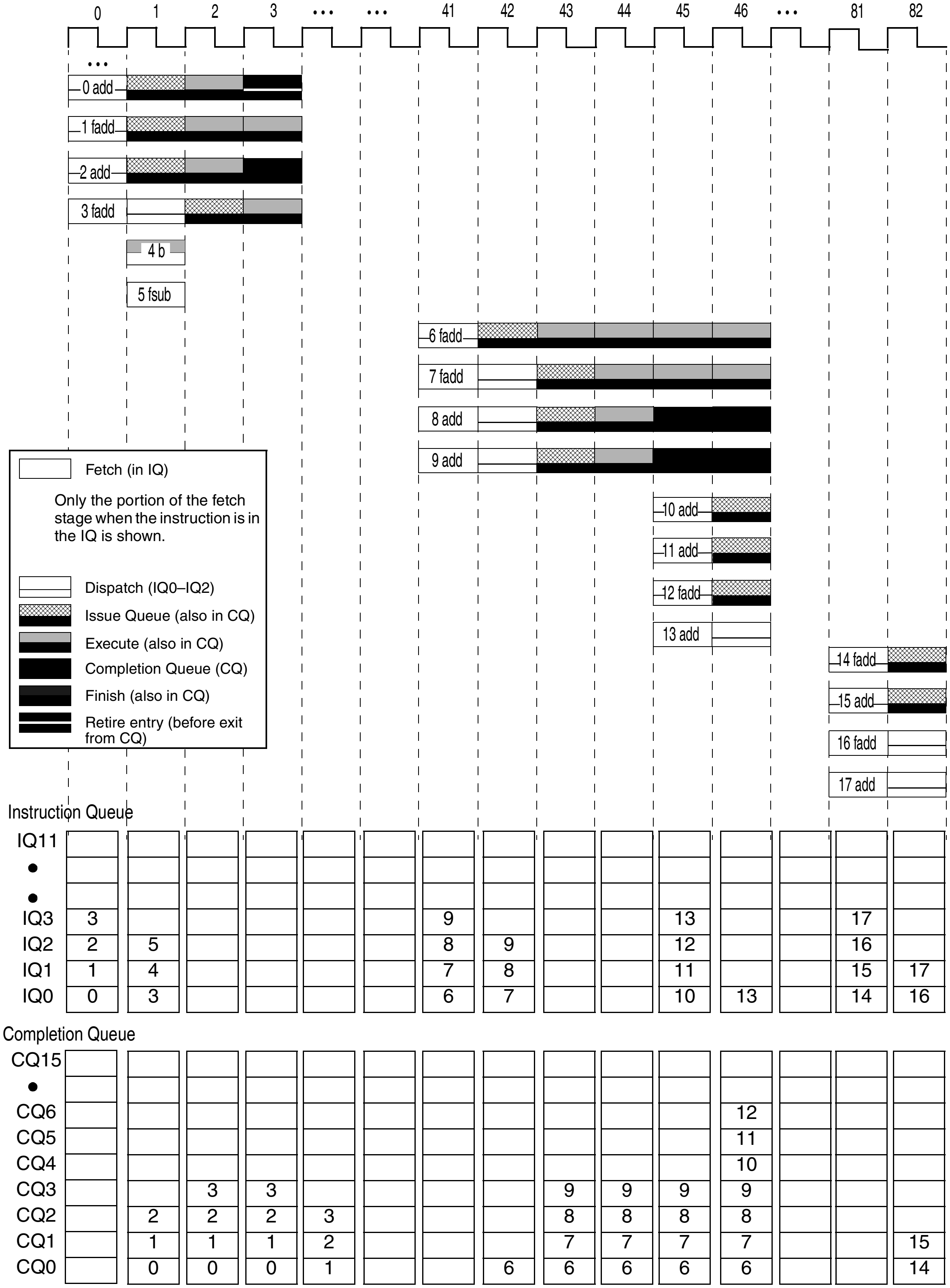
Figure 6-11 shows the same code sequence as Figure 6-9, except that there is an instruction cache/L2 cache miss and an L3 cache hit. Note that L3 cache is not supported on the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448. This example assumes the following:

- Use of DDR SRAM
- 4:1 bus ratio
- 5-cycle L3 clock sample point
- 0-cycle L3 processor-clock sample point

For an L3 hit, full fetch latency (from fetch1 to the arrival into IQ) is 39 cycles. Note that an L3 forwards a quad word at a time, followed by a full cache-line reload. The time for the first beat is affected by three major parameters—miss time, SRAM time, and forwarding time. Miss and forwarding times are constant and do not interfere with traffic. The SRAM time takes 23 cycles; the formula for calculating the SRAM time is as follows:

$$\text{synchronization to L3 clock} + (\text{SRAM\_RATIO}) \times (\text{L3 clock sample point}) + (\text{L3 P-clock sample})$$

For the example shown in Figure 6-11, the synchronization to L3 clock is 3 cycles.



Only the portion of the fetch stage when the instruction is in the IQ is shown.

Instruction Queue

	0	1	2	3	•••	•••	41	42	43	44	45	46	•••	81	82
IQ11															
•															
•															
IQ3	3						9				13			17	
IQ2	2	5					8	9			12			16	
IQ1	1	4					7	8			11			15	17
IQ0	0	3					6	7			10	13		14	16

Completion Queue

	0	1	2	3	•••	•••	41	42	43	44	45	46	•••	81	82
CQ15															
•															
CQ6												12			
CQ5												11			
CQ4												10			
CQ3			3	3					9	9	9	9			
CQ2		2	2	2	3				8	8	8	8			
CQ1		1	1	1	2				7	7	7	7			15
CQ0		0	0	0	1			6	6	6	6	6			14

Figure 6-11. Instruction Timing—Instruction Cache Miss/L3 Cache Hit

6.3.3 Dispatch, Issue, and Completion Considerations

Several factors affect the core's ability to dispatch instructions at a peak rate of three per cycle—the mix of instructions and the availability of issue queues, destination rename registers, and CQ entries. Several of these factors are shown in the previous instruction timing examples.

Although as many as three instructions can be dispatched in parallel, they cannot be dispatched out-of-order; for example, an instruction in IQ1 cannot be dispatched unless the instruction in IQ0 can also dispatch.

To reduce issue unit stalls due to data dependencies, the LSU, IU2, and FPU have two-entry reservation stations; the other execution units have single-entry reservation stations. If a data dependency keeps an instruction from starting execution, that instruction is issued to a reservation station and the rename registers are assigned, eliminating the issue queue stalls. Execution begins during the same clock cycle that the rename buffer is updated with the data on which the instruction depends.

The issue queues allow the MPC7450 to dispatch decoded instructions even if execution units are busy. The dispatcher also allocates rename registers, locates source operands, and assigns rename registers for destination operands. The issue logic reads operands from register files and rename registers, and supplies rename tags for unavailable operands. The issue stage routes instructions to the proper execution unit. Execution begins when all operands are available, the instruction is in the bottom reservation station, and any execution serialization requirements are met.

The CQ maintains program order after instructions are dispatched, guaranteeing in-order completion and a precise exception model. In-order completion ensures the correct architectural state when the MPC7450 must recover from a mispredicted branch or exception.

Instruction state and other information required for completion are kept in the 16-entry, FIFO completion queue. Instructions cannot be retired ahead of previous instructions, as shown in Section 6.3.2.2, “Cache Hit,” and Section 6.3.2.3, “Cache Miss.”

Instructions are retired much as they are dispatched. As many as three instructions can be retired simultaneously, but instructions cannot be retired out-of-order. Note the following:

- Instructions must be non-speculative to complete.
- As many as three rename registers can be updated in a given register file. For example, a sequence of three **lfdu** instructions, which requires three GPR rename registers and three FPR rename registers, can complete in one cycle. However, the sequence **lwzu**, **add**, **add** requires four GPR rename registers, so only the first two instructions can complete in a cycle.

Program-related exceptions are signaled when the instruction causing the exception reaches CQ0. Previous instructions are allowed to complete before the exception is taken, which ensures that any exceptions those instructions may cause are 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 MPC7450 provides rename registers for holding instruction results before the completion commits them to the architecture-defined register. The GPRs, FPRs, and VRs each have 16 rename registers. The CR, LR, and CTR have 1 rename register.

When an instruction is dispatched, a rename register (or registers) are allocated for the results of that instruction. If an instruction is issued to a reservation station because of a data dependency, issue logic also provides a tag to the execution unit identifying the rename register that forwards the required data at completion. Execution can begin when source data reaches the rename register.

Results from rename registers are transferred to the architecture-defined registers in the write-back stage. Renames are also deallocated in the write-back stage.

If branch prediction is incorrect, instructions after the branch are flushed from the CQ and any results of those instructions are flushed from the rename registers.

6.3.3.2 Instruction Serialization

Although the MPC7450 core can dispatch and complete three instructions per cycle, serializing instructions limit dispatch and completion to one instruction per cycle. There are three basic types of instruction serialization:

- Execution serialization—Execution-serialized instructions are issued and held in the functional unit's reservation station. They do not execute until all prior instructions have completed. A functional unit holding an execution-serialized instruction does not accept further instructions from the issue queues. For example, execution serialization is used for instructions that modify non-renamed resources. Results from these instructions are generally not available or forwarded to subsequent instructions until the instruction completes.
- Refetch serialization—Refetch-serialized instructions force refetching of subsequent instructions after completion. Refetch serialization is used when an instruction has changed or may change a particular context needed by subsequent instructions. Examples include **isync**, **sc**, **rfi**, **mtspr[XER]**, and any instruction that toggles the summary-overflow (SO) bit.
- Store serialization—(Applicable to stores and some LSU instructions that access the data cache.) Store-serialized instructions are dispatched and held in the LSU's finished store queue. They are not committed to memory until all prior instructions have completed. While a store-serialized instruction waits in the finished store queue, other load/store instructions can be freely executed. Store-serialized instructions complete only from CQ0, so only one store-serialized instruction can complete per cycle, although non-serialized instructions can complete in the same cycle as a store-serialized instruction. In general, all stores and cache operation instructions are store-serialized.

6.4 Execution Unit Timings

The following sections describe instruction timing considerations within each of the respective execution units in the MPC7450.

6.4.1 Branch Processing Unit Execution Timing

Flow control operations (conditional branches, unconditional branches, and traps) are sometimes expensive to execute in most machines because they may disrupt normal instruction flow. When program flow changes, the IQ must be reloaded with the target instruction stream. Previously issued instructions continue executing while the new instruction stream makes its way into the IQ, but depending on whether the target instruction is in the BTIC, instruction cache, L2 cache, off-chip L3 cache, or in system memory, opportunities may be missed to execute instructions, as the examples in Section 6.3.2.3, “Cache Miss,” show. Note that L3 cache is not supported on the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448.

The penalties associated with MPC7450 flow control operations are minimized by performance features such as branch folding, removal of fall-through branch instructions, BTIC, dynamic branch prediction (implemented in the BHT), three-level branch prediction, an eight-entry link stack, and the implementation of nonblocking caches. Timing for branch instruction execution is affected by whether the following occur:

- The branch is taken.
- Instructions in the target stream, typically the first four instructions in the target stream, are in the BTIC.
- The target instruction stream is in the on-chip cache.
- The branch is predicted.
- The prediction is correct.

6.4.1.1 Branch Folding and Removal of Fall-Through Branch Instructions

After a branch enters the IQ, the BPU immediately begins to decode it and tries to resolve it. Except those that update the LR or CTR, most branch instructions are removed from the instruction flow before they take a position in the CQ.

Branch folding occurs either when a branch is predicted taken or is resolved taken (as in the case with unconditional branches). The cycle after branch execution, the BPU folds the branch out of the instruction stream (removes the branch from the IQ) and also flushes instructions in the IQ after the branch.

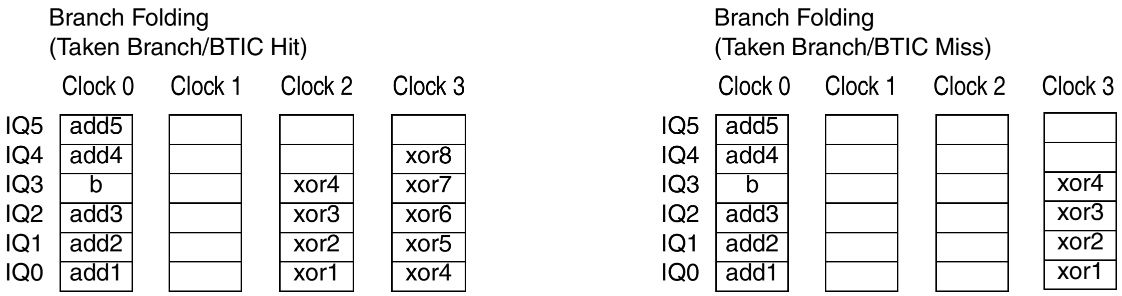
Figure 6-12 shows branch folding with a BTIC hit and with a BTIC miss/instruction cache hit. Here a **b** instruction encountered in a series of **add** instructions is resolved as taken. What happens on the next clock cycle depends on whether the target instruction stream is cached or if a bus transfer is required.

If there is a BTIC hit, on the next clock cycle the **b** instruction is folded and the instructions after the branch (**add**4 and **add**5) are flushed. On the cycle after that (clock2), instructions **xor**1–**xor**4 have arrived from the BTIC, and **xor**1–**xor**3 are dispatched. In clock 3, **xor**5–**xor**8 arrive.

If the target instructions are not in the BTIC, it takes 2 cycles (fetch1 and fetch2) to attempt to fetch the first four instructions from the instruction cache. These instructions arrive and are ready for dispatch in clock 3.

The effect of the taken branch on the instruction supply is known as the branch-taken bubble. For the BTIC hit case, there is a 1-cycle bubble (clock1). For the BTIC miss case, there is a 2-cycle bubble (clocks 1 and 2).

If the fetch misses all of the caches, a system memory access is required, the latency of which depends on factors such as processor:bus clock ratio. In most cases, an L3 cache or memory access indicates that execution units remain idle, as shown in Section 6.3.2.3, “Cache Miss.”



Branch Folding
(Taken Branch/BTIC Hit)

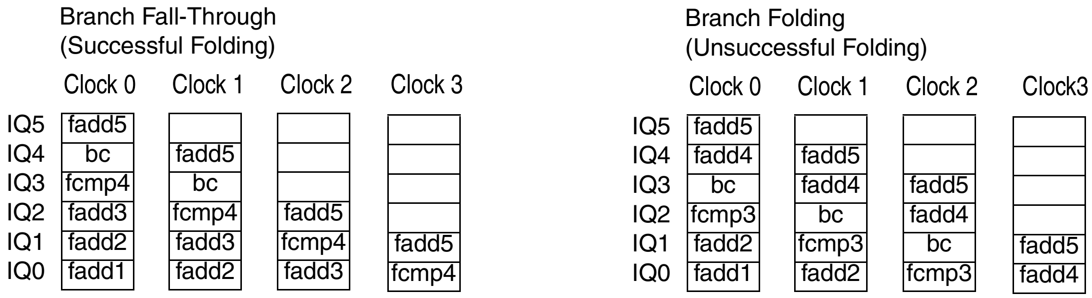
	Clock 0	Clock 1	Clock 2	Clock 3
IQ5	add5			
IQ4	add4			xor8
IQ3	b		xor4	xor7
IQ2	add3		xor3	xor6
IQ1	add2		xor2	xor5
IQ0	add1		xor1	xor4

Branch Folding
(Taken Branch/BTIC Miss)

	Clock 0	Clock 1	Clock 2	Clock 3
IQ5	add5			
IQ4	add4			
IQ3	b			xor4
IQ2	add3			xor3
IQ1	add2			xor2
IQ0	add1			xor1

Figure 6-12. Branch Folding

Figure 6-13 shows the attempted removal of fall-through branch instructions, which can occur when a branch is not taken or is predicted as not taken.



Branch Fall-Through
(Successful Folding)

	Clock 0	Clock 1	Clock 2	Clock 3
IQ5	fadd5			
IQ4	bc	fadd5		
IQ3	fcmp4	bc		
IQ2	fadd3	fcmp4	fadd5	
IQ1	fadd2	fadd3	fcmp4	fadd5
IQ0	fadd1	fadd2	fadd3	fcmp4

Branch Folding
(Unsuccessful Folding)

	Clock 0	Clock 1	Clock 2	Clock3
IQ5	fadd5			
IQ4	fadd4	fadd5		
IQ3	bc	fadd4	fadd5	
IQ2	fcmp3	bc	fadd4	
IQ1	fadd2	fcmp3	bc	fadd5
IQ0	fadd1	fadd2	fcmp3	fadd4

Figure 6-13. Removal of Fall-Through Branch Instruction

A not-taken branch instruction stays in the IQ for at least the cycle after execution. If it does not update the LR or CTR and is in IQ3–IQ7 in the cycle after execution, it can be removed from the IQ. In Figure 6-13, the branch is predicted as not taken and executes in cycle 0 in both cases. When the branch is in IQ3, it is removed from the IQ (clock 2). In the unsuccessful case, it reaches IQ2 in clock 2, from which it cannot be removed and so must be dispatched.

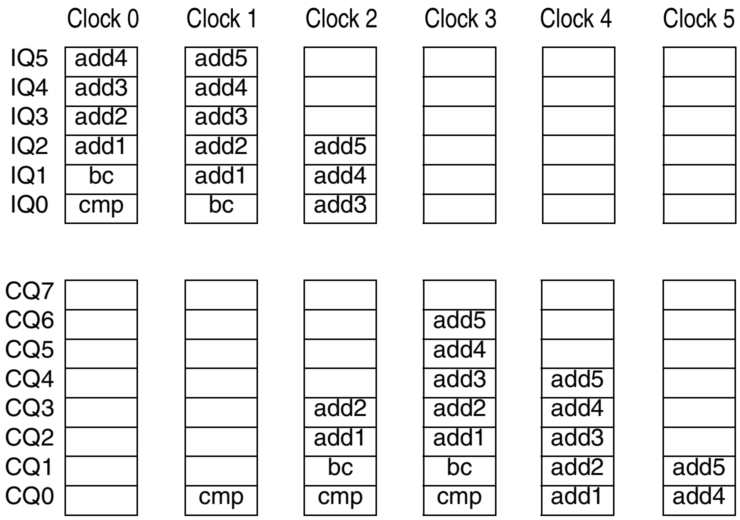
When a correctly predicted taken branch instruction is detected before reaching a dispatch position, folding the branch instruction and flushing any instructions from the incorrect path may eliminate any latency required for control flow. Execution proceeds as though the branch were never there. However, in many cases the branch-taken bubble may waste a few dispatch opportunities.

The advantage of removing not-taken branch instructions (fall-through) is slightly less than that of branch folding. Although the branch may be removed from the instruction stream and not require a dispatch slot, if the branch reaches IQ0–IQ2 it requires both a dispatch slot and a CQ entry.

6.4.1.2 Branch Instructions and Completion

As described in the previous section, instructions that do not update the LR or CTR are often removed from the instruction stream before they reach the CQ. However, branch instructions that update the architecture-defined LR and CTR must write back in program order like the instructions that update the FPRs, GPRs, and VRs.

Executed branch instructions that are not removed are not sent to an issue queue at dispatch but are assigned a CQ slot, as shown in Figure 6-14.



	Clock 0	Clock 1	Clock 2	Clock 3	Clock 4	Clock 5
IQ5	add4	add5				
IQ4	add3	add4				
IQ3	add2	add3				
IQ2	add1	add2	add5			
IQ1	bc	add1	add4			
IQ0	cmp	bc	add3			

	Clock 0	Clock 1	Clock 2	Clock 3	Clock 4	Clock 5
CQ7						
CQ6				add5		
CQ5				add4		
CQ4				add3	add5	
CQ3			add2	add2	add4	
CQ2			add1	add1	add3	
CQ1			bc	bc	add2	add5
CQ0		cmp	cmp	cmp	add1	add4

Figure 6-14. Branch Completion (LR/CTR Write-Back)

In this example, the **bc** that executes in clock cycle 0 depends on **cmp** and is predicted as not taken. Because the branch executes in clock cycle 0, it cannot be dispatched. Because the branch is in IQ0 in the cycle after execution, it cannot be folded and is dispatched with **add1** and **add2**. The **cmp** executes in clock cycle2 and the branch resolves as correctly predicted in clock cycle 3.

Also at the end of clock cycle 3, **cmp** and **bc** retire. Even if **add1** were finished, it could not retire because the **bc** is resolving in this cycle. In the cycle after branch resolution, cycle 4, **add1–add5** are marked as non-speculative and **add1–add3** are allowed to retire.

6.4.1.3 Branch Prediction and Resolution

The MPC7450 supports the following two types of branch direction prediction:

- Static branch prediction—This is defined by the PowerPC architecture as part of the encoding of branch instructions.
- Dynamic branch prediction—This is a processor-specific mechanism implemented in hardware (in particular the branch history table, or BHT) that monitors branch instruction behavior and maintains a record from which the next occurrence of the branch instruction is predicted.

When a conditional branch direction cannot be resolved due to a CR or CTR data dependency, the BPU predicts whether it will be taken and instruction fetching proceeds down the predicted path. If the prediction is wrong, subsequent instructions and their results are purged. Instructions ahead of the predicted branch proceed normally, and instruction fetching resumes along the correct path.

The MPC7450 executes through three prediction levels. Instructions from all three unresolved branches are allowed to execute but cannot complete until all older branches are resolved. If three predicted branches are outstanding, no further conditional branches can be processed (although BPU processes any unconditional branch). When one or more of the three previous conditional branches is resolved, the BPU can begin processing new conditional branches.

The number of instructions that can be executed after the issue of a predicted branch instruction is limited by the fact that no instruction executed after a predicted branch can update the register files or memory until the branch is resolved. That is, instructions may be issued and executed, but cannot reach the write-back stage. When an instruction in a predicted branch stream finishes, it does not write back its results to the architecture-defined registers until the branch is resolved, which may cause a stall in the CQ.

In case of a misprediction, the MPC7450 can easily redirect its machine state because the programming model has not been updated. If a branch is mispredicted, all instructions dispatched after the predicted branch instruction are flushed from the CQ and any results are flushed from the rename registers.

If the search for the branch target hits in the BTIC, one of the four target instructions is available in the IQ 2 cycles later (shown in Figure 6-12). The BTIC is described in detail in Section 6.3.1, “General Instruction Flow.”

In some situations, an instruction sequence creates dependencies that keep a branch instruction from being resolved immediately, thereby delaying execution of the subsequent instruction stream based on the prediction. The instruction sequences and the resulting action of the branch instruction are as follows:

- An **mtspr**(LR) followed by a **bclr**—The link stack is used to predict the instruction target address.
- An **mtspr**(LR) followed by a conditional bclr—The BPU stalls this branch until either the LR becomes available or the required CR/CTR data becomes available. The BPU can predict a branch either on a direction basis (CR/CTR) or an address basis (LR) but not both simultaneously. Note that a conditional **bclr** also requires 2 cycles to execute in the BPU.
- An **mtspr**(CTR) followed by a **bcctr**—Fetching stops and the branch waits for the **mtspr** to execute.
- A fourth conditional branch—A fourth conditional branch is encountered while three branches are unresolved (based on CR and CTR direction predictions or LR address predictions). The fourth **bc** is not executed and the BPU stalls. This normally forces the fetcher to stall a cycle or two later

when the IQ fills up behind the stalled branch. This stall continues until at least one unresolved branch resolves. Note that branch conditions can be a function of the CTR and the CR; if the CTR condition is sufficient to resolve the branch, any CR dependency is ignored.

6.4.1.3.1 Static Branch Prediction

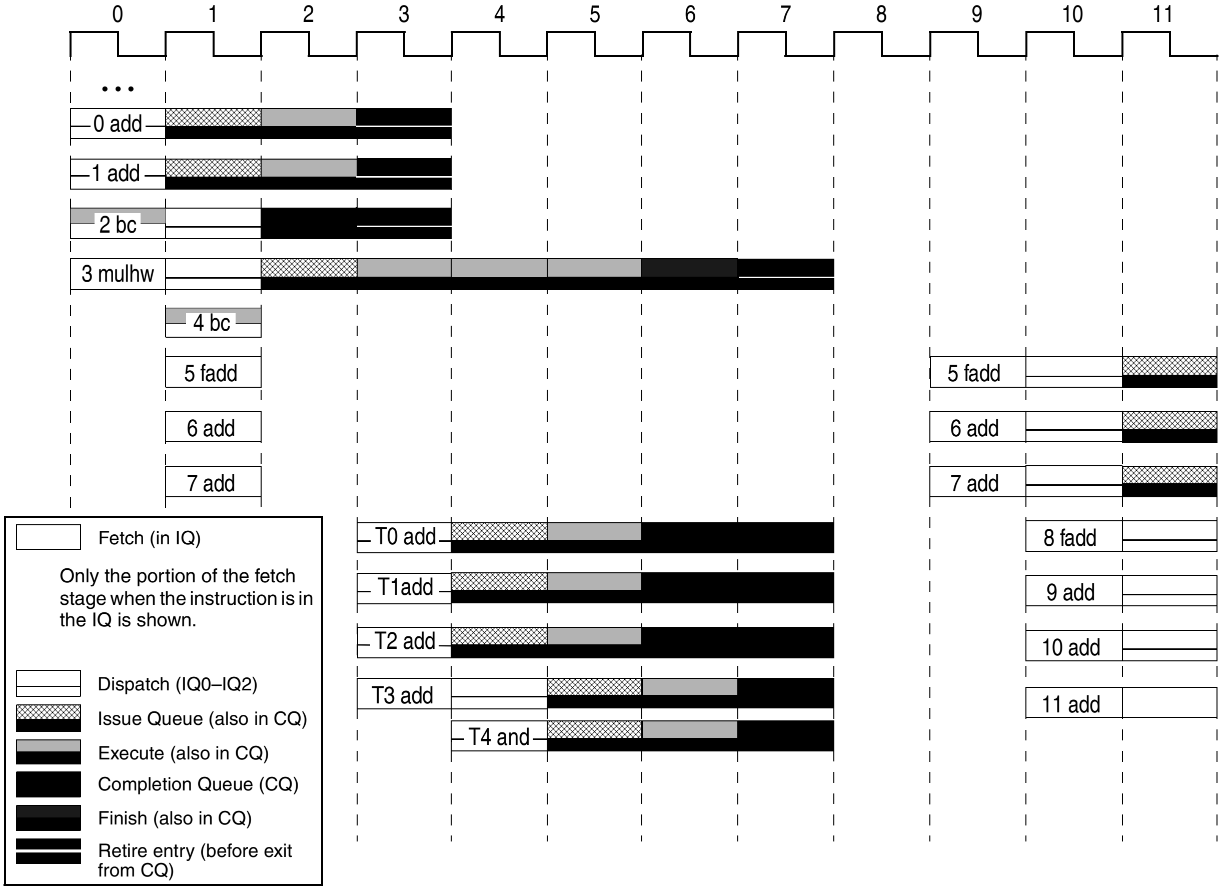
The PowerPC architecture provides a field in branch instructions (the BO field) to allow software to hint whether a branch is likely to be taken. Rather than delaying instruction processing until the condition is known, the MPC7450 begins fetching and executing along the predicted path. When the branch condition is known, the prediction is evaluated. If the prediction is correct, program flow continues along that path; otherwise, the processor flushes any instructions and their results from the mispredicted path, and program flow resumes along the correct path.

Static branch prediction is used when HID0[BHT] is cleared, which disables the branch history table. For information about static branch prediction, see “Conditional Branch Control,” in Chapter 4, “Addressing Modes and Instruction Set Summary,” in the “*Programming Environments Manual*.”

6.4.1.3.2 Predicted Branch Timing Examples

Figure 6-15 shows cases where branch instructions are predicted. It shows how both taken and not-taken branches are handled and how the MPC7450 handles both correct and incorrect predictions. The example shows the timing for the following instruction sequence:

```
0        add
1        add
2        bc
3        mulhw.
4        bc T0
5        fadd
6        add
7        add
T0       add
T1       add
T2       add
T3       add
T4       and
T5       add
```



Instruction Queue

IQ11												
•												
•												
IQ5		7										
IQ4		6										
IQ3	3	5		T3								11
IQ2	2	4		T2							7	10
IQ1	1	3		T1	T4						6	9
IQ0	0	2		T0	T3						5	8

Completion Queue

CQ15												
•												
•												
CQ5						T4	T4	T4				
CQ4						T3	T3	T3				
CQ3			3	3	T2	T2	T2	T2				
CQ2			2	2	T1	T1	T1	T1				7
CQ1		1	1	1	T0	T0	T0	T0				6
CQ0		0	0	0	3	3	3	3				5

* Instructions 5 and 6 are not in the IQ in clock cycle 5. Here, the fetch stage shows cache latency.

Figure 6-15. Branch Instruction Timing

Instruction timing for this example is described cycle-by-cycle as follows:

0. At the end of clock cycle 0, instructions 0 and 1 are dispatched. Instruction 2, a branch instruction that updates the CTR, is predicted as not taken. Instruction 3 is in IQ3.
1. Instructions 0 and 1 are in the GIQ. After the branch instruction is executed, it goes into the CQ from which it updates the CTR when it retires. At the end of this cycle, instruction 3 is dispatched to the GIQ. Because the second **bc** instruction (instruction 4) is predicted as taken, it can be folded. Because it is a BTIC hit, the target instruction stream can be fetched and available for dispatch at the end of clock cycle 3. Although the second **bc** is predicted as taken, it continues through the IQ until the misprediction is detected. Instructions 5–7 are fetched. Because the second branch (instruction 4) is predicted as taken, instructions 5–7 are removed from the IQ at the end of this cycle.
2. Instructions 0 and 1 enter two IU1s. Instruction 2 must remain in the CQ until instructions 0 and 1 complete. Instruction 3 is issued to the IU2.
3. Instructions 0–2 are retired. Instruction 3 continues through the IU2. The BTIC provides instructions T0–T3 to the IQ as branch instruction 4 is executed, predicted taken, and folded. T0–T2 are dispatched at the end of this cycle.
4. Instruction 3 recognizes an early-out condition and moves to the final IU2 stage. Instructions T0–T2 are issued to the three IU1s, and T3 and T4 are dispatched at the end of this clock cycle.
5. Instruction 3 moves to the final IU2 stage. T0–T2 are executed in the IU1s. Instructions T3 and T4 are issued at the end of this clock cycle.
6. Instruction 3 enters the IU2 finish stage. T3 and T4 execute in two IU1s, and T0–T2 wait in the CQ.
7. Instruction 3 is retired and its results indicate that the branch (instruction 4) was mispredicted, so instructions T0–T4 are flushed at the end of this cycle.
8. The mispredicted branch is resolved, so instructions 5–7 are refetched and are now in the fetch1 stage.
9. Instructions 5–7 are in fetch2 and instructions 8–11 are in fetch1.
10. Instructions 5–7 are dispatched at the end of this cycle and instructions 8–11 are in fetch2.
11. Instructions 5–7 are issued and instructions 8–11 are dispatched.

6.4.2 Integer Unit Execution Timing

The MPC7450 has three single-cycle integer (IU1s) and one multiple-cycle integer unit (IU2). The three IU1s execute all integer instructions except multiplies and divides. The IU2 executes multiplies, divides, and several miscellaneous instructions, including CR logic and move to/from SPR instructions. Table 6-5 lists integer unit instruction latencies. As Figure 6-5 shows, most integer instructions have a single-cycle execution latency.

6.4.3 FPU Execution Timing

The FPU executes single- and double-precision floating-point operations compliant with the IEEE-754 floating-point standard. Single- and double-precision floating-point multiply, add, and subtract execute in a five-stage pipeline. Most floating-point instructions execute with 5-cycle latency and 1-cycle throughput; however, **fdivs**, **fres**, and **fdiv** have latencies of 14 to 35 cycles. The **fdivs**, **fdiv**, **fres**, **mcrfs**, **mtfsb0**, **mtfsb1**, **mtfsfi**, **mffs**, and **mtfsf** instructions block the FPU pipeline until they complete execution, inhibiting the issue of additional floating-point instructions. Table 6-6 shows floating-point instruction execution timing.

6.4.3.1 Effect of Floating-Point Exceptions on Performance

For the best, most predictable MPC7450 floating-point performance, IEEE floating-point exceptions should be disabled in the FPSCR and MSR.

If an exception is enabled (through a combination of MSR[FE0, FE1] and one or more FPSCR enable bits), the instruction traps to the program interrupt handler. Floating-point operations that create an exception sticky bit in the FPSCR may degrade performance.

6.4.4 Load/Store Unit Execution Timing

The LSU executes load and store instructions, including AltiVec LRU and transient instructions. The execution of most load instructions is pipelined in the three LSU stages, during which the effective address is calculated, MMU translation is performed, the data cache array and tags are read, and cache way selection and data alignment are carried out. If there are no data dependencies, cacheable GPR and vector register load instructions have a 3-cycle latency and a 1-cycle throughput. Cacheable FPR load instructions have a 4-cycle latency and a 1-cycle throughput. The data cache array is updated after a store instruction is retired.

If operands are misaligned, additional latency may be required either for an alignment exception or for additional bus accesses. Load instructions that miss in the cache block subsequent cache accesses during the cache line refill. Table 6-7 gives load and store instruction execution latencies.

6.4.4.1 Effect of Operand Placement on Performance

The location and alignment of operands in memory may affect performance of memory accesses, in some cases significantly, as shown in Table 6-1.

Alignment of memory operands on natural boundaries guarantees the best performance. For the best performance across the widest range of implementations, the programmer should assume the performance model described in Chapter 3, “Operand Conventions,” in the “*Programming Environments Manual*.”

The effect of alignment on memory operation performance is the same for big- and little-endian addressing modes except for multiple and string operations, which cause an alignment interrupt in little-endian mode.

In Table 6-1, optimal means that one effective address (EA) calculation occurs during the memory operation. Fair means that multiple EA calculations occur during the operation, which may cause additional bus activities with multiple bus transfers. Poor means that an alignment interrupt is generated by the memory operation.

Table 6-1. Performance Effects of Memory Operand Placement

Operand		Boundary Crossing [1]			
Size	**Byte Alignment**	**None**	**8 Byte**	**Cache Line**	**Protection Boundary**
Integer					
4 byte	4 <4	Optimal Optimal	— Fair	— Fair	— Fair
2 byte	2 <2	Optimal Optimal	— Fair	— Fair	— Fair
1 byte	1	Optimal	—	—	—
lmw, **stmw** [2]	4 <4	Fair Poor	Fair Poor	Fair Poor	Fair Poor
String [2, 4]		Fair	Fair	Fair	Fair
Floating-Point					
8 byte	8 4 <4	Optimal — —	— Fair Poor	— Fair Poor	— Fair Poor
4 byte	4 <4	Optimal Poor	— Poor	— Poor	— Poor
AltiVec [3]					
16 byte	16	Optimal	—	—	—

[1] Vector operands are not shown because they are always aligned.
Optimal: One EA calculation occurs.
Fair: Multiple EA calculations occur which may cause additional bus activities with multiple bus transfers.
Poor: Alignment exception occurs.

[2] These operations are not supported in little-endian mode and would cause an alignment exception.

[3] AltiVec memory operations are forced to align on a 16-byte boundary.

[4] Usage of string instructions is strongly discouraged.

Note that the MPC7450 differs from the MPC750 in some aspects of little-endian operation; in little-endian mode, the MPC7450 does not work with the MPC106.

6.4.4.2 Store Gathering

The MPC7450 performs store gathering for cache-inhibited and write-through operations to nonguarded space as well as for cacheable write-back stores. However, stores are gathered only if the successive stores meet the criteria and are queued and pending. Store gathering occurs regardless of the address order of the stores and is enabled by setting HID0[SGE]. Bytes can be gathered into half words, which can be gathered into words, which can be gathered into double words, which can be gathered into quad words (MPX bus mode only), and quad words can be gathered into cache lines. In addition, cacheable write-back stores to the same cache line can be merged regardless of size or alignment.

Store gathering is not done for the following:

- Stores to guarded cache-inhibited or write-through space
- **stwcx.** instructions
- **ecowx** instructions
- Double word-to-quad word gathering if in 60x mode
- Cache-inhibited or write-through stores if the result would violate MPX bus alignment

If store gathering is enabled and the stores do not fall under the above categories, an **eieio** or **sync** instruction must be used to prevent two stores from being gathered.

6.4.4.3 AltiVec Instructions Executed by the LSU

The LSU executes the AltiVec LRU and transient instructions.

6.4.4.3.1 LRU Instructions

The AltiVec architecture specifies that the **lvxl** and **stvxl** instructions differ from other AltiVec load and store instructions in that they leave cache entries in a least-recently-used state instead of a most-recently-used state. This is used to identify data that is known to have little reuse and poor caching characteristics.

On the MPC7450, these instructions follow the cache allocation and replacement policies described in Chapter 3, "L1, L2, and L3 Cache Operation," but they leave their addressed cache entries in the LRU state. In addition, all LRU instructions are also interpreted to be transient and are also treated as described in the next section. Additional discussion of LRU effects can be found in Chapter 3, "L1, L2, and L3 Cache Operation."

6.4.4.3.2 Transient Instructions

The AltiVec architecture describes a difference between static and transient memory accesses. A static memory access should have some reasonable degree of locality and should be referenced several times or reused over some reasonably long period of time. A transient memory reference has poor locality and is likely to be referenced a very few times or over a very short period of time.

The MPC7450 supports both static and transient memory access behavior. If a memory access is designated as transient, that cache block is not allocated into the L2 or L3. As L1 castouts are not consumed by the L2 or L3 caches unless the line is already resident in the L2 or L3 cache, this forces the block to be written directly to main memory, bypassing the L2 and L3 caches. Note that L3 cache is not supported on the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448.

The following instructions are interpreted to be transient:

- **dstt** and **dststt** (transient forms of the two data stream touch instructions)
- **lvxl** and **stvxl**

Use of the **dstst** and **dststt** instructions is not recommended. Almost always, a **dst** or **dstt**, or sometimes a series of **dcbz** instructions, is more appropriate.

6.4.5 AltiVec Instructions

The MPC7450 implements all instructions in the AltiVec specification. The AltiVec instruction set has no optional instructions; however, a few instructions associated with the load/store model are defined to allow significant differences between implementations. The following sections describe the MPC7450 implementation of these options.

6.4.5.1 AltiVec Unit Execution Timing

The AltiVec unit contains the following four independent execution units:

- Vector permute unit (VPU)—All AltiVec permute instructions are executed in 2 cycles.
- Vector simple integer unit (VIU1)
- Vector complex integer unit (VIU2)
- Vector floating-point unit (VFPU)

Execution timing for these units is described in the following sections.

6.4.5.1.1 AltiVec Permute Unit (VPU) Execution Timing

The VPU executes all AltiVec permute instructions, which have a 2-cycle latency.

6.4.5.1.2 Vector Simple Integer Unit (VIU1) Execution Timing

The VIU1 executes all AltiVec simple integer instructions, all of which have a single-cycle latency plus a separate finish stage.

The **mtvscr**, **mfvscr**, **vcmpbpfp**, **vcmpeqfp**, **vcmpgtfp**, **vmaxfp**, and **vminfp** instructions, which were in the VIU1 (VALU(VSIU)) in the MPC7400 and MPC7410, have been moved to the VFPU in MPC7450. The **vsl** are **vsr** instructions, which were also in the VIU1 (VALU(VSIU)) in the MPC7400, have been moved to the VPU in the MPC7450.

6.4.5.1.3 Vector Complex Integer Unit (VIU2) Execution Timing

The VIU2 executes all AltiVec complex integer instructions, which have a 4-cycle latency.

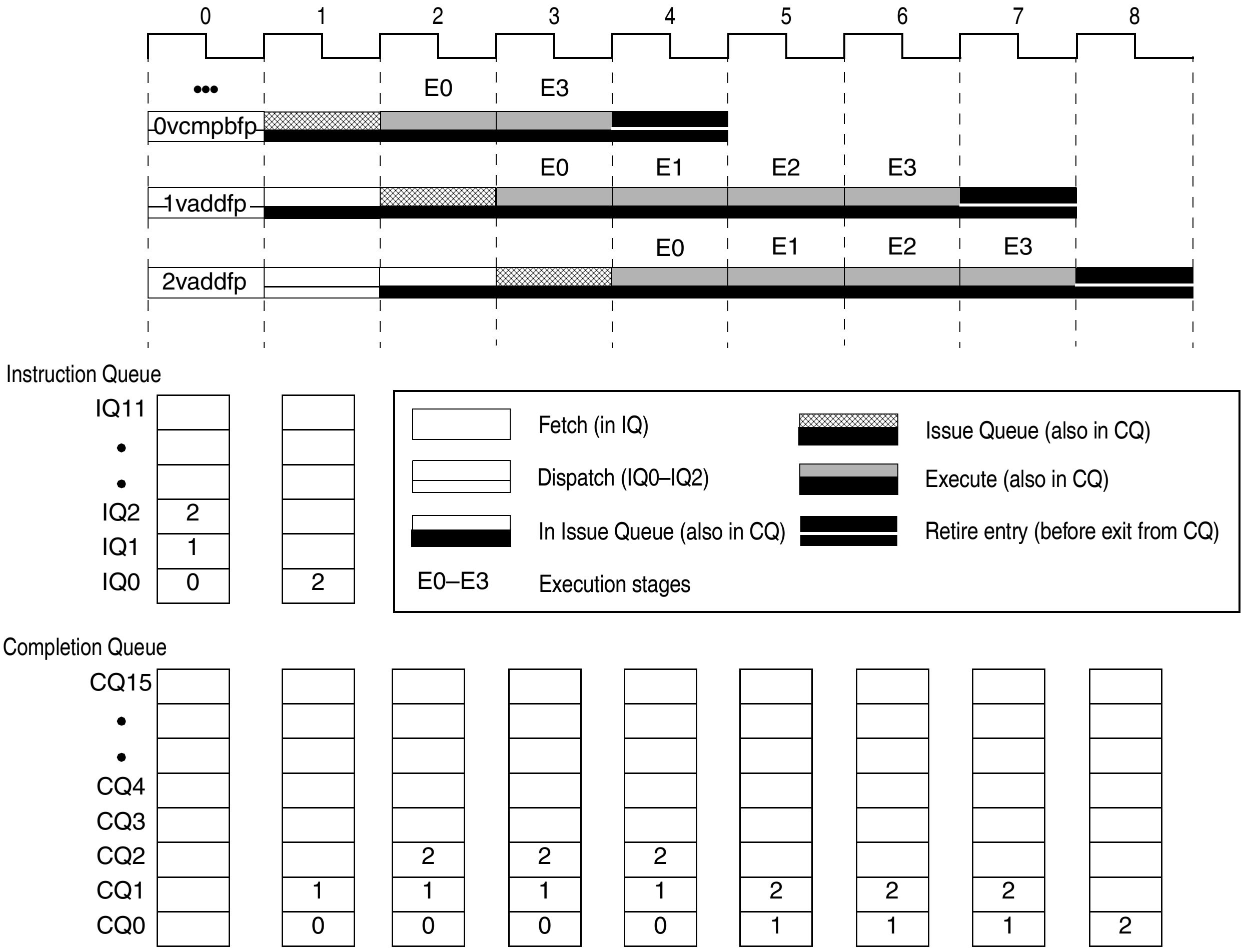
6.4.5.1.4 Vector Floating-Point Unit (VFPU) Execution Timing

Most AltiVec floating-point instructions (regardless of Java/non-Java mode) have a 4-cycle latency on the MPC7450, unlike the MPC7400 or MPC7410 which has a 4- or 5-cycle latency depending on non-Java or Java mode, respectively. However, the **vcmpbfp**, **vcmpeqfp**, **vcmpgtfp**, **vmaxfp**, and **vminfp** instructions have 2-cycle latency. Under the conditions shown in Figure 6-17, these instructions may increase the latency of other VFPU instructions.

The following two examples show a VFPU pipelining special case for vector-float-compare instructions.

In the following code sequence, **vcmpbfp** takes only two cycles. Note that the **vcmpbfp** jumps from execution stage 0 in clock cycle 2 to execution stage 3 in clock cycle 3. However, the bypass does not block other instructions.

```
0 vcmpbfp
1 vaddfp
2 vaddfp
```



Instruction Queue

IQ11		
•		
•		
IQ2	2	
IQ1	1	
IQ0	0	2

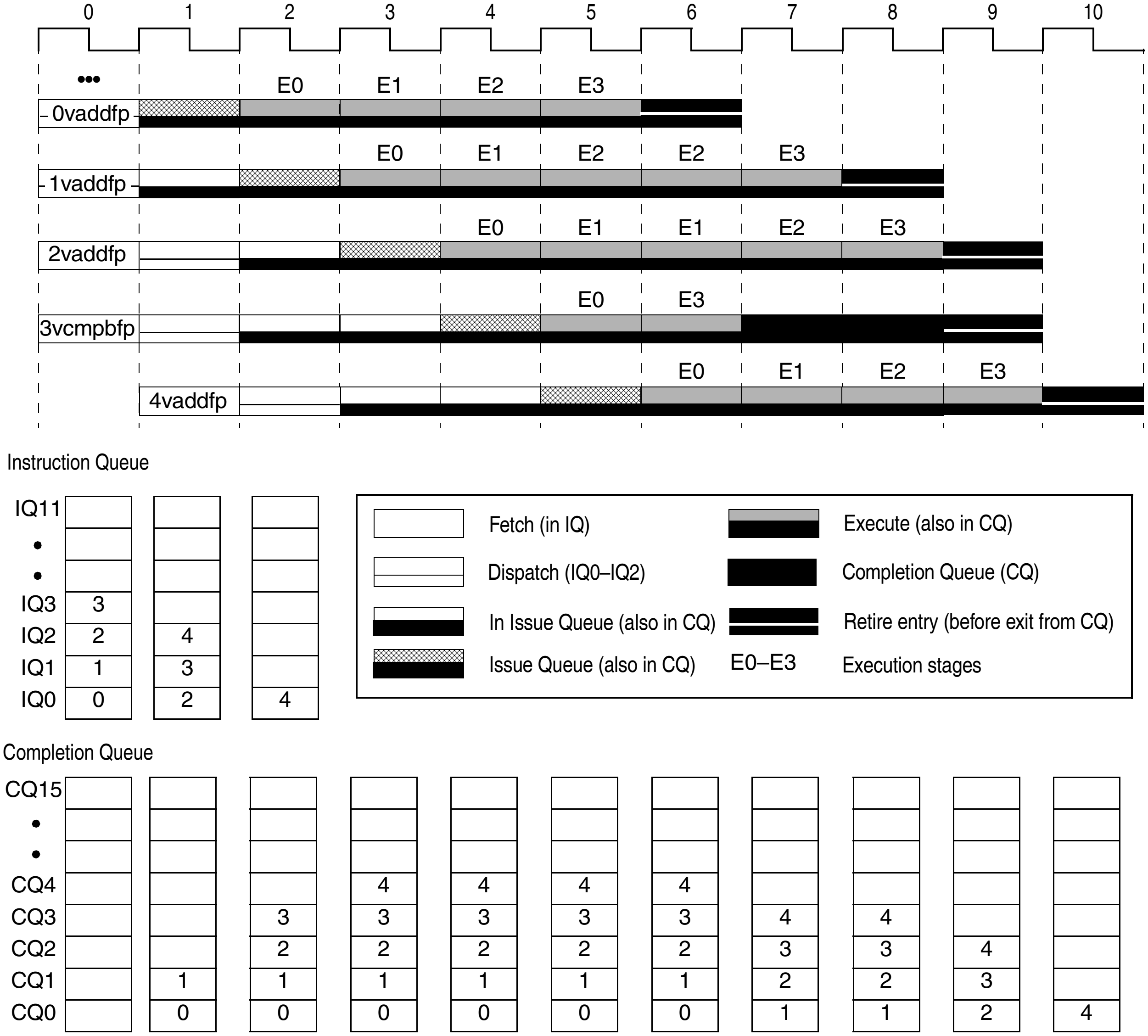
Completion Queue

CQ15									
•									
•									
CQ4									
CQ3									
CQ2			2	2	2				
CQ1		1	1	1	1	2	2	2	
CQ0		0	0	0	0	1	1	1	2

Figure 6-16. Vector Floating-Point Compare Bypass Non-Blocking

In the next sequence, **vcmpbfp** has a 2-cycle latency and bypasses other instructions. Note that in clock cycle 5, the bypass blocks all other instructions in the sequence. Because of the bypass, the second and third **vaddfp** do not advance in that cycle. Effectively, the bypass causes the second and third **vaddfp** to have a 5th cycle of latency.

```
0 vaddfp
1 vaddfp
2 vaddfp
3 vcmpbfp
4 vaddfp
```



Instruction Queue

IQ11			
•			
•			
IQ3	3		
IQ2	2	4	
IQ1	1	3	
IQ0	0	2	4

Completion Queue

CQ15											
•											
•											
CQ4				4	4	4	4				
CQ3			3	3	3	3	3	4	4		
CQ2			2	2	2	2	2	3	3	4	
CQ1		1	1	1	1	1	1	2	2	3	
CQ0		0	0	0	0	0	0	1	1	2	4

Figure 6-17. Vector Float Compare Bypass Blocking

6.5 Memory Performance Considerations

Because the MPC7450 has a maximum instruction throughput of three instructions per clock cycle, lack of memory bandwidth can affect performance. To maximize performance, the MPC7450 must be able to read and write data efficiently. If a system has multiple bus devices, one device may experience long memory latencies while another device (for example, a direct-memory access controller) is using the external bus.

6.5.1 Caching and Memory Coherency

To minimize the effect of bus contention, the PowerPC architecture defines WIM bits that define caching characteristics for the corresponding page or block. Accesses to caching-inhibited memory locations never update the L1, L2, or L3 caches. Note that L3 cache is not supported on the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448. If a cache-inhibited access hits in any of the caches, the cache block is invalidated. If the cache block is marked modified, it is copied back to memory before being invalidated. Where caching is permitted, memory is configured as either write-back or write-through, as described in the Chapter 3, “L1, L2, and L3 Cache Operation.”

6.6 Instruction Latency Summary

Instruction timing is shown in Table 6-2 through Table 6-8. The latency tables use the following conventions:

- Pipelined load/store and floating-point instructions are shown with cycles of total latency and throughput cycles separated by a colon.
- Floating-point instructions with a single entry in the cycles column are not pipelined.
- In addition, additional cycles due to serialization are indicated in the cycles column with the following:
  - e (execution serialization)
  - r (refetch serialization)
  - s (store serialization)

Figure 6-2 through Figure 6-8 list latencies associated with instructions executed by each execution unit. Figure 6-2 describes branch instruction latencies.

Table 6-2. Branch Operation Execution Latencies

Mnemonic	Primary	Extend	Form	Unit	Cycles
b[l][a]	18	—	i	BPU	1[1]
bc[l][a]	16	—	b	BPU	1[1]
bcctr[l]	19	528	xl	BPU	1[1]
bclr[l]	19	016	xl	BPU	1, 2[1]

[1] Branches that do not modify the LR or CTR can be folded and not dispatched. Branches that are dispatched go only to the CQ.

Table 6-3 lists system operation instruction latencies.

Table 6-3. System Operation Instruction Execution Latencies

Mnemonic	Primary	Extend	Form	Unit	Cycles
eieio	31	854	X	LSU	3:5 {s}
isync	19	150	XL	—[1]	0{r}
mfmsr	31	083	X	IU2	3-2
mfspr (DBATs)	31	339	XFX	IU2	4:3{e}
mfspr (IBATs)	31	339	XFX	IU2	4:3
mfspr (MSS)	31	339	XFX	IU2	5{e}[2]
mfspr (other)	31	339	XFX	IU2	3{e}
mfspr (Time Base)	31	339	XFX	IU2	5{e}
mfspr (VRSAVE)	31	339	XFX	IU2	3:2
mfsr	31	595	X	IU2	4:3
mfsrin	31	659	X	IU2	4:3
mftb	31	371	X	IU2	5{e}
mtmsr	31	146	X	IU2	2{e}
mtspr (DBATs)	31	467	XFX	IU2	2{e}
mtspr (IBATs)	31	467	XFX	IU2	2{e}
mtspr (MSS)	31	467	XFX	IU2	5{e}
mtspr (other)	31	467	XFX	IU2	2{e}
mtspr (XER)	31	467	XFX	IU2	2{e,r}[1]
mtsr	31	210	X	IU2	2{e}
mtsrin	31	242	X	IU2	2{e}
mttb	31	467	XFX	IU2	5{e}
rfi	19	050	XL	—[1]	0{r}
sc	17	—[1]	SC	—[1]	0{r}
sync	31	598	X	LSU	35[3]{e,s}
tlbsync	31	566	X	LSU	3:5{s}

[1] Refetch serialized instructions (if marked with a 0-cycle execution time) do not have an execute stage, and all refetch serialized instructions have 1-cycle between the time they are completed and the time the target/sequential instruction enters the fetch1 stage.

[2] Memory subsystem SPRs are implementation specific and are described in Chapter 2, "Programming Model."

[3] Assuming a 5:1 processor to clock ratio.

Table 6-4 lists condition register logical instruction latencies.

Table 6-4. Condition Register Logical Execution Latencies

Mnemonic	Primary	Extend	Form	Unit	Cycles
crand	19	257	XL	IU2	2{e}
crandc	19	129	XL	IU2	2{e}
creqv	19	289	XL	IU2	2{e}
crnand	19	225	XL	IU2	2{e}
crnor	19	033	XL	IU2	2{e}
cror	19	449	XL	IU2	2{e}
crorc	19	417	XL	IU2	2{e}
crxor	19	193	XL	IU2	2{e}
mcrf	19	000	XL	IU2	2{e}
mcrxr	31	512	X	IU2	2{e}
mfcr	31	019	X	IU2	2{e}
mtcrf	31	144	XFX	IU2/IU1	2{e}/1[1]

[1] **mtcrf** of a single field is executed by an IU1 in a single cycle and is not serialized.

Table 6-5 lists integer unit instruction latencies.

Table 6-5. Integer Unit Execution Latencies

Mnemonic	Primary	Extend	Form	Unit	Cycles
addc[o][.]	31	010	XO	IU1	1
adde[o][.]	31	138	XO	IU1	1{e}
addi	14	—	D	IU1	1
addic	12	—	D	IU1	1
addic.	13	—	D	IU1	1
addis	15	—	D	IU1	1
addme[o][.]	31	234	XO	IU1	1{e}
addze[o][.]	31	202	XO	IU1	1{e}
add[o][.]	31	266	XO	IU1	1
andc[.]	31	060	X	IU1	1
andi.	28	—	D	IU1	1
andis.	29	—	D	IU1	1
and[.]	31	028	X	IU1	1
cmp	31	000	X	IU1	1

Table 6-5. Integer Unit Execution Latencies (continued)

Mnemonic	Primary	Extend	Form	Unit	Cycles
cmpi	11	—	D	IU1	1
cmpl	31	032	X	IU1	1
cmpli	10	—	D	IU1	1
cntlzw[.]	31	026	X	IU1	1
divwu[o][.][1]	31	459	XO	IU2	23
divw[o][.][2]	31	491	XO	IU2	23
eqv[.]	31	284	X	IU1	1
extsb[.]	31	954	X	IU1	1 [3]
extsh[.]	31	922	X	IU1	1 [3]
mulhwu[.]	31	011	XO	IU2	4:2 [4]
mulhw[.]	31	075	XO	IU2	4:2 [4]
mulli	07	—	D	IU2	3:1
mull[o][.]	31	235	XO	IU2	4:2 [4]
nand[.]	31	476	X	IU1	1
neg[o][.]	31	104	XO	IU1	1
nor[.]	31	124	X	IU1	1
orc[.]	31	412	X	IU1	1
ori	24	—	D	IU1	1
oris	25	—	D	IU1	1
or[.]	31	444	X	IU1	1
rlwimi[.]	20	—	M	IU1	1 [3]
rlwinm[.]	21	—	M	IU1	1 [3]
rlwnm[.]	23	—	M	IU1	1 [3]
slw[.]	31	024	X	IU1	1 [3]
srawi[.]	31	824	X	IU1	2 [5]
sraw[.]	31	792	X	IU1	2 [5]
srw[.]	31	536	X	IU1	1 [1]
subfc[o][.]	31	008	XO	IU1	1
subfe[o][.]	31	136	XO	IU1	1{e}
subfic	08	—	D	IU1	1
subfme[o][.]	31	232	XO	IU1	1{e}
subfze[o][.]	31	200	XO	IU1	1{e}
subf[.]	31	040	XO	IU1	1

Table 6-5. Integer Unit Execution Latencies (continued)

Mnemonic	Primary	Extend	Form	Unit	Cycles
tw	31	004	X	IU1	2
twi	03	—	D	IU1	2
xori	26	—	D	IU1	1
xoris	27	—	D	IU1	1
xor[.]	31	316	X	IU1	1

[1] For the special case of division by zero, the latency is 3 cycles.

[2] For the special case of division by zero or 0x8000_0000 divided by 0xFFFF_FFFF, the latency is 3 cycles

[3] If the record bit is set, the result is available in 1 cycle and execution takes 2 cycles.

[4] 32*32-bit multiplication has an early exit condition. If the 15 msbs of the B operand are either all set or all cleared, the multiply finishes with a latency of 3 and a throughput of 1.

[5] **srawi**[.] and **sraw**[.] produce a GPR result in 1 cycle, but the full results, including the CA, OV, CR results, are available in 2 cycles.

Table 6-6 shows latencies for FPU instructions. Instructions with a single entry in the cycles column are not pipelined; all FPU stages are busy for the full duration of instruction execution and are unavailable to subsequent instructions. Floating-point arithmetic instructions execute in the FPU; floating-point loads and stores execute in the LSU

For pipelined instructions, two numbers are shown separated by a colon. The first shows the number of cycles required to fill the pipeline. The second is the throughput once the pipeline is full. For example, **fabs**[.] passes through five stages with a 1-cycle throughput.

Table 6-6. Floating-Point Unit (FPU) Execution Latencies

Mnemonic	Primary	Extend	Form	Cycles
fabs[.]	63	264	X	5:1
fadds[.]	59	021	A	5:1
fadd[.]	63	021	A	5:1
fcmpo	63	032	X	5:1
fcmpu	63	000	X	5:1
fctiwz[.]	63	015	X	5:1
fctiw[.]	63	014	X	5:1
fdivs[.]	59	018	A	21
fdiv[.]	63	018	A	35
fmadds[.]	59	029	A	5:1
fmadd[.]	63	029	A	5:1
fmr[.]	63	072	X	5:1
fmsubs[.]	59	028	A	5:1

Table 6-6. Floating-Point Unit (FPU) Execution Latencies (continued)

Mnemonic	Primary	Extend	Form	Cycles
fmsub[.]	63	028	A	5:1
fmuls[.]	59	025	A	5:1
fmul[.]	63	025	A	5:1
fnabs[.]	63	136	X	5:1
fneg[.]	63	040	X	5:1
fnmadds[.]	59	031	A	5:1
fnmadd[.]	63	031	A	5:1
fnmsubs[.]	59	030	A	5:1
fnmsub[.]	63	030	A	5:1
fres[.]	59	024	A	14
frsp[.]	63	012	X	5:1
frsqrte[.]	63	026	A	5:1
fsel[.]	63	023	A	5:1
fsubs[.]	59	020	A	5:1
fsub[.]	63	020	A	5:1
mcrfs	63	064	X	5{e}
mffs[.]	63	583	X	5{e}
mtfsb0[.]	63	070	X	5{e}
mtfsb1[.]	63	038	X	5{e}
mtfsfi[.]	63	134	X	5{e}
mtfsf[.]	63	711	XFL	5{e}

Table 6-7 shows load and store instruction latencies. Load/store multiple and string instruction cycles are represented as a fixed number of cycles plus a variable number of cycles, where *n* = the number of words accessed by the instruction. Pipelined load/store instructions are shown with total latency and throughput separated by a colon.

Table 6-7. Load/Store Unit (LSU) Instruction Latencies

Mnemonic	Primary	Extend	Form	Cycles[1]
dcba	31	758	X	3:1{s}
dcbf	31	86	X	3:11{s}
dcbst	31	54	X	3:11{s}
dcbt	31	278	X	3:1
dcbtst	31	246	X	3:1

Table 6-7. Load/Store Unit (LSU) Instruction Latencies (continued)

Mnemonic	Primary	Extend	Form	Cycles[1]
dcbz	31	1014	X	3:1{s}
dss	31	582	X	3:1
dssall	31	582	X	3:1
dsts[t]	31	550	X	3:1
dst[t]	31	518	X	3:1
eciwx	31	310	X	3:1
icbi	31	982	X	3:1{s}
lbz	34	—	D	3:1
lbzu	35	—	D	3:1
lbzux	31	119	X	3:1
lbzx	31	87	X	3:1
lfd	50	—	D	4:1
lfdu	51	—	D	4:1
lfdux	31	631	X	4:1
lfdx	31	599	X	4:1
lfs	48	—	D	4:1
lfsu	49	—	D	4:1
lfsux	31	567	X	4:1
lfsx	31	535	X	4:1
lha	42	—	D	3:1
lhau	43	—	D	3:1
lhaux	31	375	X	3:1
lhax	31	343	X	3:1
lhbrx	31	790	X	3:1
lhz	40	—	D	3:1
lhzux	31	311	X	3:1
lhzx	31	279	X	3:1
lmw	46	—	D	3 + n
lswi	31	597	X	3 + n
lswx	31	533	X	3 + n
lvebx	31	7	X	3:1
lvehx	31	39	X	3:1
lvewx	31	71	X	3:1

Table 6-7. Load/Store Unit (LSU) Instruction Latencies (continued)

Mnemonic	Primary	Extend	Form	Cycles[1]
lvsl	31	6	X	3:1
lvsr	31	38	X	3:1
lvx	31	103	X	3:1
lvxl	31	359	X	3:1
lwarx	31	20	X	3{e}
lwbrx	31	534	X	3:1
lwz	32	—	D	3:1
lwzu	33	—	D	3:1
lwzux	31	55	X	3:1
lwzx	31	23	X	3:1
stb	38	—	D	3:1{s}
stbu	39	—	D	3:1{s}
stbux	31	247	X	3:1{s}
stbx	31	215	X	3:1{s}
stfd	54	—	D	3:3{s} [2]
stfdu	55	—	D	3:3{s} [2]
stfdux	31	759	X	3:3{s} [2]
stfdx	31	727	X	3:3{s} [2]
stfiwx	31	983	X	3:1{s}
stfs	52	—	D	3:3{s} [2]
stfsu	53	—	D	3:3{s} [2]
stfsux	31	695	X	3:3{s} [2]
stfsx	31	663	X	3:3{s} [2]
sth	44	—	D	3:1{s}
sthbrx	31	918	X	3:1{s}
sthu	45	—	D	3:1{s}
stmw	47	—	D	3 + n{s}
stswi	31	725	X	3+ n{s}
stswx	31	661	X	3 + n{s}
stvebx	31	135	X	3:1{s}
stvehx	31	167	X	3:1{s}
stvewx	31	199	X	3:1{s}
stvx	31	231	X	3:1{s}

Table 6-7. Load/Store Unit (LSU) Instruction Latencies (continued)

Mnemonic	Primary	Extend	Form	Cycles[1]
stvxl	31	487	X	3:1{s}
stw	36	—	D	3:1{s}
stwbrx	31	662	X	3:1{s}
stwcx.	31	150	X	3:1{s}
stwu	37	—	D	3:1{s}
stwux	31	183	X	3:1{s}
stwx	31	151	X	3:1{s}
tlbie	31	306	X	3:1{s}
tlbld	31	978	X	3{e}
tlbli	31	1010	X	3{e}

[1] For cache operations, the first number indicates the latency for finishing a single instruction and the second number indicates the throughput for a large number of back-to-back cache operations. The throughput cycle may be larger than the initial latency because more cycles may be needed for the data to reach the cache. If the cache remains busy, subsequent cache operations cannot execute.

[2] Floating-point stores may take as many as 24 additional cycles if the value being stored is a denormalized number.

Table 6-8 describes AltiVec instruction latencies.

Table 6-8. AltiVec Instruction Latencies

Mnemonic	Primary	Extend	Form	Unit	Cycles[1]
mfvscr	04	1540	VX	VFPU	2{e}
mtvscr	04	1604	VX	VFPU	2{e}
vaddcuw	04	384	VX	VIU1	1
vaddfp	04	10	VX	VFPU	4:1
vaddsbs	04	768	VX	VIU1	1
vaddshs	04	832	VX	VIU1	1
vaddsws	04	896	VX	VIU1	1
vaddubm	04	0	VX	VIU1	1
vaddubs	04	512	VX	VIU1	1
vadduhm	04	64	VX	VIU1	1
vadduhs	04	576	VX	VIU1	1
vadduwm	04	128	VX	VIU1	1
vadduws	04	640	VX	VIU1	1
vand	04	1028	VX	VIU1	1

Table 6-8. AltiVec Instruction Latencies (continued)

Mnemonic	Primary	Extend	Form	Unit	Cycles[1]
vandc	04	1092	VX	VIU1	1
vavgsb	04	1282	VX	VIU1	1
vavgsh	04	1346	VX	VIU1	1
vavgsw	04	1410	VX	VIU1	1
vavgub	04	1026	VX	VIU1	1
vavguh	04	1090	VX	VIU1	1
vavguw	04	1154	VX	VIU1	1
vcfsx	04	842	VX	VFPU	4:1
vcfux	04	778	VX	VFPU	4:1
vcmpbfp[.]	04	966 [1990]	VX	VFPU	2:1
vcmpeqfp[.]	04	198 [1222]	VX	VFPU	2:1
vcmpequb[.]	04	6 [1030]	VX	VIU1	1
vcmpequh[.]	04	70 [1094]	VX	VIU1	1
vcmpequw[.]	04	134 [1158]	VX	VIU1	1
vcmpgefp[.]	04	454 [1478]	VX	VFPU	2:1
vcmpgtfp[.]	04	710 [1734]	VX	VFPU	2:1
vcmpgtsb[.]	04	774 [1798]	VX	VIU1	1
vcmpgtsh[.]	04	838 [1862]	VX	VIU1	1
vcmpgtsw[.]	04	902 [1926]	VX	VIU1	1
vcmpgtub[.]	04	518 [1542]	VX	VIU1	1
vcmpgtuh[.]	04	582 [1606]	VX	VIU1	1
vcmpgtuw[.]	04	646 [1670]	VX	VIU1	1
vctsxs	04	970	VX	VFPU	4:1
vctuxs	04	906	VX	VFPU	4:1
vexptefp	04	394	VX	VFPU	4:1
vlogefp	04	458	VX	VFPU	4:1
vmaddfp	04	46	VA	VFPU	4:1
vmaxfp	04	1034	VX	VFPU	2:1
vmaxsb	04	258	VX	VIU1	1
vmaxsh	04	322	VX	VIU1	1
vmaxsw	04	386	VX	VIU1	1
vmaxub	04	2	VX	VIU1	1
vmaxuh	04	66	VX	VIU1	1

Table 6-8. AltiVec Instruction Latencies (continued)

Mnemonic	Primary	Extend	Form	Unit	Cycles[1]
vmaxuw	04	130	VX	VIU1	1
vmhaddshs	04	32	VA	VIU2	4:1
vmhraddshs	04	33	VA	VIU2	4:1
vminfp	04	1098	VX	VFPU	2:1
vminsb	04	770	VX	VIU1	1
vminsh	04	834	VX	VIU1	1
vminsw	04	898	VX	VIU1	1
vminub	04	514	VX	VIU1	1
vminuh	04	578	VX	VIU1	1
vminuw	04	642	VX	VIU1	1
vmladduhm	04	34	VA	VIU2	4:1
vmrghb	04	12	VX	VPU	2:1
vmrghh	04	76	VX	VPU	2:1
vmrghw	04	140	VX	VPU	2:1
vmrglb	04	268	VX	VPU	2:1
vmrglh	04	332	VX	VPU	2:1
vmrglw	04	396	VX	VPU	2:1
vmsummbm	04	37	VA	VIU2	4:1
vmsumshm	04	40	VA	VIU2	4:1
vmsumshs	04	41	VA	VIU2	4:1
vmsumubm	04	36	VA	VIU2	4:1
vmsumuhm	04	38	VA	VIU2	4:1
vmsumuhs	04	39	VA	VIU2	4:1
vmulesb	04	776	VX	VIU2	4:1
vmulesh	04	840	VX	VIU2	4:1
vmuleub	04	520	VX	VIU2	4:1
vmuleuh	04	584	VX	VIU2	4:1
vmulosb	04	264	VX	VIU2	4:1
vmulosh	04	328	VX	VIU2	4:1
vmuloub	04	8	VX	VIU2	4:1
vmulouh	04	72	VX	VIU2	4:1
vnmsubfp	04	47	VA	VFPU	4:1
vnor	04	1284	VX	VIU1	1

Table 6-8. AltiVec Instruction Latencies (continued)

Mnemonic	Primary	Extend	Form	Unit	Cycles[1]
vor	04	1156	VX	VIU1	1
vperm	04	43	VA	VPU	2:1
vpkpx	04	782	VX	VPU	2:1
vpkshss	04	398	VX	VPU	2:1
vpkshus	04	270	VX	VPU	2:1
vpkswss	04	462	VX	VPU	2:1
vpkswus	04	334	VX	VPU	2:1
vpkuhum	04	14	VX	VPU	2:1
vpkuhus	04	142	VX	VPU	2:1
vpkuwum	04	78	VX	VPU	2:1
vpkuwus	04	206	VX	VPU	2:1
vrefp	04	266	VX	VFPU	4:1
vrfim	04	714	VX	VFPU	4:1
vrfin	04	522	VX	VFPU	4:1
vrfip	04	650	VX	VFPU	4:1
vrfiz	04	586	VX	VFPU	4:1
vrlb	04	4	VX	VIU1	1
vrlh	04	68	VX	VIU1	1
vrlw	04	132	VX	VIU1	1
vrsqrtefp	04	330	VX	VFPU	4:1
vsel	04	42	VA	VIU1	1
vsl	04	452	VX	VPU	2:1
vslb	04	260	VX	VIU1	1
vsldoi	04	44	VA	VPU	2:1
vslh	04	324	VX	VIU1	1
vslo	04	1036	VX	VPU	2:1
vslw	04	388	VX	VIU1	1
vspltb	04	524	VX	VPU	2:1
vsplth	04	588	VX	VPU	2:1
vspltisb	04	780	VX	VPU	2:1
vspltish	04	844	VX	VPU	2:1
vspltisw	04	908	VX	VPU	2:1
vspltw	04	652	VX	VPU	2:1

Table 6-8. AltiVec Instruction Latencies (continued)

Mnemonic	Primary	Extend	Form	Unit	Cycles[1]
vsr	04	708	VX	VPU	2:1
vsrab	04	772	VX	VIU1	1
vsrah	04	836	VX	VIU1	1
vsraw	04	900	VX	VIU1	1
vsrb	04	516	VX	VIU1	1
vsrh	04	580	VX	VIU1	1
vsro	04	1100	VX	VPU	2:1
vsrw	04	644	VX	VIU1	1
vsubcuw	04	1408	VX	VIU1	1
vsubfp	04	74	VX	VFPU	4:1
vsubsbs	04	1792	VX	VIU1	1
vsubshs	04	1856	VX	VIU1	1
vsubsws	04	1920	VX	VIU1	1
vsububm	04	1024	VX	VIU1	1
vsububs	04	1536	VX	VIU1	1
vsubuhm	04	1088	VX	VIU1	1
vsubuhs	04	1600	VX	VIU1	1
vsubuwm	04	1152	VX	VIU1	1
vsubuws	04	1664	VX	VIU1	1
vsum2sws	04	1672	VX	VIU2	4:1
vsum4sbs	04	1800	VX	VIU2	4:1
vsum4shs	04	1608	VX	VIU2	4:1
vsum4ubs	04	1544	VX	VIU2	4:1
vsumsws	04	1928	VX	VIU2	4:1
vupkhpx	04	846	VX	VPU	2:1
vupkhsb	04	526	VX	VPU	2:1
vupkhsh	04	590	VX	VPU	2:1
vupklpx	04	974	VX	VPU	2:1
vupklsb	04	654	VX	VPU	2:1
vupklsh	04	718	VX	VPU	2:1
vxor	04	1220	VX	VIU1	1

[1] Most AltiVec floating-point instructions on the MPC7450 (regardless of Java/non-Java mode) have a 4-cycle latency, unlike the MPC7400 or the MPC7410. However, some VFPU instructions have 2-cycle latency which under some conditions may cause other instructions to have greater than 4-cycle latency (see Section 6.4.5.1.4, “Vector Floating-Point Unit (VFPU) Execution Timing,” for details).

6.7 Instruction Scheduling Guidelines

This section provides an overview of instruction scheduling guidelines, followed by detailed examples showing how to optimize scheduling with respect to various pipeline stages. Performance can be improved by avoiding resource conflicts and scheduling instructions to take fullest advantage of the parallel execution units. Instruction scheduling can be improved by observing the following guidelines:

- To reduce branch mispredictions, separate the instruction that sets CR bits from the branch instruction that evaluates them. Because there can be no more than 24 instructions in the processor (with the instruction that sets CR in CQ0 and the dependent branch instruction executing in IQ7), there is no advantage to having more than 22 instructions between them.

  The MPC7450 branch prediction table is four times larger than the MPC7400’s. This helps prevent aliasing in the BHT, which often reduces prediction accuracy.
- Likewise, when branching to a location specified by the CTR or LR, separate the **mtspr** instruction that initializes the CTR or LR from the dependent branch instruction. This ensures the register values are immediately available to the branch instruction.
- Schedule instructions so three can be dispatched at a time.
- Schedule instructions to minimize stalls due to busy execution units. To avoid branch stalls, MPC7450 has added a third branch prediction buffer over MPC7400’s two. This allows the branch prediction engine to go one level deeper before stalling.
- Avoid scheduling high-latency instructions close together. Interspersing single-cycle latency instructions between longer-latency instructions minimizes the effect that instructions such as integer divide and multiply can have on throughput.
- Avoid using serializing instructions.
- Schedule instructions to avoid dispatch stalls:
  - — Intersperse instructions to maximize the ability to dispatch three GPR instructions, two VR instructions, and one FPR instruction.
  - — 16 instructions can be tracked in the CQ; therefore, 16 instructions can be in the execute stages at any one time
  - — There are 16 GPR rename registers, so only 16 GPRs can be specified as destination operands at any time. If no rename registers are available, instructions cannot enter the execute stage and remain in the reservation station or IQ until they become available.

    Note that load with update address instructions use two destination registers
  - — Similarly, there are 16 FPR rename registers and 16 VR rename registers, so 16 FPR and 16 AltiVec rename register destination operands can be in the execute and complete stages at any time.
- Avoid branches where possible; favor fall-through branches over taken branches.

The following sections give detailed information regarding optimizing code for the MPC7450 pipeline stages.

6.7.1 Fetch/Branch Considerations

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 for resolution. However, it uses the link stack to predict the target address in order to avoid stalling fetch.
- The **bcctr** instruction requires CTR availability.
- The branch conditional on counter decrement and the CR condition requires CTR availability or the CR condition must be false.
- A fourth conditional branch instruction cannot be executed following three unresolved predicted branch instructions.

6.7.1.1 Fetching Examples

Branches that target an instruction at or near the end of a cache block can cause instruction supply problems. Consider a four-instruction loop branch (including the branch) where the entry point is the last word of the cache block. The MPC7450 needs at least 2 cycles to fetch the four instructions because the cache block boundary breaks the four instructions into two accesses. Aligning this loop significantly increases the instruction supply.

Additionally, on the MPC7450 this tight loop encounters the branch-taken bubble problem. That is, the BTIC supplies instructions 1 cycle after the branch executes. For the instructions in the cache block crossing case, this leads to four instructions fetched every 3 cycles. Aligning instructions to be within a cache block increases the number of instructions fetched to 4 every 2 cycles. For longer loops, the branch-taken bubble overhead can be better amortized or in some cases can disappear (because the branch is executed early and the bubble disappears at dispatch time). Software loop unrolling can increase the number of instructions per branch.

NOTE

> The BTIC contains targets for only **b** and **bc** branches. Indirect branches (**bcctr** and **bclr**) must go to the instruction cache for instructions, which incurs an additional cycle of fetch latency (branch-taken bubble).

6.7.1.1.1 Fetch Alignment Example

The following code loop is a simple array accumulation operation.

```
xxxxxx18  loop:   lwzu r10,0x4(r9)
xxxxxx1C          add r11,r11,r10
xxxxxx20          bdnz loop
```

The **lwzu** and **add** are the last two instructions in one cache block, and the **bdnz** is the first instruction in the next. In this example, the fetch supply is the primary restriction. Table 6-9 assumes instruction cache and BTIC hits. The **lwzu**/**add** of the second iteration are available for dispatch in cycle 3, as a result of a

BTIC hit for the **bdnz** executed in cycle 1. The **bdnz** of the second iteration is available in the IQ one cycle later (cycle 4) because the cache block break forced a fetch from the instruction cache. Overall, the loop is limited to one iteration for every 3 cycles.

Table 6-9. Fetch Alignment Example

Instruction	0	1	2	3	4	5	6	7	8	9	10	11
lwzu (1)	D	I	E0	E1	E2	C						
add (1)	D	I	-	-	-	E	C					
bdnz (1)	F2	BE	D	-	-	-	C					
lwzu (2)				D	I	E0	E1	E2	C			
add (2)				D	I	-	-	-	E	C		
bdnz (2)			F1	F2	BE	D	-	-	-	C		
lwzu (3)							D	I	E0	E1	E2	C
add (3)							D	I	-	-	-	E
bdnz (3)						F1	F2	BE	D	-	-	-

Performance can be increased if the loop is aligned so that all three instructions are in the same cache block, as in the following example.

```
xxxxxx00  loop:   lwzu r10,0x4(r9)
xxxxxx04          add r11,r11,r10
xxxxxx08          bdnz loop
```

The fact that the loop fits in the same cache block allows the BTIC entry to provide all three instructions. Table 6-10 shows pipelined execution results (again assuming BTIC and instruction cache hits). While fetch supply is still a bottleneck, it is improved by proper alignment. The loop is now limited to one iteration every 2 cycles, increasing performance by 50%.

Table 6-10. Loop Example—Three Iterations

Instruction	0	1	2	3	4	5	6	7	8	9
lwzu (1)	D	I	E0	E1	E2	C				
add (1)	D	I	-	-	-	E	C			
bdnz (1)	BE	D	-	-	-	-	C			
lwzu (2)			D	I	E0	E1	E2	C		
add (2)			D	I	-	-	-	E	C	
bdnz (2)			BE	D	-	-	-	-	C	
lwzu (3)					D	I	E0	E1	E2	C
add (3)					D	I	-	-	-	E
bdnz (3)					BE	D	-	-	-	-

Loop unrolling and vectorization can further increase performance.

6.7.1.1.2 Branch-Taken Bubble Example

The following code shows how favoring taken branches affects fetch supply.

```
xxxxxx00          lwz r10,0x4(r9)
xxxxxx04          cmpi 4,r10,0x0
xxxxxx08          bne 4, targ
xxxxxx0C          stw r11,0x4(r9)
xxxxxx10  targ    add (next basic block)
```

This example assumes the **bne** is usually taken (that is, most of the data in the array is non-zero). Table 6-11 assumes correct prediction of the **bne**, and cache and BTIC hits.

Table 6-11. Branch-Taken Bubble Example

Instruction	0	1	2	3	4	5	6
lwz	D	I	E0	E1	E2	C	
cmpi	D	I	-	-	-	E	C
bne	BE						
add			D	I	E	-	C

Rearranging the code as follows improves the fetch supply.

```
xxxxxx00          lwz r10,0x4(r9)
xxxxxx04          cmpi 4,r10,0x0
xxxxxx08          beq 4,targ
xxxxxx0C  targ2   add (next basic block)
...
yyyyyy00  targ    stw r11,0x4(r9)
yyyyyy04          b targ2
```

Using the same assumptions as before, Table 6-12 shows the performance improvement. Note that the first instruction of the next basic block (**add**) completes in the same cycle as before. However, by avoiding the branch-taken bubble (because the branch is usually not taken), it also dispatches 1 cycle earlier, so that the next basic block begins executing 1 cycle sooner.

Table 6-12. Eliminating the Branch-Taken Bubble

Instruction	0	1	2	3	4	5	6	7	8	9	10
lwz	D	I	E0	E1	E2	C					
cmpi	D	I	-	-	-	E	C				
beq	BE	D	-	-	-	-	C				
add		D	I	E	-	-	C				

6.7.1.2 Branch Conditionals

The cost of mispredictions increases with pipeline length. The following section shows common problems and suggests how to minimize them.

6.7.1.2.1 Branch Mispredict Example

Table 6-13 uses the same code as the two previous examples but assumes the **bne** mispredicts. The compare executes in cycle 5, which means the branch mispredicts in cycle 6 and the fetch pipeline restarts at that correct target for the add in cycle 7. This particular mispredict effectively costs 7 cycles (**add** dispatches in cycle 2 in Table 6-11 and in cycle 9 in Table 6-13).

Table 6-13. Misprediction Example

Instruction	0	1	2	3	4	5	6	7	8	9	10	11	12
lwz	D	I	E0	E1	E2	C							
cmpi	D	I	-	-	-	E	C						
bne	BE						M						
add								F1	F2	D	I	E	C

6.7.1.2.2 Branch Loop Example

Use CTR whenever possible for branch loops, especially for tight inner loops. After the CTR is loaded (using **mtctr**), a branch dependent on the CTR requires no directional prediction. Additionally, loop termination conditions are always handled correctly, which is not so with the normal branch predictor.

```
xxxxxx18 outer_loop:addi. r6,r6,#FFFF
xxxxxx1C    cmpi 1,r6,#0
xxxxxx20 inner_loop:addic. r7,r7,#FFFF
xxxxxx24    lwzu r10,0x4(r9)
xxxxxx28    add r11,r11,r10
xxxxxx2C    bne inner_loop
xxxxxx30    stwu r11,0x4(r8)
xxxxxx34    xor r11,r11,r11
xxxxxx38    ori r7,r0,#4
xxxxxx3C    bne cr1,outer_loop
```

For this example, assume the inner loop executes four times per outer iteration. The inner loop termination is always mispredicted because the branch predictor learns to predict the inner **bne** as taken, which is wrong every fourth time. Table 6-14 shows that the misprediction causes the outer loop code to be dispatched in cycle 13. If the branch had been correctly predicted as not taken, these instructions would have dispatched 5 cycles earlier in cycle 8.

Table 6-14. Three Iterations of Code Loop

Instruction	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
addi	D	I	E	C											
cmp	D	I	-	E	C										
addic (1)	F2	D	I	E	C										

Table 6-14. Three Iterations of Code Loop (continued)

Instruction	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
lwzu (1)	F2	D	I	E0	E1	E2	C								
add (1)	F2	D	I	-	-	-	E	C							
bne (1)	F2	BE													
addic. (2)				D	I	E	-	C							
lwzu (2)				D	I	E0	E1	E2	C						
add (2)				D	I	-	-	-	E	C					
bne (2)				BE											
addic. (3)						D	I	E	-	C					
lwzu (3)						D	I	E0	E1	E2	C				
add (3)						D	I	-	-	-	E	C			
bne (3)						BE									
addic. (4)								D	I	E	-	C			
lwzu (4)								D	I	E0	E1	E2	C		
add (4)								D	I	-	-	-	E	C	
bne (4)								BE			M				
stwu												F1	F2	D	I
xor												F1	F2	D	I
ori												F1	F2	D	I
bne												F1	F2	BE	

The following code uses CTR, which shortens the loop because the compare test (done by the **addic.** at xxxxxx20) is combined into the **bdnz** branch.

```
xxxxxx1C outer_loop:addic. r6,r6,#FFFF
xxxxxx20 inner_loop:lwzu r10,0x4(r9)
xxxxxx24      add r11,r11,r10
xxxxxx28      bdnz inner_loop
xxxxxx2C      mtctr r7
xxxxxx30      stwu r11,0x4(r8)
xxxxxx34      xor r11,r11,r11
xxxxxx38      bne 0,outer_loop
```

As Table 6-15 shows, the inner loop termination branch does not need to be predicted and is executed as a fall-through branch. Instructions in the outer loop start dispatching in cycle 8, saving 5 cycles over the code in Table 6-14. Note that because **mtctr** is execution serialized, it does not complete until cycle 16; nevertheless, the CTR value is forwarded to the BPU by cycle 11. This early forwarding starts for a **mtctr**/**mtlr** when the instruction reaches reservation station 0 of the IU2 and the source register for the **mtctr**/**mtlr** is available.

Table 6-15. Code Loop Example Using CTR

Instruction	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
addic	D	I	E	C														
lwzu (1)	F2	D	I	E0	E1	E2	C											
add (1)	F2	D	I	-	-	-	E	C										
bdnz (1)	F2	BE	D	-	-	-	-	C										
lwzu (2)				D	I	E0	E1	E2	C									
add (2)				D	I	-	-	-	E	C								
bdnz (2)				BE	D	-	-	-	-	C								
lwzu (3)						D	I	E0	E1	E2	C							
add (3)						D	I	-	-	-	E	C						
bdnz (3)						BE	D	-	-	-	-	C						
lwzu (4)								D	I	E0	E1	E2	C					
add (4)								D	I	-	-	-	E	C				
bdnz (4)								BE	D	-	-	-	-	C				
mtctr									D	I						E	C	
stwu									D	I	E0	-	-	-	-	-	-	C
xor									-	D	I	E	-	-	-	-	-	C
bne									BE									

6.7.1.3 Static versus Dynamic Prediction

Using static prediction (HID0[BHT] = 0) means that the hint bit in the branch opcode predicts the branch and the dynamic predictor (the BHT) is ignored. Sometimes static prediction is superior, either through informed guessing or through available profile-directed feedback. Run-time for code using static prediction is more nearly deterministic, which can be useful in an embedded system.

6.7.1.4 Using the Link Stack for Branch Indirect

On the MPC7450, a **bclr** uses the link stack to predict the target. To use the link stack correctly, each branch-and-link instruction must be paired with a branch-to-link instruction. Using the architecture-defined LR for computed targets corrupts the link stack. In general, the CTR should be used for computed target addresses and the LR should be used only for call/return addresses. If using the CTR for a loop conflicts with a computed GOTO, the computed GOTO should be used and the loop should be converted to a GPR form.

When generating position-independent code, many compilers use an instruction sequence such as the following to obtain the current instruction address.

```
bcl 20,31,$+4
mflr r3
```

Note that this is not a true call and is not paired with a return. The link stack is optimized to ignore position-independent code where the **bcl 20,31,$+4** form is used. This conditional call, which is used only for putting the instruction address in a program-visible register, does not force a push on the link stack and is treated as a not-taken branch.

6.7.1.4.1 Link Stack Example

The following code sequence is a common code sequence for a subroutine call/return sequence, where main calls foo, foo calls ack, and ack possibly calls additional functions (not shown).

```
main:    ...
         mflr  r5
         stwu  r5,-4(r1)
         bl    foo
5        add   r3,r3,r20
         ....
foo:     stwu  r31,-4(r1)
         stwu  r30,-4(r1)
         ....
         mflr  r4
         stwu  r4,-4(r1)
         bl    ack
         add   r3,r3,r6
         ....
0        lwzu  r30,4(r1)
1        lwzu  r31,4(r1)
2        lwzu  r5,4(r1)
3        mtlr  r5
4        bclr
ack:     ....
         (possible calls to other functions)
         ....
         lwzu  r4,4(r1)
         mtlr  r4
         bclr
```

The **bl** in main pushes a value onto the hardware-managed link stack (and to the architecturally-defined LR). Then, the **bl** in foo pushes a second value onto the stack.

When ack later returns through the **bclr**, the hardware link stack is used to predict the value of the LR, if the actual value of the LR is not available when the branch is executed (typically because the **lwzu**/**mtlr** pair has not finished executing). It also pops a value off of the stack, leaving only the first value on the stack. This occurs again with the **bclr** in foo, which returns to main, leaving the stack empty.

Table 6-16 shows the performance implications of the link stack. The following code starts executing from the instruction 0 in procedure foo.

Table 6-16. Link Stack Example

Inst#	Instruction	0	1	2	3	4	5	6	7	8	9	10	11	12
0	**lwzu r**30, 4(**r**1)	F1	F2	D	I	E0	E1	E2	C					
1	**lwzu r**31, 4(**r**1)	F1	F2	-	D	I	E0	E1	E2	C				
2	**lwzu r**5, 4(**r**1)	F1	F2	-	-	D	I	E0	E1	E2	C			

Table 6-16. Link Stack Example (continued)

Inst#	Instruction	0	1	2	3	4	5	6	7	8	9	10	11	12
3	**mtlr**		F1	F2	-	D	I	-	-	-	*	-	E	C
4	**bclr**		F1	F2	BE	D								
...														
5	**add r**3,**r**3,**r**20					F1	F2	D	I	E	-	-	-	C

With link stack prediction, the BPU can successfully predict the target of the **bclr** (instruction 4), which allows the instruction to be executed at the return address (instruction 5) in cycle 8. The IU2 forwards the LR value to the BPU in cycle 9 (which implies that the branch resolution occurs in cycle 10), even though it cannot execute from an execution serialization viewpoint until cycle 11.

Without link stack prediction, the branch would stall on the LR dependency and not execute until after the LR is forwarded (that is, branch execution would occur in cycle 10), which allows instruction 5 not to execute until cycle 15 (seven cycles later than it executes with link stack prediction).

6.7.1.4.2 Position-Independent Code Example

Position-independent code is used when not all addresses are known at compile time or link time. Because performance is typically not good, position-independent code should be avoided when possible. The following example expands on the code sequence, which is described in Section 4.2.4.2, “Conditional Branch Control,” in the *Programming Environments Manuals*.

Table 6-17. Position-Independent Code Example

Inst#	Instruction	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	**bcl** 20, 31, $+4	F1	F2	BE	D	C													
1	**mflr** r2	F1	F2	-	D	I	-	E0	E1	E2	E3	F	C						
2	**addi r**2, **r**2,#constant	F1	F2	-	D	I	-	-	-	-	-	E	C						
3	**mtctr r**2	F1	F2	-	-	D	I	-	-	-	-	-	*	-	E	C			
4	**bcctr**		F1	F2	-	-	-	-	-	-	-	-	-	BE					
...																			
5	**add r**3, **r**3, **r**20														F1	F2	D	I	E

Because a return (**bclr**) is never paired with this **bcl** (instruction 0), the MPC7450 takes two special actions when it recognizes this special form (“**bcl** 20,31,$+4”):

- Although the **bcl** does update the link register as architecturally required, it does not push the value onto the link stack. Not pairing a return with this **bcl** prevents the link stack from being corrupted, which would likely require a later branch mispredict for some later **bclr**.
- Because the branch has the same next instruction address whether it is taken or fall-through, the branch is forced as a fall-through branch. This avoids a potential branch-taken bubble and saves a cycle.

The instruction address is available for executing a subsequent operation (instruction 2, **addi**) in cycle 10, primarily due to the long latency of the execution-serialized **mflr**. However, the data must be transferred back to the BPU through the CTR, which prevents the **bcctr** from executing until cycle 12, so its target instruction (5) cannot start execution until cycle 17.

Note that instructions 3 and 4 must be a **mtctr**/**bcctr** pair rather than a **mtlr**/**bclr** pair. A **bclr** would try to use the link stack to predict the target address, which would almost certainly be an address mispredict, which would be even more costly than the 7-cycle branch execution stall for instruction 4 in this example. In addition, an address mispredict would require the link stack to be flushed, which would mean that **bclr** instructions later in the program would stall rather than use link stack address prediction, further degrading performance.

6.7.1.5 Branch Folding

Branches that do not set the LR or update the CTR are eligible for folding. Taken branches are folded immediately. In the MPC7450, not-taken branches cannot be fall-through folded if they are in IQ0–IQ2; however, branches are removed in the cycle after execution if they are in IQ3–IQ7.

6.7.2 Dispatch Unit Resource Requirements

The following is a list of resources required to avoid stalls in the dispatch unit:

- The appropriate issue queue must be available.
- Sufficient CQ entries must be available.
- Previous instructions in the IQ dispatch entries (IQ0–IQ2) must dispatch.
- Needed rename registers must be available.

The following sections describe how to optimize code for dispatch.

6.7.2.1 Dispatch Groupings

Maximum dispatch throughput is three instructions per cycle. The dispatch process includes a CQ available check, an issue queue available check, a branch ready check, and a rename check.

The dispatcher can send three instructions to the various issues queues, with a maximum of three to the GIQ, two to the VIQ, and one to the FIQ. Thus only two instructions can be dispatched per cycle to the AltiVec units (VIU1, VIU2, VPU, and VFPU). Only one FPU instruction can be dispatched per cycle, so three **fadds** would take three cycles to dispatch.

Only one load/store instruction can dispatch per cycle.

The dispatcher can rename as many as four GPRs, three VRs, and two FPRs per cycle, so a three-instruction dispatch window composed of **vaddfp**, **vaddfp**, and **lvewx** could be dispatched in one cycle.

Note that a load/store update form (for example, **lwzu**), requires a GPR rename for the update. This means that an **lwzu** needs two GPR renames and an **lfdu** needs one FPU rename and one GPR rename. The possibility that one instruction may need two GPR renames means that even though the MPC7450 has a 16-entry CQ and 16 GPR renames, GPR renames could run out even though there is space in the

CQ, as when eight **lwzu** instructions are in the CQ. Eight CQ entries are available, but because all 16 GPR renames are in use, no instruction needing a GPR target can be dispatched.

The restriction to four GPR renames in a dispatch group means that the sequence **lwzu**, **add**, **add** can be dispatched in one cycle. The instruction pair **lwzu**, **lwzu** also uses four GPR renames and passes this rule but is disallowed by the rule that enforces a dispatch of only one load/store per cycle.

6.7.2.1.1 Dispatch Stall Due to Rename Availability

Table 6-18 shows the code example which allows dispatch stall due to rename availability:

Table 6-18. Dispatch Stall Due to Rename Availability

Inst#	Instruction	0	1	2	3	4	5	6	7	8	9	...	25	26	27	28	29	30
0	**divw** **r**4,**r**3,**r**2	F1	F2	D	I	E0	E1	E2	E3	E4	E5	...	E21	E22	C	WB		
1	**lwzu** **r**22,0x04(**r**1)	F1	F2	D	I	E0	E1	E2	-	-	-	...	-	-	C	WB		
2	**lwzu** **r**23,0x04(**r**1)	F1	F2	-	D	I	E0	E1	E2	-	-	...	-	-	-	C	WB	
3	**lwzu** **r**24,0x04(**r**1)	F1	F2	-	-	D	I	E0	E1	E2	-	...	-	-	-	-	C	WB
4	**lwzu** **r**25,0x04(**r**1)		F1	F2	-	-	D	I	E0	E1	E2	...	-	-	-	-	-	C
5	**lwzu** **r**26,0x04(**r**1)		F1	F2	-	-	-	D	I	E0	E1	...	-	-	-	-	-	
6	**lwzu** **r**27,0x04(**r**1)		F1	F2	-	-	-	-	D	I	E0	...	-	-	-	-	-	
7	**lwzu** **r**28,0x04(**r**1)		F1	F2	-	-	-	-	-	D	I	...	-	-	-	-	-	
8	**lwzu** **r**29,0x04(**r**1)			F1	F2	-	-	-	-	-	-	...	-	-	-	-	D	I

Instruction 8 stalls in cycle 9 because it needs 2 renames, and 15 renames are in use (1 for the **divw**, and 2 each for instructions 1 through 7). Since only 16 GPR renames are allowed, instruction 8 cannot be dispatched until at least one rename is released.

When the **div** later completes (cycle 27 in example above), renames are released during the write-back stage and instruction 8 can thus dispatch in cycle 29.

6.7.2.2 Dispatching Load/Store Strings and Multiples

The MPC7450 splits multiples (**lmw** and **stmw**) and strings (**lsw*** and **stsw***) into micro-operations at the dispatch point. The processor can dispatch only one micro-operation per cycle, which does not use the dispatcher to its full advantage.

Using load/store multiple instructions is best restricted to cases where minimizing code size is critical or where there are no other available instructions to be scheduled, such that the under-utilization of the dispatcher is not a consideration.

6.7.2.2.1 Example of Load/Store Multiple Micro-Operation Generation

Consider the following assembly instructions code:

```
0  lmw  r25,0x00(r1)
1  addi r25,r25,0x01
2  addi r26,r26,0x01
3  addi r27,r27,0x01
4  addi r28,r28,0x01
5  addi r29,r29,0x01
6  addi r30,r30,0x01
7  addi r31,r31,0x01
```

The load multiple specified with value 25 loads registers 25–31. The MPC7450 splits this instruction into seven micro-operations at dispatch, after which the **lmw** executes as multiple operations, as Table 6-19 shows.

Table 6-19. Load/Store Multiple Micro-Operation Generation Example

Inst#	Instruction	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0–0	**lmw** **r**25,0x00(**r**1)	F1	F2	D	I	E0	E1	E2	C								
0–1	**lmw** **r**26,0x04(**r**1)	F1	F2	-	D	I	E0	E1	E2	C							
0–2	**lmw** r27,0x08(**r**1)	F1	F2	-	-	D	I	E0	E1	E2	C						
0–3	**lmw** **r**28,0x0C(**r**1)	F1	F2	-	-	-	D	I	E0	E1	E2	C					
0–4	**lmw** **r**29,0x10(**r**1)	F1	F2	-	-	-	-	D	I	E0	E1	E2	C				
0–5	**lmw** **r**30,0x14(**r**1)	F1	F2	-	-	-	-	-	D	I	E0	E1	E2	C			
0–6	**lmw** **r**31,0x1C(**r**1)	F1	F2	-	-	-	-	-	-	D	I	E0	E1	E2	C		
1	**addi** **r**25,**r**25,0x01	F1	F2	-	-	-	-	-	-	D	I	E	-	-	C		
2	**addi** **r**26,**r**26,0x01	F1	F2	-	-	-	-	-	-	D	I	E	-	-	C		
3	**addi** **r**27,**r**27,0x01	F1	F2	-	-	-	-	-	-	-	D	I	E	-	-	C	
4	**addi** **r**28,**r**28,0x01		F1	F2	-	-	-	-	-	-	D	I	E	-	-	C	
5	**addi** **r**29,**r**29,0x01		F1	F2	-	-	-	-	-	-	D	I	E	-	-	C	
6	**addi** **r**30,**r**30,0x01		F1	F2	-	-	-	-	-	-	-	D	I	E	-	-	C
7	**addi** **r**31,**r**31,0x01		F1	F2	-	-	-	-	-	-	-	D	I	-	E	-	C

Because the MPC7450 can dispatch only one LSU operation per cycle, the **lmw** is micro-oped at a rate of 1 per cycle and so in this example takes 7 cycles to dispatch all the operations. However, when the last operation in the multiple is dispatched (cycle 8), instructions 1 and 2 can dispatch along with it.

The use of load/store string instructions is strongly discouraged.

6.7.3 Issue Queue Resource Requirements

Instructions cannot be issued unless the specified execution unit is available. The following sections describe how to optimize use of the three issue queues.

6.7.3.1 GPR Issue Queue (GIQ)

As many as three instructions can be dispatched to the six-entry GPR issue queue (GIQ) per cycle. As many as three instructions can be issued in any order to the LSU, IU2, and IU1 reservation stations from the bottom three GIQ entries.

Issuing instructions out-of-order can help in a number of situations. For example, if the IU2 is busy and a multiply is stalled at the bottom GIQ entry, instructions in the next two GIQ entries can be issued to LSU or IU1s, bypassing that multiply.

The following sequence is not well scheduled, but effectively the MPC7450 dynamically reschedules around the potential multiply bottleneck.

```
0        xxxxxx00          mulhw r10,r20,r21
1        xxxxxx04          mulhw r11,r22,r23
2        xxxxxx08          mulhw r12,r24,r25
3        xxxxxx0C          lwzu r13,0x4(r9)
4        xxxxxx10          add r10,r10,r11
5        xxxxxx14          add r13,r13,r25
6        xxxxxx18          add r14,r5,r4
7        xxxxxx20          subf r15,r6,r4
```

The timing for this sequence (Table 6-20) shows which instructions are in which GIQ entries. Instruction 3 issues out-of-order in cycle 2; instructions 4 and 5 issue out-of-order in cycle 3.

Note that instruction 7 (**subf**) does not issue in cycle 4 because all three IU1 reservation stations have an instruction (4, 5, and 6). Instructions 4 and 5 are waiting in the reservation station for their source registers to be forwarded from the IU2 and LSU, respectively. Because instruction 6 executes immediately after issue (in cycle 5), instruction 7 can issue in that cycle.

Table 6-20. GIQ Timing Example

Inst#	Instruction	0	1	2	3	4	5	6	7	8	9	10	11
0	**mulhw**	D	I	E0	E0	E1	F	C					
1	**mulhw**	D	-	I	-	E0	E0	E1	F	C			
2	**mulhw**	D	-	-	-	I	-	E0	E0	E1	F	C	
3	**lwzu**	-	D	I	E0	E1	E2	-	-	-	-	C	
4	**add**	F2	D	-	I	-	-	-	E	-	-	C	
5	**add**	F2	D	-	I	-	-	E	-	-	-	-	C
6	**add**	F2	-	D	-	I	E	-	-	-	-	-	C

Table 6-20. GIQ Timing Example (continued)

Inst#	Instruction	0	1	2	3	4	5	6	7	8	9	10	11
7	**subf**	F2	-	-	D	-	I	E	-	-	-	-	C

GIQ5											
GIQ4		5									
GIQ3		4	6								
GIQ2	2	3	5	7							
GIQ1	1	2	4	6							
GIQ0	0	1	2	2	7						

Similar examples could also be given for load bypassing adds, multiply bypassing loads.

6.7.3.2 Vector Issue Queue (VIQ)

The four-entry vector issue queue (VIQ) handles all AltiVec computational instructions. Two instructions can dispatch to it per cycle, and it can issue two instructions in-order per cycle from its bottom two entries (VIQ1–VIQ0) if reservation stations are available. The MPC7448 can issue two instructions out-of-order from its bottom two entries, preventing an instruction in VIQ1 destined for VIU1 from waiting for an instruction in VIQ0 that is stalled behind an instruction waiting for operand availability. Moreover, the VIU2, VFPU, and VPU are pipelined, so they can operate on multiple instructions.

Table 6-21 shows two cases where a vector add and a vector multiply-add (**vmsummbm**) start execution simultaneously (cycles 2 and 3). Note that the load-vector instructions go to the GIQ because its address source operands (**r**A and **r**B) are GPRs. This example also shows the MPC7450's ability to dispatch three instructions with vector targets in a cycle (cycles 0 and 1) as well as to retire three instructions with vector targets (cycle 7).

Table 6-21. VIQ Timing Example

Instruction	0	1	2	3	4	5	6	7
vaddshs V20,V24,V25	D	I	E	F	C			
vmsummbm V10,V11,V12,V13	D	I	E0	E1	E2	E3	C	
lvewx V5,**r**5,**r**9	D	I	E0	E1	E2	-	C	
vmsummbm V11,V11,V14,V15	-	D	I	E0	E1	E2	E3	C
vaddshs V21,V26,V27		D	I	E	F	-	-	C
lvewx V5,**r**6,r9		D	I	E0	E1	E2	-	C

6.7.3.3 Floating-Point Issue Queue (FIQ)

The two-entry floating-point issue queue (FIQ) can accept one dispatched instruction per cycle, and if an FPU reservation station is available, it can also issue one instruction from the bottom FIQ entry.

6.7.4 Completion Unit Resource Requirements

The MPC7450 completion queue has 16 entries, so as many as 16 instructions can be in the execution window, not counting branches, which execute from the IQ. The following resources are required to avoid stalls in the completion unit; note that the three completion entries are described as CQ0–CQ2, where CQ0 is at the end of the CQ (see Figure 6-4). Requirements for retiring instructions from CQ0–CQ2 are as follows:

- Instruction must have finished execution.
- Previous instructions must be retired from the CQ retirement entries.
- Instructions in the CQ must not follow an unresolved predicted branch.
- Instructions in CQ1 and CQ2 must not cause an exception.
- Instructions in CQ1 and CQ2 must be integer (IU1 only), floating-point, load, **dcbt**, data streaming, non-folded resolved branches, or AltiVec instructions.
- Number of CR updates from CQ0–CQ2 must not exceed three.
- Number of GPR updates from CQ0–CQ2 must not exceed three.
- Number of FPR updates from CQ0–CQ2 must not exceed three.
- Number of VR updates from CQ0–CQ2 must not exceed three.

6.7.4.1 Completion Groupings

The MPC7450 can retire as many as three instructions per cycle. Only three renames of a given type can be retired per cycle. For example, an **lwzu**, **add**, **subf** sequence has four GPR rename targets and all cannot retire in the same cycle. The **lwzu** and **add** retire first and **subf** retires one cycle later.

6.7.5 Serialization Effects

The MPC7450 supports refetch, execution, and store serialization. Store serialization is described in Section 6.7.6.5.2, “Store Hit Pipeline.”

Refetch serialized instructions include **isync**, **rfi**, **sc**, **mtspr[XER]**, and any instruction that toggles XER[SO]. Refetch serialization forces a pipeline flush when the instruction is the oldest in the machine. Avoid these instructions in performance critical code.

Note that XER[SO] is a sticky bit for XER[OV] updates, so avoiding toggling XER[SO] often means avoiding these instructions (overflow-record, O form).

Execution-serialized instructions wait until the instruction is the oldest in the machine to begin executing. Tables in Section 6.6, “Instruction Latency Summary,” list execution-serialized instructions, which include **mtspr**, **mfspr**, CR logical instructions, and carry consuming instructions (such as **adde**).

Table 6-22 shows the execution of a carry chain. The **addc** executes normally and generates a carry. As an execution-serialized instruction, **adde** must become the oldest instruction (cycle 4) before it can execute (cycle 5). A long chain of carry generation/carry consumption can execute at a rate of one instruction every 3 cycles.

Table 6-22. Serialization Example

Code	0	1	2	3	4	5	6
addc r11,**r**21,**r**23	D	I	E	C			
adde r10,**r**20,**r**22	D	I	-	-	-	E	C

6.7.6 Execution Unit Considerations

The following sections describes how to optimize use of the execution units.

6.7.6.1 IU1 Considerations

Each of the three IU1s has one reservation station in which instructions are held until operands are available. The IU1s allow a potentially large window for out-of-order execution. IU1 instructions can progress until three IU1 instructions are stuck in the three reservation stations, requiring operands (or until the GIQ or dispatcher stalls for other reasons).

Table 6-23 shows a case where, although two IU1s are blocked, the third makes progress.

Also note that some IU1 instructions take more than one cycle and that some are not fully pipelined. The most common 2-cycle instructions are **sraw** and **srawi**.

The following instructions are not fully pipelined when their record bit is set: **extsb**, **extsh**, **rlwimi**, **rlwinm**, **rlwnm**, **slw**, and **srw**. These instructions return GPR data after the first cycle but continue executing into a second cycle to generate the CR result.

Table 6-23 shows **sraw**, **extsh**, and **extsh**. latency effects. The two **sraw** instructions both take 2 cycles to execute, blocking the **extsh**/**extsh**. pair from issuing until cycle 3 but allowing the dependent **add** to execute in cycle 3 (See Table 6-5, footnote 3). Note that **extsh**. takes 2 cycles to execute, but that the dependent **subf** can pick up the forwarded GPR value after the first cycle of execution (cycle 4) and execute in cycle 5.

Table 6-23. IU1 Timing Example

Code	0	1	2	3	4	5	6
sraw r1,r20,r21	D	I	E	E	C		
sraw r2,**r**20,**r**22	D	I	E	E	C		
add r4,**r**2,**r**3	D	I	-	E	C		
extsh r5,**r**25,	F2	D	-	I	E	C	
extsh. r6,**r**26	F2	D	-	I	E	E	C
subf r7,**r**5,**r**6	F2	D	-	I	-	E	C

6.7.6.2 IU2 Considerations

The IU2 has two reservation station entries. Instruction execution is allowed only from the bottom station. Although **mtctr**/**mtlr** instructions are execution-serialized, if data is available, their values are forwarded to the BPU as soon as they are in the bottom reservation station.

Divides, **mulhwu**, **mulhw**, and **mull** are not fully pipelined; they iterate in execution stage 0 and block other instructions from entering reservation station 0. For example, in Table 6-20, the second multiply issues to IU2 in cycle 2. Because the first multiply still occupies reservation station 0, the second is issued to reservation station 1. When the first multiply enters E1, the second moves down to reservation station 0 and begins execution.

Note that the IU2 takes an extra cycle beyond the latencies listed in Table 6-5 to return CR data and finish. This implies that, as the example in Section 6.7.3.1, "GPR Issue Queue (GIQ)," shows, a 3-cycle instruction such as **mulhw** requires a separate finish stage, even though GPR data is still forwarded and used after 3 execution cycles. In the previous example, instruction 4 executed in cycle 7, the cycle after the dependent instruction 2 had gone through its third execution stage.

6.7.6.3 FPU Considerations

The FPU has two reservation station entries. Instruction execution is allowed from only the bottom reservation station (reservation station 0).

Like the IU2, the FPU requires a separate finish stage to return CR and FPSCR data, as shown in Table 6-24. However, FPR data produced in E4 (the fifth stage) is ready and can be forwarded directly (if needed) to an instruction entering E0 in the next cycle.

The five-stage scalar FPU pipeline has a 5-cycle latency. However, when the pipeline contains instructions in stages E0–E3, the pipeline stalls and does not allow a new instruction to start in E0 on the following cycle. This bubble limits maximum FPU throughput to four instructions every 5 cycles, as the following code example shows:

```
xxxxxx00         fadd f10,f20,f21
xxxxxx04         fadd f11,f20,f22
xxxxxx08         fadd f12,f20,f23
xxxxxx0c         fadd f13,f20,f24
xxxxxx10         fadd f14,f20,f25
xxxxxx14         fadd f15,f20,f26
xxxxxx18         fadd f16,f20,f27
xxxxxx1c         fadd f17,f20,f28
xxxxxx20         fadd f18,f20,f29
```

Table 6-24 shows the timing for this sequence.

Table 6-24. FPU Timing Example

Instruction	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
fadd	D	I	E0	E1	E2	E3	E4	F	C								
fadd	-	D	I	E0	E1	E2	E3	E4	F	C							
fadd	-	-	D	I	E0	E1	E2	E3	E4	F	C						

Table 6-24. FPU Timing Example (continued)

Instruction	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
fadd	-	-	-	D	I	E0	E1	E2	E3	E4	F	C					
fadd	F2	-	-	-	D	I	-	E0	E1	E2	E3	E4	F	C			
fadd	F2	-	-	-	-	D	-	I	E0	E1	E2	E3	E4	F	C		
fadd	F2	-	-	-	-	-	D	-	I	E0	E1	E2	E3	E4	F	C	
fadd	F2	-	-	-	-	-	-	-	D	I	E0	E1	E2	E3	E4	F	C
fadd	F1	F2	-	-	-	-	-	-	-	D	I	-	E0	E1	E2	E3	E4

The FPU is also constrained by the number of FPSCR renames. The MPC7450 supports four outstanding FPSCR updates. An FPSCR is allocated in the E3 FPU stage and is deallocated at completion. If no FPSCR rename is available, the FPU pipeline stalls. A fully pipelined case such as that in Table 6-24 is not affected, but if something blocks completion it can become a bottleneck. Consider the following code example:

```
xxxxxx00 lfdu f3,0x8(r9)
xxxxxx04 fadd f11,f20,f22
xxxxxx08 fadd f12,f20,f23
xxxxxx0c fadd f13,f20,f24
xxxxxx10 fadd f14,f20,f25
xxxxxx14 fadd f15,f20,f26
xxxxxx18 fadd f16,f20,f27
xxxxxx1c fadd f17,f20,f28
xxxxxx20 fadd f18,f20,f29
```

The timing for this sequence in Table 6-25 assumes that the load misses in the data cache. Here, after the first four **fadds**, the MPC7450 runs out of FPSCR renames and the pipeline stalls. When the load completes, the pipeline restarts after an additional 2-cycle lag.

Table 6-25. FPSCR Rename Timing Example

Instruction	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lfdu	D	I	E0	E1									C			
fadd	D	I	E0	E1	E2	E3	E4	F	-	-	-	-	C			
fadd	-	D	I	E0	E1	E2	E3	E4	F	-	-	-	C			
fadd	-	-	D	I	E0	E1	E2	E3	E4	F	-	-	-	C		
fadd	F2	-	-	D	I	E0	E1	E2	E3	E4	F	-	-	C		
fadd	F2	-	-	-	D	I	-	E0	E1	E2	E3	E4	E4	E4	E4	F
fadd	F2	-	-	-	-	D	-	I	E0	E1	E2	E3	E3	E3	E3	E4
fadd	F2	-	-	-	-	-	D	-	I	E0	E1	E2	E2	E2	E2	E3
fadd	F1	F2	-	-	-	-	-	-	D	I	E0	E1	E1	E1	E1	E2

Note that denormalized numbers can cause problems for the FPU pipeline, so the normal latencies in Table 6-6 may not apply. Output denormalization in the very unlikely worst case can add as many as 4 cycles of latency. Input denormalization takes 4–6 additional cycles, depending on whether 1, 2, or 3 input source operands are denormalized.

6.7.6.4 Vector Unit Considerations

On the MPC7450, the four vector execution units are fully independent and fully pipelined. Latencies are given in Table 6-26.

Table 6-26. Vector Execution Latencies

Unit	Latency
VIU1	1
VIU2	4
VFPU	4
VPU	2

VFPU latency is usually 4 cycles, but some instructions, particularly the vector float compares and vector float min/max (see Table 6-8 for a list) have only a 2-cycle latency. This can create competition for the VFPU register forwarding bus. This is solved by forcing a partial stall when a bypass is needed. Consider the following code example:

```
xxxxxx20         vaddfp v10,v11,v12
xxxxxx24         vsubfp v11,v14,v13
xxxxxx28         vaddfp v12,v13,v14
xxxxxx2c         vcmpbfp. v13,v18,v19
xxxxxx30         vmaddfp v14,v20,v21,v14
```

Table 6-27 shows the timing for this vector compare bypass/stall situation. In cycle 6 the **vcmp** bypasses from E0 to E3, stalling the **vsubfp** and **vlogefp** for a cycle in stages E1 and E2. Note that an instruction in E1 stalls in E1 under a bypass scenario even if no instruction is in E2.

Table 6-27. Vector Unit Example

Instruction	0	1	2	3	4	5	6	7	8	9	10
vaddfp	D	I	E0	E1	E2	E3	C				
vsubfp	D	-	I	E0	E1	E2	E2	E3	C		
vlogefp	-	D	-	I	E0	E1	E1	E2	E3	C	
vcmpbfp.	-	D	-	-	I	E0	E3	-	-	C	
vmaddfp	F2	-	D	-	-	I	E0	E1	E2	E3	C

6.7.6.5 Load/Store Unit (LSU)

The LSU controls the 32 Kbyte L1 data cache and contains a variety of queues and latches, as shown in Figure 6-18. The memory subsystem interconnect referred to on the diagram is an interface point to the L2 cache, L3 cache, and system bus. Note that L3 cache is not supported on the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448. Note that this is a simplified block diagram, and does not contain the fully detailed LSU microarchitecture.

Instruction execution is allowed only from the lower of the two reservation stations.

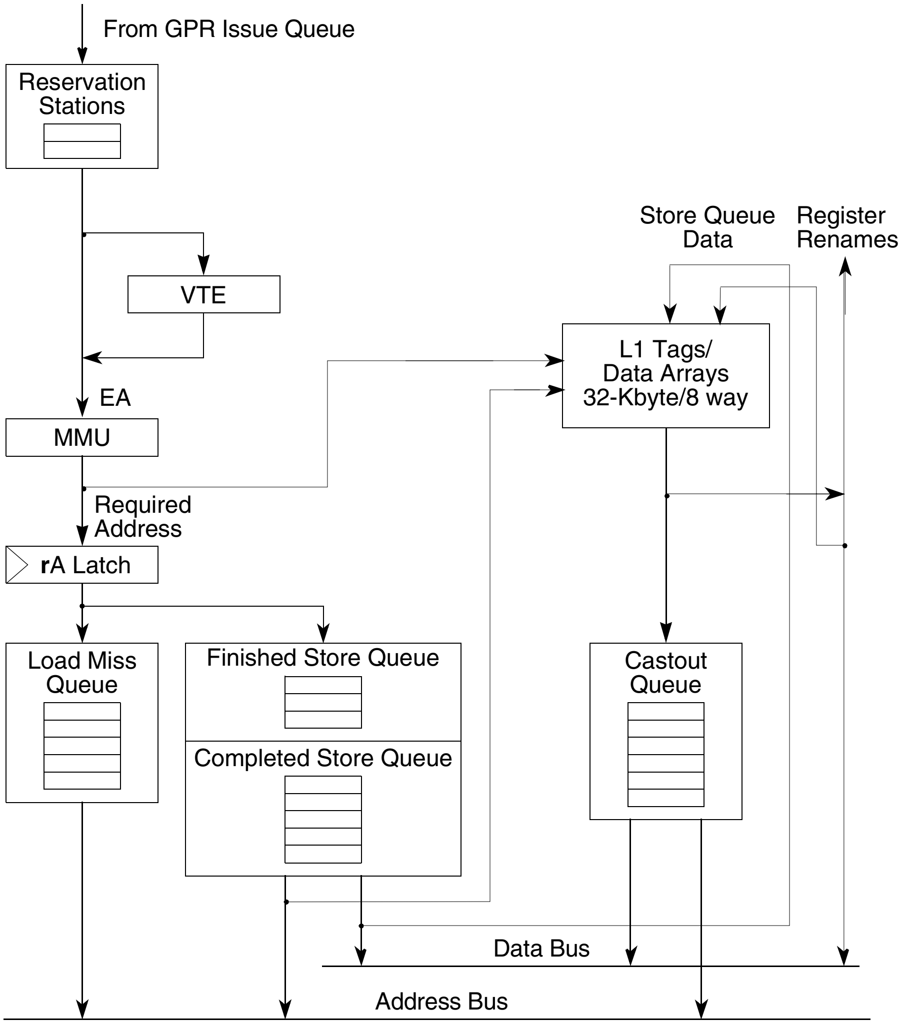


Figure 6-18. LSU Block Diagram

Loads that have all required available source operands can start execution. If the load hits in the data cache, data is forwarded to one of the three register rename types. A floating-point load has one additional cycle of latency (4 cycles) beyond that of an integer or vector load (3 cycles). If a stall or hazard occurs, the instruction is typically held in the required address (**r**A) latch.

Loads that miss go to the 5-entry load miss queue (LMQ), where they are held while the line transaction proceeds to the L2 cache, the L3 cache, and/or the system bus. Critical data forwarding can occur from the L3 cache or system bus to directly update the required rename. A load that receives critical data can finish.

However (for a cacheable load), the LMQ entry can be deallocated only when the full line returns. As the full line is available, the L1 data cache is updated. If an L1 data cache update requires that a line currently in the cache be evicted, that line is cast out and placed into the 6-entry L1 castout queue.

Stores that have required address source operands (**r**A and possibly **r**B) available start execution similar to loads. However, they are transferred to the three entry finished store queue (FSQ). The FSQ holds stores until they have been retired by the completion unit. Once retired, the stores travel through wb0 and wb1, two write-back stages (not shown in Figure 6-18), while acquiring data (**r**S, **fr**S, or **v**S) from the appropriate register file, and are written into the 5-entry committed store queue (CSQ). Stores in the CSQ arbitrate into the L1 data cache. When arbitration is successful, the data is written and the store is removed from the CSQ.

The vector touch engine (VTE) contains the control logic execution of the **dst** instructions.

6.7.6.5.1 Load Hit Pipeline

The following code sequence shows the various normal load latencies:

```
xxxxxx00 lfdu f3,0x8(r10)
xxxxxx04 fadd f1,f3,f4
xxxxxx08 lwzu r3,0x4(r11)
xxxxxx0c add r1,r3,r4
xxxxxx10 subf r5,r11,r6
xxxxxx14 lvewx v3,r12,r13
xxxxxx18 vaddsws v1,v3,v4
```

As Table 6-28 shows, the load-floating-point latency is 4 and the load-integer and load-vector latency are each 3. Although the load has a 4-cycle latency, it also completes on that fourth cycle. The update has an effective latency of 1. The **lwzu** forwards its update target R11 from E0 in cycle 3 to the **subf** instruction, such that it executes in cycle 4.

Table 6-28. Load Hit Pipeline Example

Inst#	Instruction	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	**lfdu**	D	I	E0	E1	E2	E3/C								
1	**fadd**	D	I	-	-	-	-	E0	E1	E2	E3	E4	F	C	
2	**lwzu**	-	D	I	E0	E1	E2	-	-	-	-	-	-	C	
3	**add**	-	D	I	-	-	-	E	-	-	-	-	-	C	
4	**subf**	F2	D	I	-	E	-	-	-	-	-	-	-	-	C
5	**lvewx**	F2	-	D	I	E0	E1	E2	-	-	-	-	-	-	C
6	**vaddsws**	F2	-	D	I	-	-	-	E	F	-	-	-	-	C

6.7.6.5.2 Store Hit Pipeline

The pipeline for stores before the data is written to the cache includes several different queues. A store must go through E0 and E1 to handle address generation and translation. It is then placed in the three-entry finished store queue (FSQ). When the store is the oldest instruction, it can access the store data and update architecture-defined resources (store serialization). From this point on, the store is considered part of architectural state.

However, before the data reaches the data cache, two write-back stages (WB0 and WB1) are needed to acquire the store data and transfer it from the FSQ to the 5-entry committed store queue (CSQ). Arbitration into the data cache from the CSQ is pipelined so a one store per cycle throughput can be maintained. During this arbitration and cache write, stores arbitrate into the data cache from the CSQ and stay there for at least 4 cycles. Table 6-29 shows pipelining of four **stw** instructions to the data cache.

Table 6-29. Store Hit Pipeline Example

Instruction	0	1	2	3	4	5	6	7	8	9	10	11	12	13
stw	D	I	E0	E1	FSQ0/C	WB0	WB1	CSQ0	CSQ0	CSQ0	CSQ0			
stw	-	D	I	E0	E1	FSQ0/C	WB0	WB1	CSQ1	CSQ1	CSQ1	CSQ0		
stw	-	-	D	I	E0	E1	FSQ0/C	WB0	WB1	CSQ2	CSQ2	CSQ1	CSQ0	
stw	-	-	-	D	I	E0	E1	FSQ0/C	WB0	WB1	CSQ3	CSQ2	CSQ1	CSQ0

Floating-point stores are not fully pipelined. The bottleneck is at the FSQ point, where only one floating-point store can be done every 3 cycles. See Table 6-30 for an example execution of four **stfd** instructions. Vector stores do not suffer from this problem and are fully pipelined (similar to the integer stores as shown in Table 6-29).

To avoid floating-point store throughput bottlenecks, avoid strings of back-to-back floating-point stores (like that shown in Table 6-30). Instead, intermix floating-point stores with other instructions wherever possible. For maximum store throughput, use vector stores.

Table 6-30. Execution of Four stfd Instructions

Inst#	Instruction	Cycle Number									
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	**stfd**	D	I	E0	E1	FSQ0/C	WB0	WB1	CSQ0	CSQ0	CSQ0
1	**stfd**	-	D	I	E0	E1	FSQ0	FSQ0	FSQ0/C	WB0	WB1
2	**stfd**	-	-	D	I	E0	E1	FSQ1	FSQ1	FSQ0	FSQ0
3	**stfd**	-	-	-	D	I	E0	E1	FSQ2	FSQ1	FSQ1
		10	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
0	**stfd**	CSQ0									
1	**stfd**	CSQ1	CSQ0	CSQ0	CSQ0						
2	**stfd**	FSQ0/C	WB0	WB1	CSQ1	CSQ0	CSQ0	CSQ0			
3	**stfd**	FSQ1	FSQ0	FSQ0	FSQ0/C	WB0	WB1	CSQ1	CSQ0	CSQ0	CSQ0

6.7.6.5.3 Load/Store Interaction

When loads and stores are intermixed, the stores normally lose arbitration to the cache. A store that repeatedly loses arbitration can stay in the CSQ much longer than 4 cycles, which is not normally a performance problem because a store in the CSQ is effectively part of the architecture-defined state. However, sometimes—including if the CSQ fills up or if a store causes a pipeline stall (as in a partial address alias case of store to load)—the arbiter gives higher priority to the store, guaranteeing forward progress.

Also, accesses to the data cache are pipelined (two stages) such that back-to-back loads and back-to-back stores are fully pipelined (single-cycle throughput). However, a store followed by a load cannot be performed in subsequent clock cycles. Loads have higher priority than stores and the LSU store queues stage store operations until a cache cycle is available. When the LSU store queues become full, stores take priority over subsequent loads.

From an architectural perspective, when a load address aliases to a store address the load needs to read the store data rather than the data in the cache. A store can forward only after acquiring its data, which means forwarding happens only from the CSQ. Additionally, the load address and size must be contained within the store address and size for store forwarding to occur. If the alias is only a partial alias (for example an **stb** and an **lwz**) the load stalls. Table 6-31 shows a forwardable load/store alias, where the load stalls in E1 for 3 cycles until the store arrives in CSQ0 and can forward its data.

Table 6-31. Load/Store Interaction (Assuming Full Alias)

Instruction	0	1	2	3	4	5	6	7	8
stw r3,0x0(**r**9)	E0	E1	FSQ0/C	WB0	WB1	CSQ0	CSQ0	CSQ0	CSQ0
lwz r4,0x0(**r**9)	I	E0	E1	E1	E1	E1	E2	C	

6.7.6.5.4 Misalignment Effects

Misalignment, particularly the back-to-back misalignment of loads, can cause strange performance effects. The MPC7450 splits misaligned transactions into two transactions, so misaligned load latency is at least 1 cycle greater than the default latency. On the MPC7450, misalignment typically occurs when an access crosses a double-word boundary. Table 6-32 shows what is considered misaligned based on the EA of the access. Note that vector transactions ignore non-size-aligned low-order address bits and so are not considered misaligned.

Table 6-32. Misaligned Load/Store Detection

EA[29:31]	Byte	Halfword	Word	Double Word	Quad-Word Bus (Not supported in 60x bus mode)
000	—	—	—	—	—
001	—	—	Multi/floating-point exception	Floating-point exception	Align to QW
010	—	—	Multi/floating-point exception	Floating-point exception	Align to QW
011	—	—	Multi/floating-point exception	Floating-point exception	Align to QW
100	—	—		Misaligned	Align to QW
101	—	—	Misaligned or multi/floating-point exception	Floating-point exception	Align to QW
110	—	—	Misaligned or multi/floating-point exception	Floating-point exception	Align to QW
111	—	Misaligned	Misaligned or multi/floating-point exception	Floating-point exception	Align to QW

Future generations of high-performance microprocessors that implement the PowerPC architecture may experience greater misalignment penalties.

6.7.6.5.5 Load Miss Pipeline

The MPC7450 supports as many as five outstanding load misses in the load miss queue (LMQ). Table 6-33 shows a load followed by a dependent **add**. Here, the load misses in the data cache and the full line are reloaded from the L2 cache back into the data cache. The load L2 cache hit latency is effectively 9 cycles. In the MPC7448, if ECC is disabled, the load access time is 11 cycles. If ECC is enabled, the load access time is 12 cycles. Instruction fetch latency for an L2 hit increases from 13 cycles to 15/16 cycles in the MPC7448.

Table 6-33. Data Cache Miss, L2 Cache Hit Timing

Instruction	0	1	2	3-7	8	9	10
lwz r4,0x0(**r**9)	E0	E1	Miss	LMQ0	LMQ0/E2	C	
add r5,**r**4,**r**3	—	—	—	—	—	E	C

If a load misses in the L1 data cache and in the L2 data cache, critical data forwarding occurs, followed shortly by the rest of the line. The following example shows that the load L3 cache hit latency is effectively 33 cycles. Note that L3 cache is not supported on the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448.

The following L3 parameters are assumed for the example in Table 6-34:

- DDR SRAM at 4:1 L3 bus ratio
- L3 clock sample point is 5 clocks
- L3 P-clock sample point is 0 clocks

Table 6-34. Data Cache Miss, L2 Cache Miss, L3 Cache Hit Timing

Instruction	0	1	2	3–31	32	33	34	35–36
lwz r4,0x0(**r**9)	E0	E1	Miss	LMQ0	LMQ0/E2	LMQ0/C	LMQ0	LMQ0
add r5,**r**4,**r**3	—	—	—	—	—	E	C	—

Note that the LMQ0 entry for the load remained allocated for 4 cycles after the critical data arrived in cycle 32. This is because with a 4:1 DDR SRAM, there is a 4-cycle gap between critical data and full line data, and the LMQ entry is only deallocated when the full line has returned.

If a load/store miss aliases to the same line as a previously outstanding miss, the LSU halts new access until this stall condition is resolved. The example in Table 6-35 contains a series of loads, where the data starts in the L3 cache, with the L3 cache configured similarly to the example in Table 6-34.

Table 6-35. Load Miss Line Alias Example

Inst#	Instruction	Cycle Number							
		0	**1**	**2**	**3–31**	**32**	**33**	**34**	**35–36**
0	**lwz r**3,0x0(**r**9)	E0	E1	Miss	LMQ0	LMQ0/E2	LMQ0/C	LMQ0	LMQ0
1	**add r**4,**r**3,**r**20						E	C	
2	**lwz r**5,0x4(**r**9)	I	E0	E1	E1	E1	E1	E1	E1
3	**add r**6,**r**5,**r**4	I							
4	**lwz r**7,0x20(**r**9)	D	I	E0	E0	E0	E0	E0	E0
5	**add r**8,**r**7,**r**6	D	I						
		37–39	**40**	**41**	**42**	**43–61**	**62**	**63**	**64**
0	**lwz r**3,0x0(**r**9)								
1	**add r**4,**r**3,**r**20								
2	**lwz r**5,0x4(**r**9)	E1	E2	C					
3	**add r**6,**r**5,**r**4			E	C				
4	**lwz r**7,0x20(**r**9)	E0	E1	Miss	LMQ0	LMQ0	LMQ0/E2	LMQ0/C	LMQ0
5	**add r**8,**r**7,**r**6							E	C

Note that instruction 2 stalls in stage E1 (in the **r**A latch in Table 6-35). This stall is due to the fact that the line miss caused by instruction 0 is the same line that instruction 2 requires. Instruction 2 does not finish execution until cycle 40—8 cycles after instruction 0. This delay is due to 2 major components. The first delay component is that instruction 0 finished by using critical forwarded data, whereas instruction 2 must wait for the full cache line to appear before it can start execution (a 4-cycle delay, in this example). The second delay component is also due to the cache being updated and a pipeline restart.

The second problem that this example shows is that the misses are not fully pipelined. Instructions 0 and 4 miss in the data cache and L2 cache but hit in the L3 cache. The stall caused by the line miss alias between instructions 0 and 2 has caused the miss for instruction 4 to delay its access start by many cycles. A simple reordering of the code, as the example in Table 6-36 allows the two load misses to pipeline to the L3 cache, improving performance by nearly 50%.

Table 6-36. Load Miss Line Alias Example With Reordered Code

Inst#	Instruction	Cycle Number						
		0	**1**	**2**	**3**	**4–31**	**32**	**33**
0	**lwz r**3,0x0(**r**9)	E0	E1	Miss	LMQ0	LMQ0	LMQ0/E2	LMQ0/C
1	**add r**4,**r**3,**r**20							E
2	**lwz r**7,0x20(**r**9)	I	E0	E1	Miss	LMQ1	LMQ1	LMQ1
3	**lwz r**5,0x4(**r**9)	D	I	E0	E1	E1	E1	E1
4	**add r**6,**r**5,**r**4	D	I					

Table 6-36. Load Miss Line Alias Example With Reordered Code (continued)

Inst#	Instruction	Cycle Number						
5	**add r**8,**r**7,**r**6	D	I					
		34	**35–36**	**37–39**	**40**	**41**	**42**	**43**
0	**lwz r**3,0x0(**r**9)	LMQ0	LMQ0					
1	**add r**4,**r**3,**r**20	C						
2	**lwz r**7,0x20(**r**9)	LMQ1	LMQ1	LMQ1	LMQ1	LMQ1/E2	LMQ1/C	LMQ1
3	**lwz r**5,0x4(**r**9)	E1	E1	E1	E2		C	
4	**add r**6,**r**5,**r**4					E	C	
5	**add r**8,**r**7,**r**6						E	C

This type of stall is common in some specific kinds of code, including simple data streaming or striding array accesses. For example, a long stream of vector loads with addresses incrementing by 16 bytes (a quad word) per load stalls every other load stalled in this manner, and no miss pipelining occurs. This stall causes an even larger performance bottleneck when cache misses are required to go to the system bus and when missed opportunities to pipeline system bus misses occur. This performance problem can be solved by code reordering as shown in Table 6-36 or by the use of prefetch instructions (**dcbt** or **dst**).

The MPC7450 does back-end allocation of the L1 data cache, which means that it selects the line replacement (and pushes to the six-entry castout queue as needed) only when a reload returns. Because any new miss transaction may later require a castout, a new miss is not released to the memory subsystem until a castout queue slot is guaranteed.

6.7.6.5.6 Store Miss Pipeline

The MPC7450 supports only one outstanding store miss, which uses the committed store queue entry 0 (CSQ0). Having only one outstanding store miss allows no store miss pipelining; this can be a bottleneck. For applications needing considerable store-miss bandwidth to a cacheable memory region and doing read/modify/write operations, consider using **dst** or **dcbtst** instructions to prefetch needed lines. This allows the use of the five-entry LMQ to provide miss pipelining. Note that the MPC7448 supports two outstanding store misses.

However, using **dcbz** instruction is strongly encouraged for storing to a new cache block, when the entire block will be written (and not be read before being written). The **dcbz** creates an address-only transaction that avoids waiting for data to be read from the L2/L3 or the bus and then updating the data cache only to be immediately overwritten by the store. Using **dstst** instruction is discouraged (either the operation is read/modify/write, in which case **dcbt** or **dst** instructions should be used; or the operation is write-only, in which case prefetching the data is a bad idea and a **dcbz** should instead be used).

Table 6-37 shows a series of cacheable stores, where the stores miss in the data cache, L2 cache, and L3 cache and take an arbitrarily long time (N cycles) to return from the main system bus. Instructions 0–7 are all storing data to the same cache line, while instructions 8–9 are storing data to the next adjacent cache line. Instruction 0 data cache access occurs in cycle 7 and the miss is transferred to the lower levels of the cache hierarchy starting in cycle 9.

Table 6-37. Store Miss Pipeline Example

Inst#	Instruction	Cycle Number						
		0	**1**	**2**	**3**	**4**	**5**	**6**
0	**stw r**10,0x0(**r**8)	E0	E1	FSQ0/C	WB0	WB1	CSQ0	CSQ0
1	**stwu r**11,0x4(**r**8)	I	E0	E1	FSQ0/C	WB0	WB1	CSQ1
2	**stwu r**12,0x4(**r**8)	D	I	E0	E1	FSQ0/C	WB0	WB1
3	**stwu r**13,0x4(**r**8)		D	I	E0	E1	FSQ0/C	WB0
4	**stwu r**14,0x4(**r**8)			D	I	E0	E1	FSQ0/C
5	**stwu r**15,0x4(**r**8)				D	I	E0	E1
6	**stwu r**16,0x4(**r**8)					D	I	E0
7	**stwu r**17,0x4(**r**8)						D	I
8	**stwu r**18,0x4(**r**8)							D
9	**stwu r**19,0x4(**r**8)							
		7	**8**	**9**	**10**	**11**	**12**	**13**
0	**stw r**10,0x0(**r**8)	CSQ0	CSQ0	CSQ0	CSQ0	CSQ0	CSQ0	CSQ0
1	**stwu r**11,0x4(**r**8)	CSQ1	CSQ1	CSQ1	CSQ1	CSQ1	CSQ1	CSQ1
2	**stwu r**12,0x4(**r**8)	CSQ2	CSQ2	CSQ2	CSQ2	CSQ2	CSQ1	CSQ1
3	**stwu r**13,0x4(**r**8)	WB1	CSQ3	CSQ3	CSQ3	CSQ3	CSQ2	CSQ2
4	**stwu r**14,0x4(**r**8)	WB0	WB1	CSQ4	CSQ4	CSQ4	CSQ3	CSQ3
5	**stwu r**15,0x4(**r**8)	FSQ0/C	WB0	WB1	WB1	WB1	CSQ4	CSQ4
6	**stwu r**16,0x4(**r**8)	E1	FSQ0	FSQ0	FSQ0	FSQ0	FSQ0/C	WB0
7	**stwu r**17,0x4(**r**8)	E0	E1	FSQ1	FSQ1	FSQ1	FSQ1	FSQ0
8	**stwu r**18,0x4(**r**8)	I	E0	E1	FSQ2	FSQ2	FSQ2	FSQ1
9	**stwu r**19,0x4(**r**8)	D	I	E0	E1	E1	E1	E1
		14	**15**	**16**	**17**	**18**	**19**	**20**
0	**stw r**10,0x0(**r**8)	CSQ0	CSQ0	CSQ0	CSQ0	CSQ0	CSQ0	CSQ0
1	**stwu r**11,0x4(**r**8)	CSQ1	CSQ1	CSQ1	CSQ1	CSQ1	CSQ1	CSQ1
2	**stwu r**12,0x4(**r**8)	CSQ1	CSQ1	CSQ1	CSQ1	CSQ1	CSQ1	CSQ1
3	**stwu r**13,0x4(**r**8)	CSQ2	CSQ2	CSQ1	CSQ1	CSQ1	CSQ1	CSQ1
4	**stwu r**14,0x4(**r**8)	CSQ2	CSQ2	CSQ1	CSQ1	CSQ1	CSQ1	CSQ1
5	**stwu r**15,0x4(**r**8)	CSQ3	CSQ3	CSQ2	CSQ2	CSQ2	CSQ2	CSQ1
6	**stwu r**16,0x4(**r**8)	WB1	CSQ4	CSQ3	CSQ3	CSQ2	CSQ2	CSQ1
7	**stwu r**17,0x4(**r**8)	FSQ0/C	WB0	WB1	CSQ4	CSQ3	CSQ3	CSQ2
8	**stwu r**18,0x4(**r**8)	FSQ1	FSQ0	FSQ0/C	WB0	WB1	CSQ4	CSQ3

Table 6-37. Store Miss Pipeline Example (continued)

Inst#	Instruction	Cycle Number						
9	**stwu r**19,0x4(**r**8)	FSQ2	FSQ1	FSQ1	FSQ0	FSQ0/C	WB0	WB1
		21	**22**	**23–24**	**25**	**26–N**	**N+1–N+4**	**N+5**
0	**stw r**10,0x0(**r**8)	CSQ0	CSQ0	CSQ0	CSQ0	CSQ0		
1	**stwu r**11,0x4(**r**8)	CSQ1	CSQ1	CSQ1	CSQ1	CSQ1	CSQ0	
2	**stwu r**12,0x4(**r**8)	CSQ1	CSQ1	CSQ1	CSQ1	CSQ1	CSQ0	
3	**stwu r**13,0x4(**r**8)	CSQ1	CSQ1	CSQ1	CSQ1	CSQ1	CSQ0	
4	**stwu r**14,0x4(**r**8)	CSQ1	CSQ1	CSQ1	CSQ1	CSQ1	CSQ0	
5	**stwu r**15,0x4(**r**8)	CSQ1	CSQ1	CSQ1	CSQ1	CSQ1	CSQ0	
6	**stwu r**16,0x4(**r**8)	CSQ1	CSQ1	CSQ1	CSQ1	CSQ1	CSQ0	
7	**stwu r**17,0x4(**r**8)	CSQ2	CSQ1	CSQ1	CSQ1	CSQ1	CSQ0	
8	**stwu r**18,0x4(**r**8)	CSQ3	CSQ2	CSQ2	CSQ2	CSQ2	CSQ1	CSQ0
9	**stwu r**19,0x4(**r**8)	CSQ4	CSQ3	CSQ3	CSQ2	CSQ2	CSQ1	CSQ0

In this example a store gathering opportunity occurs as a store miss has happened. See Section 6.4.4.2, “Store Gathering,” and Section 3.1.2.3, “Store Gathering/Merging,” for rules about how and when a store gathering opportunity can be taken advantage of. To maximize the potential for store gathering, stores to adjacent datum should not be interleaved with other stores.

The first store gathering occurs in cycle 12, where adjacent stores instructions 1 and 2 are gathered to the same CSQ entry (1). The gathering also occurs in cycles 14, 16, 18, 20, and 22. In cycle 22, an entire cache line has gathered except for the first access that caused the miss. The second cache line (for instructions 8 and 9) access gathers in cycle 25. Given a sufficiently long miss latency for instruction 0, a full line could have been gathered into entry CSQ2, if instructions 10–15 had been shown and also were storing to the same line as instructions 8–9.

As the data reload occurs in cycle N, instruction 0 is removed from the CSQ. It takes another 4 cycles for the next store access (the gathered access of instructions 1–7) to restart and write into the cache.

6.7.6.5.7 DST Instructions and the Vector Touch Engine (VTE)

The MPC7450 VTE engine is similar to that on the MPC7400 but can only initiate an access every 3 cycles rather than 2. However, due to miss-handling differences described in Section 6.7.6.5.5, “Load Miss Pipeline,” the engine may fall behind and conflict with the processor work. Therefore, retuning the **dst** may be necessary to optimize MPC7450 performance.

Also, note the information on hardware prefetching in Section 6.7.7.3, “Hardware Prefetching.” Although hardware prefetching is useful for many general-purpose applications, it may not be the best choice for when active prefetch control through software is attempted. Hardware prefetching can sometimes interfere with the **dst** engine’s attempt to keep the bus busy with specific prefetch transactions, especially for **dst** strides larger than one cache block or transient **dst** operations. Experimentation is encouraged, but in this instance the best solution may be to disable hardware prefetching.

6.7.7 Memory Subsystem Considerations

The three-level cache implementation (two-level in the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448) affects instruction fetching, the loading and storing source, and destination operands, as described in the following sections

6.7.7.1 L2 Cache Effects

For the MPC7450, the unified 256-Kbyte on-board L2 cache (512-Kbyte for MPC7457 and 1 Mbyte for the MPC7448) has 8-way set associativity and 64-byte lines (with two sectors/line). This implies 4096 lines (8192 for the MPC7447, MPC7457, and MPC7447A, 16384 for the MPC7448) and 512 sets (1024 for the MPC7447, MPC7457, and MPC7447A, 2048 for the MPC7448). Each line has two sectors with one tag per line but separate valid and dirty bits for each sector. Because of the sectoring, code uses more of the L2 storage if spatial locality is characterized by use of the adjacent 32-byte line.

A load that misses the L1 but hits the L2 causes a full line reload. Its latency is ideally 9 cycles (6 more than for an L1 hit) assuming higher priority L2 traffic. See Table 6-33.

An access missing the L2 goes to the L3 or main memory bus to fetch the needed 32-byte sector.

The L2 cache uses a pseudo-random replacement algorithm. With 8-way set associativity, a miss randomly replaces 1 of 8 ways. This works well for smaller working set sizes, but for working set sizes close to the size of the cache, the hit rate is not quite as good. Imagine a 64-Kbyte array structure and a byte striding access pattern that loops over the array several times. The access of the first 32 Kbytes (256 Kbyte/8 ways) misses and loads correctly, but the second 32 Kbytes has a 1 in 8 chance per set of knocking out an index of the first 32 Kbytes. This means that the first pass is likely to leave 93.75% of the 64-Kbyte structure in the L2 cache; a second pass is likely to leave 99.8% of the structure in the L2 cache.

For a 128-Kbyte object, 82.8% is left in the L2 cache after one pass, but a 256-Kbyte object only slightly less than 2/3 of the structure is left in the L2 cache. However, in both cases the percentages improve with subsequent passes.

6.7.7.2 L3 Cache Effects

The L3 cache is an off-chip SRAM with on-chip cache tags. The MPC7450 supports 1- and 2-Mbyte L3 caches. A 1-Mbyte cache is two-sectored (64-byte lines) and a 2-Mbyte cache is 4-sectored (128-byte lines). The L3 is 8-way set associative, implying 16,384 lines (1 Mbyte/64 or 2 Mbyte/128) or 2,048 sets (1 Mbyte/64/8 or 2 Mbyte/128/8).

An access missing in the L3 fetches the required 32-byte sector regardless of the L3 line size. Like the L2, the L3 uses a random replacement algorithm, the implications of which are described in Section 6.7.7.1, “L2 Cache Effects.” Note that the L3 cache is not supported on the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448.

6.7.7.3 Hardware Prefetching

The MPC7450 supports alternate sector prefetching from the L2 cache. Because the L2 cache is two-sectored, an access requesting a 32-byte line from the L1 that also misses in the L2 and the L3, can generate a prefetch (if enabled) for the alternate sector as needed. As many as three outstanding prefetches are allowed.

The example shown in Table 6-35 can also be used to illustrate the benefits of hardware prefetching for code when other software techniques are not applied.

The example in Table 6-38 shows timing when the loads miss all levels of the cache hierarchy and go to the system bus. Hardware prefetching is disabled. The load misses to the bus are serialized by the load miss line alias stall (instruction 2 on instruction 0).

Table 6-38. Timing for Load Miss Line Alias Example

Inst#	Instruction	Cycle Number							
		0	**1**	**2**	**3–81**	**82**	**83**	**84**	**85–99**
0	**lwz r**3,0x0(**r**9)	E0	E1	Miss	LMQ0	LMQ0/E2	LMQ0/C	LMQ0	LMQ0
1	**add r**4,**r**3,**r**20						E	C	
2	**lwz r**5,0x4(**r**9)	I	E0	E1	E1	E1	E1	E1	E1
3	**add r**6,**r**5,**r**4	I							
4	**lwz r**7,0x20(**r**9)	D	I	E0	E0	E0	E0	E0	E0
5	**add r**8,**r**7,**r**6	D	I						
		100–102	**103**	**104**	**105**	**106–184**	**185**	**186**	**187**
0	**lwz r**3,0x0(**r**9)								
1	**add r**4,**r**3,**r**20								
2	**lwz r**5,0x4(**r**9)	E1	E2	C					
3	**add r**6,**r**5,**r**4			E	C				
4	**lwz r**7,0x20(**r**9)	E0	E1	Miss	LMQ0	LMQ0	LMQ0/E2	LMQ0/C	LMQ0
5	**add r**8,**r**7,**r**6							E	C

However, if hardware prefetching is enabled, hardware starts prefetching the line desired by instruction 4 before instruction 4 accesses (and misses) in the L1 data cache, thus parallelizing some serialized bus accesses. In the example shown in Table 6-39, with prefetching enabled, performance improves by about 40%. In this case, the prefetch was not finished when instruction 4 went to the L2 cache, so the load is forced to stall while the prefetch bus access completes. However, in other cases, the hardware prefetch is entirely finished, allowing subsequent loads to have the access time of an L2 cache hit. In general, hardware prefetch benefits are very dependent on what type of applications are run and how the system is configured.

Table 6-39. Hardware Prefetching Enable Example

Inst#	Instruction	Cycle Number							
		0	**1**	**2**	**3–81**	**82**	**83**	**84**	**85–99**
0	**lwz r**3,0x0(**r**9)	E0	E1	Miss	LMQ0	LMQ0	LMQ0/E2	LMQ0/C	LMQ0

Table 6-39. Hardware Prefetching Enable Example (continued)

Inst#	Instruction	Cycle Number							
1	**add** **r**4,**r**3,**r**20							E	C
2	**lwz** **r**5,0x4(r9)	I	E0	E1	E1	E1	E1	E1	E1
3	**add** **r**6,**r**5,**r**4	I							
4	**lwz** r7,0x20(**r**9)	D	I	E0	E0	E0	E0	E0	E0
5	**add** **r**8,**r**7,**r**6	D	I						
		100–102	**103**	**104**	**105**	**106–133**	**134**	**135**	**136**
0	**lwz** **r**3,0x0(**r**9)								
1	**add** **r**4,**r**3,**r**20								
2	**lwz** **r**5,0x4(**r**9)	E1	E2	C					
3	**add** **r**6,**r**5,**r**4			E	C				
4	**lwz** **r**7,0x20(**r**9)	E0	E1	Miss	LMQ0	LMQ0	LMQ0/E2	LMQ0/C	LMQ0
5	**add** **r**8,**r**7,**r**6							E	C

Hardware prefetching is often preferable. However, sometimes an unnecessary prefetch transaction can delay a later-arriving demand transaction and slow down the processor. Also, as described in Section 6.7.6.5.7, “DST Instructions and the Vector Touch Engine (VTE),” if software prefetching is used, hardware prefetching may sometimes provide more interference than benefit.

Chapter 7
AltiVec Technology Implementation

The AltiVec technology, a short vector parallel architecture, extends the instruction set architecture (ISA) of the PowerPC architecture. The AltiVec ISA is based on separate vector/SIMD-style (single instruction stream, multiple data streams) execution units that have high-data parallelism. That is, the AltiVec technology operations can perform on multiple data elements in a single instruction. The term ‘vector’ in this document refers to the spatial parallel processing of short, fixed-length, one-dimensional matrices performed by an execution unit. It should not be confused with the temporal parallel (pipelined) processing of long, variable-length vectors performed by classical vector machines. High degrees of parallelism are achievable with simple, in-order instruction dispatch and low instruction bandwidth. However, the ISA is designed to not impede additional parallelism through superscalar dispatch in multiple execution units or multithreaded execution unit pipelines. Note that the L3 cache is not supported by the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448.

The MPC7448 adds support for out-of-order issue of AltiVec instructions. Instructions can be issued out-of-order from the bottom two VIQ entries (VIQ1–VIQ0). An instruction in VIQ1 destined for VIU1 does not have to wait for an instruction in VIQ0 that is waiting for operand availability.

The AltiVec specification is defined in the *AltiVec Technology Programming Environments Manual*. That document describes but does not require many aspects of a preferred implementation. The MPC7450 implements the following key features of preferred implementation:

- All data paths and execution units are 128 bits wide.
- There are four independent AltiVec subunits for executing AltiVec instructions: permute, complex, simple, and float.
- The memory subsystem is redesigned to provide high bandwidth.
- The data stream touch instructions, **dst**(**t**) (for loads) and **dstst**(**t**) (for stores) are implemented in their full, four-tag form.

The AltiVec instruction set both defines entirely new resources and extends the functionality of the PowerPC architecture. These changes are described in the following sections.

7.1 AltiVec Technology and the Programming Model

The following sections describe how the AltiVec technology affects features of the programming model as described in Chapter 2, “Programming Model.” Although the AltiVec specification describes four optional user-mode SPRs for thread management, the MPC7450 does not implement these registers.

7.1.1 Register Set

The incorporation of AltiVec technology affects the register set in the MPC7450 as described in the following sections. These features are detailed in the *AltiVec Programming Environments Manual.*

7.1.1.1 Changes to the Condition Register

AltiVec vector-compare operations with Rc set can update condition register field 6 (CR[6]) in user mode.

7.1.1.2 Addition to the Machine State Register

The AltiVec available bit, MSR[VEC], indicates the availability of the AltiVec instruction set. Its default state for the MPC7450 is a zero (not available). It can be set by the supervisor-level **mtmsr** instruction.

7.1.1.3 Vector Registers (VRs)

The AltiVec programming model defines vector registers (VRs) that are used as source and destination operands for AltiVec load, store, and computational instructions.

Figure 7-1 shows the 32 registers of the vector register file (VRF). Each is 128 bits wide and can hold sixteen 8-bit elements, eight 16-bit elements, or four 32-bit elements.

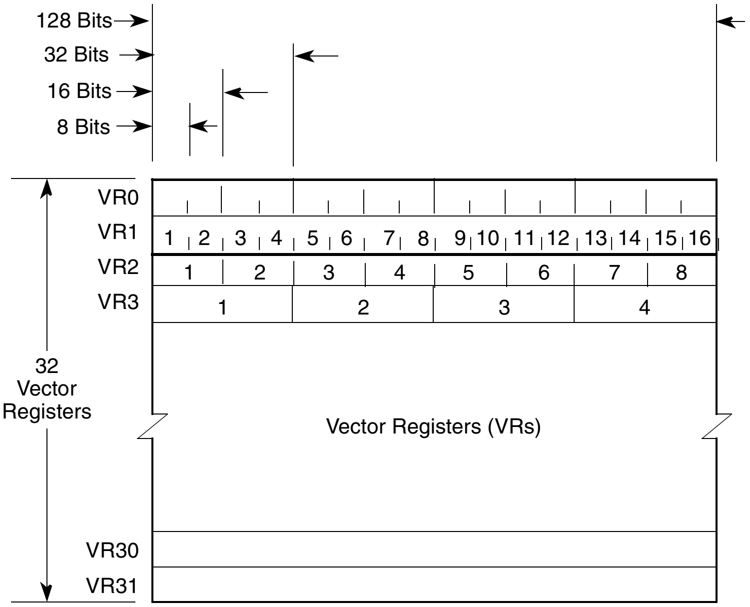


Figure 7-1. Vector Registers (VRs)

7.1.1.4 Vector Status and Control Register (VSCR)

The vector status and control register (VSCR) is a 32-bit vector register (not an SPR) that functions similarly to the FPSCR and is accessed by AltiVec instructions. The Move from Vector Status and Control Register (**mfvscr**) and Move to Vector Status and Control Register (**mtvscr**) instructions are provided to move the contents of the VSCR from and to the least-significant bits of a vector register. The VSCR is shown in Figure 7-2.

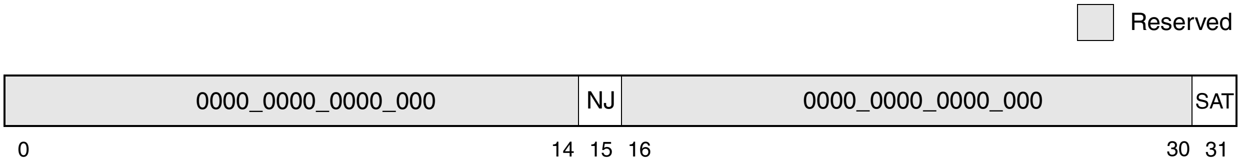


Figure 7-2. Vector Status and Control Register (VSCR)

The VSCR has two defined bits, the AltiVec non-Java mode bit (VSCR[NJ]) and the AltiVec saturation bit (VSCR[SAT]). The remaining bits are reserved.

VSCR bits are described in Table 7-1.

Table 7-1. VSCR Field Descriptions

Bits	Name	Description
0–14	—	Reserved. The handling of reserved bits is the same as that for other PowerPC registers. Software is permitted to write any value to such a bit. A subsequent reading of the bit returns 0 if the value last written to the bit was 0 and returns an undefined value (0 or 1) otherwise.
15	NJ	Non-Java. This bit determines whether AltiVec floating-point operations are performed in a Java-compliant mode or a possibly faster non-Java mode. 0 Java–compliant mode (default). In this mode, the AltiVec assist exception is enabled. The AltiVec assist exception allows software to handle denormalized values as specified in the Java standard. 1 Non-Java mode. If an element in a source vector register contains a denormalized value, the value 0 is used instead. If an instruction causes an underflow condition, the corresponding element in the target VR is cleared to 0. In both cases the 0 has the same sign as the denormalized or underflowing value.

Table 7-1. VSCR Field Descriptions (continued)

Bits	Name	Description
16–30	—	Reserved. The handling of reserved bits is the same as that for other PowerPC registers. Software is permitted to write any value to such a bit. A subsequent reading of the bit returns 0 if the value last written to the bit was 0 and returns an undefined value (0 or 1) otherwise.
31	SAT	Saturation. This sticky status bit indicates that a field in a saturating instruction saturated since the last time SAT was cleared. It is sticky in that when SAT = 1, it remains set to 1 until it is cleared to 0 by an **mtvscr** instruction. 0 Indicates no saturation occurred; **mtvscr** can explicitly clear this bit. 1 The AltiVec saturate instruction is set when saturation occurs for the results of one of the AltiVec instructions having ‘saturation’ in its name, as follows: Move To VSCR (**mtvscr**) Vector Add Integer with Saturation (**vaddubs**, **vadduhs**, **vadduws**, **vaddsbs**, **vaddshs**, **vaddsws**) Vector Subtract Integer with Saturation (**vsububs**, **vsubuhs**, **vsubuws**, **vsubsbs**, **vsubshs**, **vsubsws**) Vector Multiply-Add Integer with Saturation (**vmhaddshs**, **vmhraddshs**) Vector Multiply-Sum with Saturation (**vmsumuhs**, **vmsumshs**, **vsumsws**) Vector Sum-Across with Saturation (**vsumsws**, **vsum2sws**, **vsum4sbs**, **vsum4shs**, **vsum4ubs**) Vector Pack with Saturation (**vpkuhus**, **vpkuwus**, **vpkshus**, **vpkswus**, **vpkshss**, **vpkswss**) Vector Convert to Fixed-Point with Saturation (**vctuxs**, **vctsxs**)

7.1.1.5 Vector Save/Restore Register (VRSAVE)

The vector save/restore register (VRSAVE) is a user-mode register used to assist application and operating system software in saving and restoring the architectural state across process context-switched events. Shown in Figure 7-3, VRSAVE is a 32-bit special-purpose register (SPR 256) entirely maintained and managed by software.

VR0	VR1	VR2	VR3	VR4	VR5	VR6	VR7	VR8	VR9	VR10	VR11	VR12	VR13	VR14	VR15	VR16	VR17	VR18	VR19	VR20	VR21	VR22	VR23	VR24	VR25	VR26	VR27	VR28	VR29	VR30	VR31
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Figure 7-3. Vector Save/Restore Register (VRSAVE)

VRSAVE bit settings are shown in Table 7-2.

Table 7-2. VRSAVE Bit Settings

Bits	Name	Description
0–31	VR	Determine which VRs are used in the current process. 0 Not being used for the current process 1 Used for the current process

7.1.2 AltiVec Instruction Set

The MPC7450 implements all of the defined AltiVec instructions. The AltiVec instruction set has no optional instructions; however, a few instructions associated with the load/store model are defined to allow significant differences between implementations. The following sections describe the MPC7450’s implementation of these options.

AltiVec instructions are primarily user-level and are divided into the following categories:

- Vector integer arithmetic instructions—These include arithmetic, logical, compare, rotate, and shift instructions.
- Vector floating-point arithmetic instructions
- Vector load and store instructions
- Vector permutation and formatting instructions—These include pack, unpack, merge, splat, permute, select, and shift instructions.
- Processor control instructions—These instructions are used to read and write from the VSCR.
- Memory control instructions—These instructions are used for managing caches (user- and supervisor-level).

7.1.2.1 LRU Instructions

The AltiVec architecture suggests that the **lvxl** and **stvxl** instructions differ from other AltiVec load and store instructions in that they leave data cache entries in a least recently used (LRU) state instead of a most recently used state (MRU). This is used to identify data known to have little reuse and poor caching characteristics.

On the MPC7450, these instructions follow the cache allocation and replacement policies described in Section 3.5, “L1 Cache Operation,” but they leave their addressed data cache entries in the LRU state. In addition, all LRU instructions are also interpreted to be transient and are treated as described in Section 7.1.2.2, “Transient Instructions and Caches.”

7.1.2.2 Transient Instructions and Caches

The MPC7450 supports both static and transient memory access behavior as defined by the AltiVec technology.

A static memory access assumes a reasonable degree of locality and that the data will be needed several times over a relatively long period. A transient memory reference has poor locality and is likely to be referenced few times or over a short period of time.

For transient memory accesses that miss in the L1 cache, the MPC7450 allocates (and loads) the line in the L1 cache and marks it as LRU. The MPC7450 does not allocate entries in the L2 or L3 cache for transient accesses that miss. If the L1 cache line is modified and the line is the next candidate for replacement, a castout occurs to system memory. Note that the MPC7450 writes back to memory in this case and does not allocate the L1 castout in the L2 or L3 cache.

The following instructions are interpreted to be transient:

- **lvxl** and **stvxl**
- **dstt** and **dststt** (transient forms of the two-data stream touch instructions). These are described in detail in the following section.

The AltiVec architecture specifies the data stream touch instructions **dst(t)** and **dstst(t)**, and it specifies two data stream stop (**dss(all)**) instructions. The MPC7450 implements all of them. The term **dst***x* used below refers to all of the data stream touch instructions. The T field in the **dst***x* instruction is used as the transient hint bit indicator.

The instructions summarized in this section provide user-level programs with the ability to manage on-chip caches; see Chapter 5, “Cache Model and Memory Coherency,” in the *Programming Environments Manual* for more information about cache topics.

Bandwidth between the processor and memory is managed explicitly through the use of cache management instructions that provide a way to indicate to the cache hardware how it should prefetch and prioritize the writeback of data. The principal instruction for this purpose is the software-directed cache prefetch Data Stream Touch (**dst**). Other related instructions are provided for complete control of the software-directed cache prefetch mechanism.

Table 7-3 summarizes the directed prefetch cache instructions defined by the AltiVec VEA. Note that these instructions are accessible to user-level programs.

Table 7-3. AltiVec User-Level Cache Instructions

Name	Mnemonic	Syntax	Implementation Notes
Data Stream Touch (non-transient)	**dst**	**r**A,**r**B,STRM	—
Data Stream Touch (transient)	**dstt**	**r**A,**r**B,STRM	Used for last access
Data Stream Touch for Store (non-transient)	**dstst**	**r**A,**r**B,STRM	Not recommended for use in the MPC7450
Data Stream Touch for Store (transient)	**dststt**	**r**A,**r**B,STRM	Not recommended for use in the MPC7450
Data Stream Stop (one stream)	**dss**	STRM	—
Data Stream Stop (all streams)	**dssall**	STRM	—

7.1.2.3 Data Stream Touch Instructions

Note that, in general, prefetching data to which the program is performing only store instructions does not help and can sometimes hinder performance. User-level programs should not use the touch-for-store prefetches (**dstt**, **dstst**, and **dststt**) unless the program is performing loads and stores to the data that is being prefetched. If the user is performing only stores to the data, then performance is almost certainly better if the data is not prefetched and the stores are performed independently. In this case, a **dcbz** instruction is often the best method to initialize the cache block without creating an external memory access request.

So, in general, touch-for-store instructions (**dstt**, **dstst**, and **dststt**) should not be used. The only exception in using touch-for-store instructions is when prefetching data that is going to be both loaded from and then stored to. Otherwise, a programmer should use the normal touch-for-load instruction (**dst**) to prefetch data that the program is loading.

If HID0[NOPDST] = 1, all subsequent **dst***x* instructions are treated as no-ops, and all previously executed **dst** streams are canceled. This no-op means that the touch does not cause a load operation and cannot perform address translation. Therefore, no table search operations are initiated, and no page table entry (PTE) referenced bits are set.

The **dst***x* instructions are broken into one or more self-initiated **dcbt**-like touch line fetches by the memory subsystem. When the **dst***x* instruction is dispatched to the LSU and all of its operands are available, the **dst***x* is queued in a vector-touch queue (VTQ) in the next cycle. There are four data stream engines within the VTQ—data stream 0 uses engine VT0 within the VTQ, data stream 1 uses VT1, and so forth.

The operation of a VT data stream engine does not consume any dispatch or completion resources. A VT is an asynchronous line-fetch or line-touch engine that can prefetch data in units of 32-byte cache blocks by inserting touch requests into the normal load/store pipeline.

After the **dst***x* is queued in the VTQ, the VTQ begins to unroll the stream into 32-byte line touches. As early as the third cycle after the LSU sends its request to the VTQ, the VTQ could make its first line-fetch touch request to the data cache.

Note that a data stream engine bases its accesses on effective addresses. This means that each line fetch within a stream accesses the data MMU simultaneously with the L1 data cache and performs a normal translation. There are no arbitrary address boundaries that affect the progress of a given stream.

In addition, if a VTQ line touch accesses a page whose translation does not reside in the data MMU, a table search operation is performed to load that PTE into the data TLB. The TLB is non-blocking during a VTQ-initiated table search operation, meaning that normal loads and stores can hit in the TLB (and in the data cache) during the table search. For details on a table search operation see Section 5.5.2.1, “Conditions for a Page Table Search Operation.”

7.1.2.3.1 Stream Engine Tags

The opcodes for the **dst***x* instructions are shown in Table 7-4.

Table 7-4. Opcodes for dst*x* Instructions

Name	0 5		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
dst	0111_11	0	00	STRM	A	B	01_0101_0110	0
dstst	0111_11	0	00	STRM	A	B	01_0111_0110	0
dststt	0111_11	1	00	STRM	A	B	010_111_0110	0
dstt	0111_11	1	00	STRM	A	B	01_0101_0110	0

The STRM field in the **dst***x* instruction designates which of the four data stream engines (VT0, VT1, VT2, or VT3) is used by a given instruction, as described in Table 7-5.

Table 7-5. DST[STRM] Description

Value of STRM Field in dst*x* Instruction	Data Stream Engines (VTs)
00	VT0
01	VT1
10	VT2
11	VT3

Bits 7 and 8 of the **dst***x* opcode are reserved. If bit 7 is set, it is ignored. If bit 8 is set, the VTQ does not queue up the stream and that **dst***x* instruction is ignored.

7.1.2.3.2 Speculative Execution and Pipeline Stalls for Data Stream Instructions

Like a load miss instruction or a **dcbt**/**dcbtst** instruction, a **dst***x* instruction is executed speculatively. If the target of a particular **dst***x* line fetch is mapped with G = 1 (guarded), any reload for that line fetch is under the same constraints as a guarded load. If any of the four data stream engines encounter a TLB miss, all four pause until the **dst***x* access that caused the TLB miss is retired from the completion queue or is the oldest instruction in the queue. The **dst***x* then initiates a table search operation and completes its current cache access.

If a **dst***x* instruction to a given data stream is dispatched and the VTQ is processing a previous **dst***x* to the same data stream, the second **dst** to that tag supersedes the first one, but only after the second **dst***x* becomes non-branch-speculative; it can still be speculative with respect to exceptions. If a third **dst***x* is ready for dispatch while the second is waiting for branch speculation to resolve, instruction dispatch stalls.

7.1.2.3.3 Static/Transient Data Stream Touch Instructions

The AltiVec ISA defines two of the **dst***x* instructions as static (**dst** and **dstst**) and two as transient (**dstt** and **dststt**). Static data is likely to have a reasonable degree of locality and is referenced several times or over a reasonably long period of time. Transient data is assumed to have poor locality and is likely to be referenced only a few times over a short period of time.

The MPC7450 supports both static and transient memory-access behavior. The **lvxl** and **stvxl** instructions are interpreted as transient data accesses.

7.1.2.3.4 Relationship with the sync/tblsync Instructions

If a **sync** instruction is executed while a **dst***x* is in progress, the following happens for each of the four VTs:

- Any cache line fetch in progress continues until that single cache line refill has completed.
- The VTQ pauses and does not continue to its next line-fetch location.
- When all other necessary conditions are met in the machine, the **sync** instruction is completed.
- The **dst***x* resumes with cache accesses/reloads to the next line-fetch location.

The effect of the **sync** is a short pause in **dst***x* operation. Code sequences that are truly intended to quiet the machine, like those used to enter reduced-power states, must use **dss**/**dssall** followed by a **sync** instruction to kill outstanding transactions initiated by **dst***x* instructions. Refer to Section 7.1.2.3.8, “Differences Between dst/dstt and dstst/dststt Instructions,” for more details on the **dst***x* and **dss**/**dssall** instructions.

Note that a **tlbsync** instruction affects the VTQ identically to a **sync** instruction with the additional effect that an outstanding VTQ-initiated table search operation is canceled when a **tlbsync** is dispatched to the LSU.

7.1.2.3.5 Data Stream Termination

If one of the conditions in Table 7-6 is determined to be true when a given line fetch of a **dst***x* stream is translated, the entire **dst***x* stream is terminated. Note that this can occur in the middle of many line fetches for a **dst***x* stream.

If the condition involves address translation and the **dst***x* stream specifies an access that would cross into another page, the processor does not attempt to continue the **dst***x* stream at those new pages if it had an opportunity to fully translate the access.

Table 7-6. The dst*x* Stream Termination Conditions

Conditions
Successfully reached end of stream
The **dst***x* stream is still speculative with respect to program flow, and the control unit issues a cancel due to a mispredicted branch or exception.
Another **dst***x* instruction to this stream tag is executed, and this new **dst***x* is non-speculative with respect to branch prediction.
A **dss** instruction to this stream tag is completed.
Current line fetch caused a table search operation that did not find a matching entry in the page table.
Current line fetch is translated as cache-inhibited.
Current line fetch is translated as write-through and the stream is a touch-for-store.
Current line fetch is translated to direct-store space (SR[T] = 1).
Current line fetch is to a protected page.
L1 data cache is locked or disabled.
The processor has encountered a condition that causes a machine check exception.

Note that asserting $\overline{\text{SRESET}}$ does not terminate a **dst***x* stream.

7.1.2.3.6 Line Fetch Skipping

When an exception condition occurs, the MPC7450 terminates any **dst***x*-initiated table search operations and pauses the stream engine that initiated the table search. In this situation, the line fetch of the **dst** that caused the table search is effectively dropped and any translation exception that would have terminated the stream had the table search operation completed does not occur. Instead, the engine attempts the next line fetch when the stream resumes. This, in effect, causes a skip of one line fetch in the stream engine.

Also note that the execution of a **tlbsync** instruction cancels any **dst***x*-initiated table search operations in progress, which can cause a line fetch skip.

7.1.2.3.7 Context Awareness and Stream Pausing

Stream accesses can take place only when data translation is enabled (MSR[DR] = 1), and when the processor is in the same privilege state as it was when the **dst***x* instruction was executed.

If the privilege level setting changes or if data translation is disabled, the stream engine suspends generation of new accesses. Any outstanding transactions initiated before the pause (like cache refills and bus activity) finish normally. The stream engine resumes when translation is again enabled and the privilege level again matches the level in place when the **dst***x* instruction for that stream was executed.

7.1.2.3.8 Differences Between dst/dstt and dstst/dststt Instructions

The only difference between touch-for-load (**dst**/**dstt**) and touch-for-store (**dstst**/**dststt)** streams is that touch-for-load streams are subdivided into line fetches that are treated identically to individual **dcbt** fetches, while touch-for-store streams are subdivided into line fetches that are treated identically to individual **dcbtst** fetches.

Note that if a touch-for-store stream instruction is mapped to a write-through page, that stream is terminated. The use of the touch-for-store streams is not recommended when store-miss merging is enabled, which is the default case. See Section 3.4.4.4, “Data Cache Block Store (dcbst),” for further details on store-miss merging.

Although the MPC7450 implements touch-for-store stream instructions, use of these instructions is not recommended because it can degrade performance.

7.1.2.4 dss and dssall Instructions

The Data Stream Stop instruction (**dss)** is never executed speculatively. Instead, **dss** instructions flow into a four-entry **dss** queue (DSSQ) in which one entry is dedicated to each possible tag. If another **dss** is dispatched with a tag that matches a non-completed but valid DSSQ entry, that new **dss** remains in a hold queue and waits for the previous **dss** in the DSSQ to be completed.

If a subsequent **dst***x* is queued in the VTQ, it cancels an older **dss** entry in the DSSQ (for the same tag). When a given DSSQ entry completes, the valid bit for the VTQ entry corresponding to that tag is immediately cleared.

If a **dssall** instruction is executed, the DSSQ queues all four queue entries in order to terminate all four VT streams when the **dssall** instruction is the oldest. The **dssall** opcode differs from **dss** in that bit 6 (the A field) is set and bits 7–10 are ignored.

Note that line fetches in progress for a given **dst***x* stream are not canceled by the **dss** instruction. Only subsequent line fetches are prevented. To ensure that all line fetches from a **dst***x* are completed, a **sync** instruction must be issued after the **dss** instruction.

7.1.2.5 Java Mode, NaNs, Denormalized Numbers, and Zeros

This section describes the MPC7450 floating-point behavior for various special-case data types. The descriptions cover both Java and non-Java modes (see Section 7.1.1.4, “Vector Status and Control Register (VSCR),” for setting Java/non-Java mode), including the following:

- Denormalization for all instructions
- NaNs, denormalized numbers, and zeros for compare, min, and max MPC7450 operations
- Zero and NaN data for round-to-float integral operations

Note the following:

- The MPC7450 defaults to Java mode.
- The MPC7450 handles NaNs the same way regardless of Java or non-Java mode.
- The MPC7450 handles most denormalized numbers in Java mode by taking a trap to exception 0x01600 (AltiVec assist exception) but, for some instructions the MPC7450 can produce the exact result without trapping.

Table 7-7 describes denormalization instructions.

Table 7-7. Denormalization for AltiVec Instructions

Instruction	Input Denormalization Detected		Output Denormalization Detected	
	Java	Non-Java	Java	Non–Java
vaddfp, **vsubfp**, **vmaddfp**, **vnmsubfp**	Trap (unless result is a NaN) [1]	Input treated as correctly signed zero	Trap	Result squashed to correctly signed zero
vrefp	Trap	Denormalized number squashed to zero, returning +/–∞	Trap	Result squashed to zero
vrsqrtefp	Trap	Denormalized number squashed to zero, returning +/–∞	Never produces a denormalized number	Never produces a denormalized number
vlogefp	Trap	Denormalized number squashed to zero, returning –∞	Never produces a denormalized number	Never produces a denormalized number
vexptefp	Result is +1.0	Input squashed to zero, output result is +1.0	Trap	Result squashed to zero
vcfux, vcfsx	Never detects denormalized numbers			
vctsxs, vctuxs	Trap [1]	Output result is 0x0	Never produces a denormalized number	Never produces a denormalized number

[1] May change in the future to produce an IEEE default result in hardware instead of trapping. If the instruction has a denorm operand that would produce a NaN result, the MPC7450 returns the NaN result and does not cause an AltiVec assist exception.

Table 7-8 describes the behavior of the vector floating-point compare, min, and max instructions in non-Java mode.

Table 7-8. Vector Floating-Point Compare, Min, and Max in Non-Java Mode

vA	vB	vminfp	vmaxfp	vcmpgtfp	vcmpgefp	vcmpeqfp	vcmpbfp	
							LE	GE
NaN_A	—	QNaN_A	QNaN_A	False	False	False	0	0
—	NaN_B	QNaN_B	QNaN_B	False	False	False	0	0
+Den_A	–B	–B	+Zero	True	True	False	0	0
–Den_A	–B	–B	–Zero	True	True	False	0	0

Table 7-8. Vector Floating-Point Compare, Min, and Max in Non-Java Mode (continued)

vA	vB	vminfp	vmaxfp	vcmpgtfp	vcmpgefp	vcmpeqfp	vcmpbfp	
							LE	GE
+Den_A	+B	+Zero	+B	False	False	False	1	1
–Den_A	+B	–Zero	+B	False	False	False	1	1
–A	+Den_B	–A	+Zero	False	False	False	1	0
–A	–Den_B	–A	–Zero	False	False	False	1	0
+A	+Den_B	+Zero	+A	True	True	False	0	1
+A	–Den_B	–Zero	+A	True	True	False	0	1
+Den_A/+Zero	+Den_B/+Zero	+Zero	+Zero	False	True	True	1	1
+Den_A/+Zero	–Den_B/–Zero	–Zero	+Zero	False	True	True	1	1
–Den_A/–Zero	+Den_B/+Zero	–Zero	+Zero	False	True	True	1	1
–Den_A/–Zero	–Den_B/–Zero	–Zero	–Zero	False	True	True	1	1

Table 7-9 describes the behavior of the same instructions in Java mode.

Table 7-9. Vector Floating-Point Compare, Min, and Max in Java Mode

vA	vB	vminfp	vmaxfp	vcmpgtfp	vcmpgefp	vcmpeqfp	vcmpbfp	
							LE	GE
NaN_A	—	QNaN_A	QNaN_A	False	False	False	0	0
—	NaN_B	QNaN_B	QNaN_B	False	False	False	0	0
+Den_A	–B	–B	+Den_A	True	True	False	0	0
–Den_A	–B	–B	–Den_A	True	True	False	0	0
+Den_A	+B	+Den_A	+B	False	False	False	1	1
–Den_A	+B	–Den_A	+B	False	False	False	1	1
–A	+Den_B	–A	+Den_B	False	False	False	1	0
–A	–Den_B	–A	–Den_B	False	False	False	1	0
+A	+Den_B	+Den_B	+A	True	True	False	0	1
+A	–Den_B	–Den_B	+A	True	True	False	0	1
+Den_A	±Zero	±Zero	+Den_A	True	True	False	0	1
–Den_A	±Zero	–Den_A	±Zero	False	False	False	1	0
±Zero	+Den_B	±Zero	+Den_B	False	False	False	1	1
±Zero	–Den_B	–Den_B	±Zero	True	True	False	0	0
–Den_A	+Den_B	–Den_A	+Den_B	False	False	False	1	Result depends on input operands
+Den_A	–Den_B	–Den_B	+Den_A	True	True	False	0	

Table 7-9. Vector Floating-Point Compare, Min, and Max in Java Mode (continued)

vA	vB	vminfp	vmaxfp	vcmpgtfp	vcmpgefp	vcmpeqfp	vcmpbfp	
							LE	GE
–Den_A	–Den_B	Result depends on input operands						0
+Den_A	+Den_B							1

Table 7-10 describes the behavior of round-to-integer instructions in non-Java mode.

Table 7-10. Round-to-Integer Instructions in Non-Java Mode

vB Sign	vB exponent	Instruction			
		vrfin	vrfiz	vrfip	vrfim
neg	127 > exp > 24	**v**B	**v**B	**v**B	**v**B
	23 > exp > 0	Round towards nearest	Truncate fraction	Round towards +∞	Round towards –∞
	Exp = –1	Round to nearest	–Zero	–Zero	–1.0
	–2 > exp > –126	–Zero	–Zero	–Zero	–1.0
	Input is denormalized	–Zero	–Zero	–Zero	–Zero
	Input is zero	–Zero	–Zero	–Zero	–Zero
pos	input is zero	+Zero	+Zero	+Zero	+Zero
	Input is denormalized	+Zero	+Zero	+Zero	+Zero
	–126 < exp < –2	+Zero	+Zero	+1.0	+Zero
	exp = –1	Round towards nearest	+Zero	+1.0	+Zero
	0 < exp < 23	Round towards nearest	Truncate fraction	Round towards +∞	Round towards –∞
	24 < exp < 126	**v**B	**v**B	**v**B	**v**B

Table 7-11 describes round-to-integer instructions in Java mode. Note that round-to-integer instructions never produce denormalized numbers.

Table 7-11. Round-to-Integer Instructions in Java Mode

vB Sign	vB Exponent	Instruction			
		vrfin	**vrfiz**	**vrfip**	**vrfim**
neg	127 > exp > 24	**v**B	**v**B	**v**B	**v**B
	23 > exp > 0	Round towards nearest	Truncate fraction	Round towards +∞	Round towards –∞
	Exp = –1	Round to nearest	–Zero	–Zero	–1.0
	–2 > exp > –126	–Zero	–Zero	–Zero	–1.0
	Input is denormalized	Trap	Trap	Trap	Trap
	Input is zero	–Zero	–Zero	–Zero	–Zero
pos	Input is zero	+Zero	+Zero	+Zero	+Zero
	Input is denormalized	Trap	Trap	Trap	Trap
	–126 < exp < –2	+Zero	+Zero	+1.0	+Zero
	Exp = –1	Round towards nearest	+Zero	+1.0	+Zero
	0 < exp < 23	Round to nearest	Truncate fraction	Round To +∞	Round To –∞
	24 < exp < 126	**v**B	**v**B	**v**B	**v**B

The MPC7450 detects underflows and production of denormalized numbers on vector float results before rounding, not after. Future versions of the *AltiVec Technology Programming Environments Manual* may reflect this ordering.

7.1.3 Differences between the MPC7400/MPC7410 and the MPC7450

There exist a few differences in the AltiVec implementation between the MPC7400/MPC7410 and the MPC7450. These differences are within the bounds outlined in the *AltiVec Technology Programming Environments Manual*. The AltiVec technology implementation differences between the processors are described below.

7.1.3.1 Java and Non-Java Mode

The floating-point behavior for special case types is described in Section 7.1.2.5, “Java Mode, NaNs, Denormalized Numbers, and Zeros.” In the MPC7400/MPC7410, the default setting for floating-point behavior is non-Java mode (VSCR[NJ] = 1), and for the MPC7450 it is Java mode (VSCR[NJ] = 0).

7.1.3.2 AltiVec Instructions

The **vrefp** instruction on the MPC7450 returns a different result from the MPC7400/MPC7410 for exact powers of two. The MPC7450 reciprocal estimate for powers of 2 is exact. For example, in the MPC7450 vrefp(+2.0) = +0.50 and in the MPC7400/MPC7410 vrefp(+2.0) = +0.499939.

Also, unlike the MPC7400/MPC7410, for the 1/sqrt(x) estimate instruction **vrsqrtefp**, the MPC7450 does not round the least significant bit of the mantissa.

The **vsr** and **vsl** instructions are executed by the vector permute unit on MPC7450. In the MPC7400/MPC7410, these instructions are executed by the vector simple fixed point unit.

7.1.3.3 AltiVec Instruction Sequencing

The MPC7450 implements the AltiVec execution unit as four subunits: simple, complex, permute, and float. In the MPC7400/MPC7410, the AltiVec execution unit has two subunits: the permute and the vector arithmetic logic unit (ALU), which contains the simple, complex, and float subunits. Because of this difference, the MPC7450 has more Altivec unit parallelism than the MPC7400/MPC7410. The four-entry AltiVec issue queue can issue up to two instructions to two of the four AltiVec subunits (simple, complex, permute, and floating-point). For example, the MPC7450 can issue both a vector simple and a vector complex instruction simultaneously.

The MPC7450 implements the AltiVec execution unit as four subunits: simple (VIU1), complex (VIU2), float (VFPU), and permute (VPU). In the MPC7400/MPC7410, the AltiVec execution unit has two subunits: the permute (VPU) and the vector arithmetic logic unit (VALU), which contains the simple, complex, and float subunits. Because of this difference, the MPC7450 has more AltiVec unit parallelism than the MPC7400/MPC7410. The four-entry AltiVec issue queue (VIQ) can issue up to two instructions to two of the four AltiVec subunits (VIU1, VIU2, VFPU, and VPU). For example, the MPC7450 can issue both a vector simple and a vector complex instruction simultaneously, unlike the MPC7400/MPC7410, which only allows pairing between VPU and one of the other three VALU subunits. Some of the high-level AltiVec implementation-specific differences between the MPC7400/MPC7410 and the MPC7450 are listed in Table 7-12. To determine the specific differences for an AltiVec instruction, a comparison can be made between the execution latencies listed in the “Instruction Timing” chapters for the MPC7400/MPC7410 and the MPC7450.

Table 7-12. AltiVec Implementation-Specific Differences Between the MPC7400/MPC7410 and the MPC7450

Microarchitectural Feature	MPC7400/MPC7410	MPC7450
Available vector execution units	Vector execution units 2-issue to VPU and VALU (VALU has VIU1, VIU2, VFPU subunits) v	2-issue to any 2 vector units (VIU1, VPU, VIU2, VFPU)
VIU1 Execution Unit Timings (Latency-Throughput)	1-1	1-1
VIU2 Execution Unit Timings (Latency-Throughput)	3-1	4-1
VFPU Execution Unit Timings (Latency-Throughput)	4-1	4-1
VPU Execution Unit Timings (Latency-Throughput)	1-1	2-1

Because the MPC7450 contains more execution units, some of the instructions are executed from different execution units. The instructions that execute in different execution units from the MPC7400/MPC7410 to the MPC7450 are listed in Table 7-13.

Table 7-13. MPC7400/MPC7410 and MPC7450 AltiVec Instructions Using a Different Execution Unit

Mnemonic	Where instruction executed in MPC7400/MPC7410	Where instruction executed in MPC7450
mfvscr	VALU (VIU1)	VFPU
mtvscr	VALU (VIU1)	VFPU
vcmpbfp[.]	VALU (VIU1)	VFPU
vcmpeqfp[.]	VALU (VIU1)	VFPU
vcmpgefp[.]	VALU (VIU1)	VFPU
vcmpgtfp[.]	VALU (VIU1)	VFPU
vmaxfp	VALU (VIU1)	VFPU
vminfp	VALU (VIU1)	VFPU
vsl	VALU (VIU1)	VPU
vsr	VALU (VIU1)	VPU

7.2 AltiVec Technology and the Cache Model

The MPC7450 uses a unified LSU to load and store operands into the GPRs, FPRs, and VRs. The MPC7450’s high-bandwidth memory subsystem supports anticipated AltiVec workloads.

The memory subsystem features summarized in the following sections combine to provide high bandwidth while maintaining latencies and cache capacities similar to the MPC7410.

The following list summarizes features of the MPC7450 cache implementation that affect the AltiVec implementation:

- The 32-Kbyte, 8-way set associative L1 data cache is fully non-blocking.
  - The 128-bit interface is designed to support AltiVec load/store operations.
  - It supports both MRU (most recently used) and LRU (least recently used) vector loads.
  - New castout and modified bits support **lvx**/**stvx** LRU operations
- The L2 and L3 cache can be shared when the L1 is exclusive or modified, and the L2 can be shared when the L3 is exclusive or modified or vice-versa. By allowing this, it eliminates the need to allocate or update states in the L2 or L3 when a transient (AltiVec) store is performed to a line shared in the L2 or L3. The true coherency state of the MPC7450 requires all three levels of the cache hierarchy. The LMQ treats the L2 as invalid for stores if it is shared and the L3 is exclusive or modified. The L3 state is ignored for LMQ operations if the L2 is exclusive or modified. Refer to Chapter 3, “L1, L2, and L3 Cache Operation,” for more information. Note that the L3 cache and the L3 cache interface are not supported on the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448.
- Pseudo LRU (PLRU) replacement algorithm for L1 cache
- Random replacement for L2 and L3 cache
- Support for AltiVec LRU instructions. LRU instructions are described in Section 7.1.2.1, “LRU Instructions.”

- Support for AltiVec transient instructions. Transient instructions are described in Section 7.1.2.2, “Transient Instructions and Caches.”

7.3 AltiVec and the Exception Model

Only the four following exceptions can result from execution of an AltiVec instruction:

- An AltiVec unavailable exception occurs when executing any non-stream AltiVec instruction with MSR[VEC] = 0. After this exception occurs, execution resumes at offset 0x00F20 from the base physical address indicated by MSR[IP]. This exception does not occur for data streaming instructions (**dst(t)**, **dstst(t)**, and **dss**). Also note that VRSAVE is not protected by this exception which is consistent with the *AltiVec Programming Environments Manual*. Thus, any access to the VRSAVE register does not cause an exception when MSR[VEC] = 0.
- A DSI exception occurs only if an AltiVec load or store operation encounters a protection violation or a page fault (does not find a valid PTE during a table search operation).
- An AltiVec assist exception may occur if an AltiVec floating-point instruction detects denormalized data as an input or output in Java mode.
- AltiVec loads and stores—The 60x bus protocol does not support a 16-byte bus transaction. Therefore, cache-inhibited AltiVec loads, stores, and write-through stores take an alignment exception. Note that AltiVec loads and stores that are not quad word–aligned also take an alignment exception in MPX bus mode. If either of these conditions is encountered, a re-write of the alignment exception routines in software is required.

7.4 AltiVec and the Memory Management Model

The AltiVec functionality in the MPC7450 affects the MMU model in the following ways:

- A data stream instruction (**dst**(**t**) or **dstst**(**t**)) can cause table search operations to occur after the instruction is retired.
- MMU exception conditions can cause a data stream operation to abort.
- Aborted VTQ-initiated table search operations can cause a line fetch skip.
- Execution of a **tlbsync** instruction can cancel an outstanding table search operation for a VTQ.

Data stream touch instructions may use either of the two translation mechanisms as specified by the PowerPC architecture—segment/page or BAT. For more information, see Chapter 5, “Memory Management.”

7.5 AltiVec Technology and Instruction Timing

AltiVec computational instructions are executed in the four independent pipelined AltiVec execution units. The VPU has a two-stage pipeline, the VIU1 has a one-stage pipeline, and the VIU2 and VFPU have four-stage pipelines. As many as 10 AltiVec instructions can be executing concurrently.

The AltiVec technology defines additional data streaming instructions to help improve throughput. Those instructions are described in Section 7.1.2.3, “Data Stream Touch Instructions.” A complete description of the AltiVec instruction timing is provided in Chapter 6, “Instruction Timing.”

Chapter 8
Signal Descriptions

This chapter describes the MPC7450 microprocessor's external signals. It contains a concise description of individual signals, showing behavior when the signal is an input or an output and when the signal is asserted, negated, or tristated.

The MPC7450 provides a mode switch through the $\overline{\text{BMODE0}}$ signal that, when sampled asserted at $\overline{\text{HRESET}}$ negation, enables the MPX bus protocol operation, and when sampled negated at $\overline{\text{HRESET}}$ negation, enables a limited subset of the 60x bus protocol. The MPX bus is derived from the 60x bus interface and includes a 72-bit data bus (including 8 parity bits) and a 44-bit address bus (including 5 parity bits) along with sufficient control signals to allow for unique system level optimizations.

The 60x bus protocol signals described in the second part of this chapter provide the MPC7450 with compatibility to earlier devices that implement the PowerPC architecture.

Refer to the hardware specifications for detailed electrical and mechanical information for each signal.

8.1 Signal Groupings

The MPC7450 MPX and 60x bus interface protocol signals are grouped as follows:

- Address arbitration—The MPC7450 uses these signals to arbitrate for address bus mastership.
- Address transfer start—These signals indicate that a bus master has begun a transaction on the address bus.
- Address transfer—These signals include the address bus and address parity signals. They are used to transfer the address and to ensure the integrity of the transfer.
- Transfer attribute—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—These signals are used to acknowledge the end of the address phase of the transaction. They also indicate whether a condition exists that requires the address phase to be repeated.
- Data arbitration—The MPC7450 uses these signals to arbitrate for data bus mastership.
- Data transfer—These signals, which consist of the data bus and data parity signals, are used to transfer the data and to ensure the integrity of the transfer.
- Data transfer termination—Data termination signals are required after each data beat in a data transfer. In a single-beat transaction, data termination signals also indicate the end of the tenure. In burst accesses, data termination signals apply to individual beats and indicate the end of the tenure only after the final data beat. Data termination signals also indicate whether a condition exists that requires the data phase to be repeated.

In addition there are many other signals on the MPC7450 that control and affect other aspects of the device, aside from the bus protocol. They are as follows:

- L3 cache address/data—The MPC7450 has separate address and data buses for accessing the L3 cache. Note that the L3 cache is not supported on the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448.
- L3 cache clock/control—These signals provide clocking and control for the L3 cache.
- Interrupts/resets—These signals include the external interrupt signal, checkstop signals, and both soft reset and hard reset signals. They are used to interrupt and, under various conditions, to reset the processor.
- Processor status and control—These signals enable the time-base facility and are used to select the bus mode and control sleep mode.
- Clock control—These signals determine the system clock frequency. They are also used to synchronize multiprocessor systems.
- Test interface—The JTAG (IEEE 1149.1a-1993) interface and the common on-chip processor (COP) unit provide a serial interface to the system for performing board-level boundary-scan interconnect tests.
- Voltage selection—These signal control the electrical characteristics of the I/O circuitry of the device as appropriate to support various signalling levels.

8.1.1 Signal Summary

Table 8-1 lists in alphabetical order all the MPC7450 signals and provides a cross-reference to the section of this chapter that contains the detailed description for each.The table also shows which signals provide multiple functions and are multiplexed on the MPC7450.

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[0:4] (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.

Table 8-1. MPC7450 Signal Reference

Signal	Signal Name	Interface	No. of Signals	I/O	Section/Page
A[0:35]	Address	60x, MPX	36	I/O	8.2.6.1/8-13
$\overline{\text{AACK}}$	Address acknowledge	60x, MPX	1	I	8.3.5.1/8-36
AP[0:4]	Address parity	60x, MPX	5	I/O	8.3.3.3/8-33
$\overline{\text{ARTRY}}$	Address retry	60x, MPX	1	I/O	8.2.8.2/8-19
$\overline{\text{BG}}$	Bus grant	60x, MPX	1	I	8.2.5.2/8-12
$\overline{\text{BMODE0}}$	Bus mode select 0	60x, MPX	1	I	8.4.4.8/8-51
$\overline{\text{BMODE1}}$	Bus mode select 1	60x, MPX	1	I	8.4.4.8/8-51
$\overline{\text{BR}}$	Bus request	60x, MPX	1	O	8.2.5.1/8-12

Table 8-1. MPC7450 Signal Reference (continued)

Signal	Signal Name	Interface	No. of Signals	I/O	Section/Page
BVSEL [1]	Bus voltage select	60x, MPX	1	I	8.4.4.4/8-49
BVSEL[0:1] [2]	Bus voltage select	60x, MPX	2	I	8.4.4.5/8-50
$\overline{\text{CI}}$	Cache-inhibited	60x, MPX	1	O	8.3.4.7/8-36
$\overline{\text{CKSTP\_IN}}$	Checkstop in	60x, MPX	1	I	8.4.3.5/8-47
$\overline{\text{CKSTP\_OUT}}$	Checkstop out	60x, MPX	1	O	8.4.3.6/8-48
CLK_OUT	Clock out	60x, MPX	1	O	8.4.5.5/8-56
$\overline{\text{DBG}}$	Data bus grant	60x, MPX	1	I	8.3.6.1/8-38
D[0:63]	Data bus	60x, MPX	64	I/O	8.3.7.1/8-39
$\overline{\text{DFS2}}$ [2]	DFS divide-by-two	60x, MPX	1	I/O	8.4.4.6/8-50
$\overline{\text{DFS4}}$ [2]	DFS divide-by-four	60x, MPX	1	I/O	8.4.4.6/8-50
DP[0:7]	Data parity	60x, MPX	8	I/O	8.3.7.2/8-39
$\overline{\text{DRDY}}$	Data ready	MPX	1	O	8.2.3.3/8-7
DTI[0:3]	Data transaction index	MPX	4	I	8.2.9.2/8-23
EXT_QUAL	PLL bypass clock	60x, MPX	1	I	8.4.5.4/8-55
$\overline{\text{GBL}}$	Global	60x, MPX	1	I/O	8.3.4.5/8-35
$\overline{\text{HIT}}$	Snoop hit	MPX	1	O	8.2.8.4/8-21
$\overline{\text{HRESET}}$	Hard reset	60x, MPX	1	I	8.4.3.4/8-47
$\overline{\text{INT}}$	Interrupt request	60x, MPX	1	I	8.4.3.1/8-46
$\overline{\text{LVRAM}}$ [2]	Low voltage RAM	60x, MPX	1	I/O	8.4.4.7/8-50
L3_ADDR[17:0][3] (For MPC7457, L3_ADDR[18:0]) [4]	L3 address	L3	18	O	8.4.1.1/8-41
L3_CLK[0:1][3]	L3 clock		2	O	8.4.2.1/8-44
$\overline{\text{L3\_CNTL0}}$[3]	L3 load		1	O	8.4.2.3/8-45
$\overline{\text{L3\_CNTL1}}$[3]	L3 write		1	O	8.4.2.3/8-45
L3_DATA[0:63][3]	L3 data		64	I/O	8.4.1.2/8-42
L3_DP[0:7][3]	L3 data parity		8	I/O	8.4.1.3/8-43
L3_ECHO_CLK[0:3][3]	L3 synchronizing clock		4	I/O[5]	8.4.2.2/8-44
L3_VSEL[3]	L3 voltage select		1	I	8.4.2.4/8-45
$\overline{\text{MCP}}$	Machine check	60x, MPX	1	I	8.4.3.3/8-46
PLL_CFG[0:5][6]	PLL configuration	60x, MPX	5	I	8.4.5.2/8-55, 8.4.5.3/8-55
$\overline{\text{PMON\_IN}}$	Performance monitor in	60x, MPX	1	I	8.4.4.9/8-53
$\overline{\text{PMON\_OUT}}$	Performance monitor out	60x, MPX	1	O	8.4.4.10/8-54

Table 8-1. MPC7450 Signal Reference (continued)

Signal	Signal Name	Interface	No. of Signals	I/O	Section/Page
$\overline{\text{QACK}}$	Quiescent acknowledge	60x, MPX	1	I	8.4.4.3/8-49
$\overline{\text{QREQ}}$	Quiescent request	60x, MPX	1	O	8.4.4.2/8-49
$\overline{\text{SRESET}}$	Soft reset	60x, MPX	1	I	8.4.3.4/8-47
$\overline{\text{SHD}}$[0]	Shared 0	60x, MPX	1	I/O	8.3.5.3/8-37
$\overline{\text{SHD}}$[1]	Shared 1	MPX	1	I/O	8.2.8.3/8-20
$\overline{\text{SMI}}$	System management interrupt	60x, MPX	1	I	8.4.3.2/8-46
SYSCLK	System clock	60x, MPX	1	I	8.4.5.1/8-54
$\overline{\text{TA}}$	Transfer acknowledge	60x, MPX	1	I	8.2.11.1/8-26
TBEN	Time base enable	60x, MPX	1	I	8.4.4.1/8-48
$\overline{\text{TBST}}$	Transfer burst	60x, MPX	1	O	8.3.4.3/8-35
TCK	Scan clock	JTAG	1	I	8.4.6.1/8-57
TDI	Serial scan input		1	I	8.4.6.2/8-57
TDO	Serial scan output		1	O	8.4.6.3/8-57
$\overline{\text{TEA}}$	Transfer error acknowledge	60x, MPX	1	I	8.2.11.2/8-27
TMS	Test mode select	JTAG	1	I	8.4.6.4/8-57
$\overline{\text{TS}}$	Transfer start	60x, MPX	1	I/O	8.3.4.1/8-33
$\overline{\text{TRST}}$	Test reset	JTAG	1	I	8.4.6.5/8-57
TSIZ[0:2]	Transfer size	60x, MPX	3	O	8.3.4.4/8-35
TT[0:4]	Transfer type	60x, MPX	5	I/O	8.3.4.2/8-34
$\overline{\text{WT}}$	Write-through	60x, MPX	1	O	8.3.4.6/8-36

[1] The BVSEL pin definition has changed on the MPC7448. Refer to the hardware specifications for more information.

[2] MPC7448-specific signal.

[3] Note that L3 cache interface is not supported on the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448.

[4] MPC7457 supports an additional signal, L3_ADDR[18].

[5] These are either input or output signals depending on the type of SRAM used.

[6] The MPC7448 has a sixth PLL configuration signal, PLL_CFG[5].

8.1.2 Output Signal States During Reset

The assertion of $\overline{\text{HRESET}}$ causes all bi-directional signals to be in the input state. Table 8-2 shows the state of MPC7450 output signals during $\overline{\text{HRESET}}$ assertion.

Table 8-2. Output Signal States During System Reset

Signal Group	Signals	State During System Reset
Address arbitration	$\overline{\text{BR}}$	High impedance
Address bus	A[0:35] AP[0:4]	High impedance
Address transfer Attributes	$\overline{\text{TS}}$ TT[0:4] $\overline{\text{TBST}}$ TSIZ[0:2] $\overline{\text{GBL}}$ $\overline{\text{WT}}$ $\overline{\text{CI}}$	High impedance
Address termination	$\overline{\text{HIT}}$ $\overline{\text{ARTRY}}$ $\overline{\text{SHD0}}$ $\overline{\text{SHD1}}$	High impedance
Data	D[0:63]	High impedance
Data arbitration	$\overline{\text{DRDY}}$	High impedance
L3 cache address/data[1]	L3_ADDR[17:0] [2] L3_DATA[0:63]	High impedance
L3 cache clock/control [3]	$\overline{\text{L3\_CNTL}}$[0]	$\overline{\text{L3\_CNTL}}$[0] = 1
	$\overline{\text{L3\_CNTL}}$[1]	$\overline{\text{L3\_CNTL}}$[1] = 0
	L3_CLK[0:1]	L3_CLK[0:1] = 0
	L3_ECHO_CLK[0:3]	High impedance
Interrupts/resets	$\overline{\text{CKSTP\_OUT}}$	High impedance
Processor status/control	$\overline{\text{QREQ}}$	High impedance
Clock control	CLK_OUT	High impedance

[1] Note that L3 cache interface is not supported on the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448.

[2] MPC7457 supports an extra signal (L3_ADDR[18:0].

8.2 MPX Bus Signal Configuration

The MPC7450 has an advanced bus interface that is derived from the 60x bus. This interface, the MPX bus, includes several additional features that provide higher memory bandwidth than the 60x bus and more efficient use of the system bus in a multiprocessing environment.

The value of the $\overline{\text{BMODE0}}$ signal during $\overline{\text{HRESET}}$ negation determines whether the MPC7450 operates with the 60x bus or the MPX bus. The inverse of this value is stored in bit 16 of the BMODE field in

MSSCR0. The state of MSSCR0[BMODE] is active high, meaning that if BMODE0 is detected as asserted at the negation of HRESET, MSSCR0[16] = 1 and MPX bus mode is selected; if negated at the negation of HRESET, MSSCR0[16] = 0 and 60x bus mode is selected.

Note that the BMODE1 signal must be held negated during HRESET negation or no bus mode will be selected.

8.2.1 MPX/60x Bus Protocol Signal Compatibility

The MPX bus mode protocol defines several signals not present in the 60x bus protocol. Additionally, there are 60x signals not supported by the MPC7450. These signal differences are summarized in Table 8-3. Note that a few 60x signals have expanded or modified functionality in the MPX bus mode.

Table 8-3. Signal Compatibility Summary

MPX Bus Mode Signals	60x Bus Signals Not in MPC7450
Hit HIT Data ready DRDY Shared SHD1	Address bus busy ABB Data bus busy DBB Data bus write only DBWO Data retry DRTRY Extended transfer protocol XATS Transfer code TC[0:1] Cache set element CSE[0:1] Address parity error APE Data parity error DPE

The two types of signals in Table 8-3 (shown in the column headings) are described in Section 8.2, “MPX Bus Signal Configuration,” and Section 8.3, “60x Bus Signal Configuration.”

8.2.2 MPX Bus Mode Signals

The MPX bus mode’s support for data intervention and full data streaming for burst reads and writes is realized through the addition of two new signals—HIT and DRDY. See Section 9.4, “MPX Bus Data Tenure,” for a complete description of these functions.

The HIT signal is a point-to-point signal output from the processor or local bus slave to the system arbiter. This signal indicates a valid snoop response in the address retry (ARTRY) window (the cycle after an address acknowledge (AACK) that indicates that the MPC7450 will supply intervention data). That is, the MPC7450 has found the data that has been requested by another master’s bus transaction in its L1, L2, or L3 cache. Instead of asserting ARTRY and flushing the data to memory, the MPC7450 may assert HIT to indicate that it can supply the data directly to the other master. This external intervention functionality is disabled by MSSCR0[EIDIS].

The DRDY signal is also used by the MPX bus protocol to implement data intervention in the case of a cache hit. See Section 8.2.9.3, “Data Ready (DRDY)—Output.”

The SHD1 signal operates in conjunction with the SHD0 signal to indicate that a cached item is shared. See Section 8.2.8.3, “Shared (SHD0, SHD1) Signals.”

8.2.3 60x Bus Signals Not in the MPC7450

Several signals defined in the 60x bus protocol are not implemented in the MPC7450; however, new signals provide similar functionality for compatibility reasons.

8.2.3.1 Address Bus Busy and Data Bus Busy ($\overline{\text{ABB}}$ and $\overline{\text{DBB}}$)

The MPC7450 does not use or provide the $\overline{\text{ABB}}$ or $\overline{\text{DBB}}$ signals. The MPC7450 tracks its own outstanding transactions and relies on the system arbiter to provide grants for the address and data buses only when the bus is available and the grant may be accepted. Bus arbiters must not rely upon an $\overline{\text{ABB}}$ or $\overline{\text{DBB}}$ signal to properly arbitrate for the address or data bus.

8.2.3.2 Data Bus Write Only ($\overline{\text{DBWO}}$)

The $\overline{\text{DBWO}}$ signal is not implemented on the MPC7450. This functionality is replaced for MPX mode with the DTI[0:3] signals, which implement more extensive data reordering functionality. DTI support is not functional in 60x mode, leaving this mode with no data reordering capability. See Section 8.2.9.2, “Data Transaction Index (DTI[0:3])—Input.”

8.2.3.3 Data Retry ($\overline{\text{DRTRY}}$)

The data retry input signal is not implemented on the MPC7450. Only the no-$\overline{\text{DRTRY}}$ mode defined in the 60x bus protocol is supported.

8.2.3.4 Extended Transfer Protocol ($\overline{\text{XATS}}$)

The extended transfer protocol signal, used for accesses to direct-store segments, is not supported by the MPC7450 processor interface. The MPC7450 does not generate extended transfer protocol ($\overline{\text{XATS}}$) transactions.

8.2.3.5 Transfer Code (TC[0:1])

The transfer code signals are not implemented on the MPC7450. The information provided by these pins for code versus data during read operations is provided on the $\overline{\text{WT}}$ signal.

8.2.3.6 Cache Set Element (CSE[0:1])

These signals are not implemented as the MPC7450 does not support snoop-filtering devices.

8.2.3.7 Address Parity Error and Data Parity Error ($\overline{\text{APE}}$, $\overline{\text{DPE}}$)

The address parity and data parity error signals are not implemented in the MPC7450.

8.2.4 MPX Bus Mode Functional Groupings

Figure 8-1 illustrates the signal configuration in MPX bus mode for the MPC7450, MPC7451, MPC7441, MPC7455, and MPC7445, showing how the signals are grouped. A pinout showing pin numbers is included in the hardware specifications. Note that in Figure 8-1 through Figure 8-4 the left side of each figure depicts the signals that implement the MPX bus protocol and the right side of each figure shows the remaining signals on the MPC7450 (not part of the bus protocol).

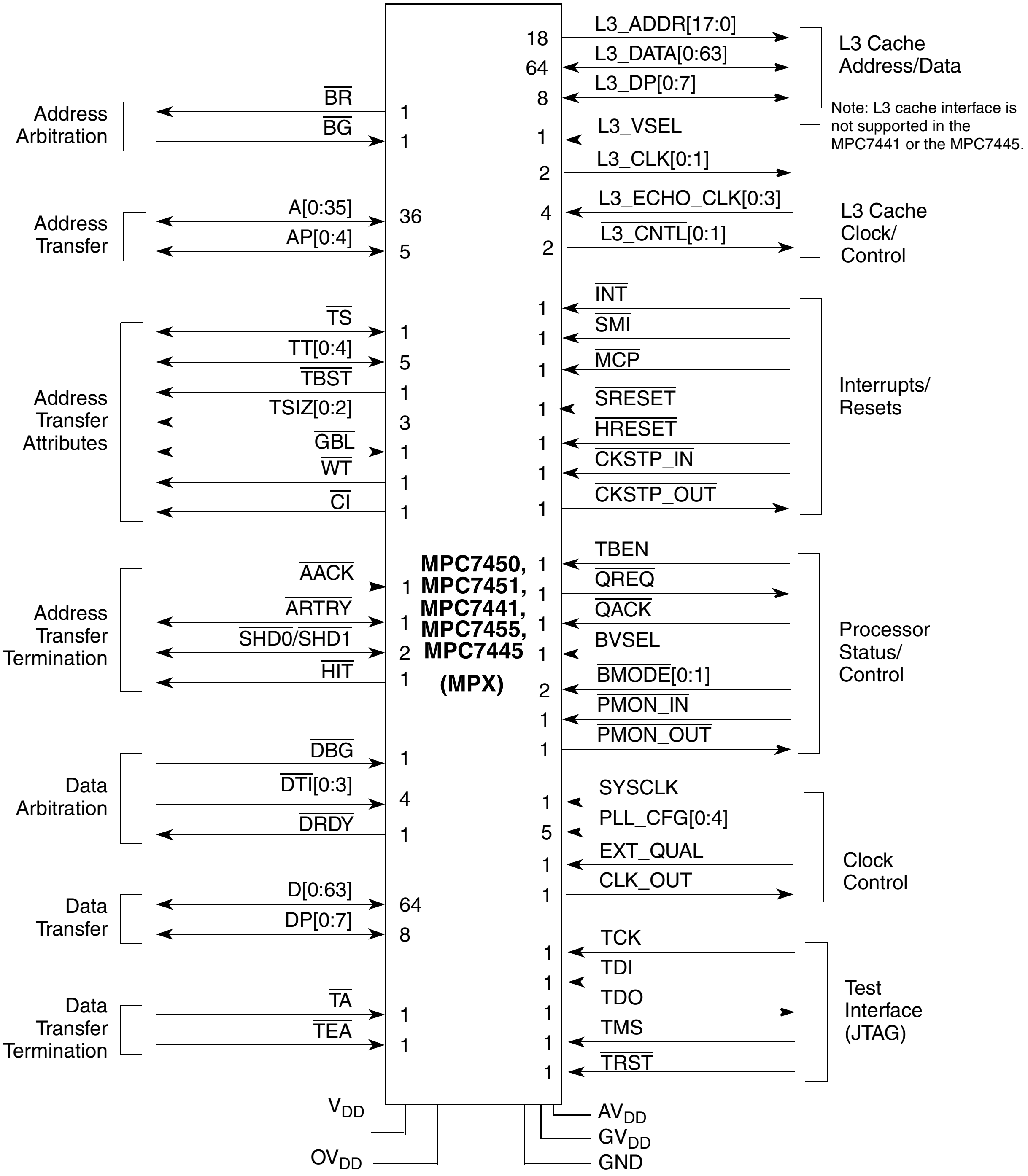
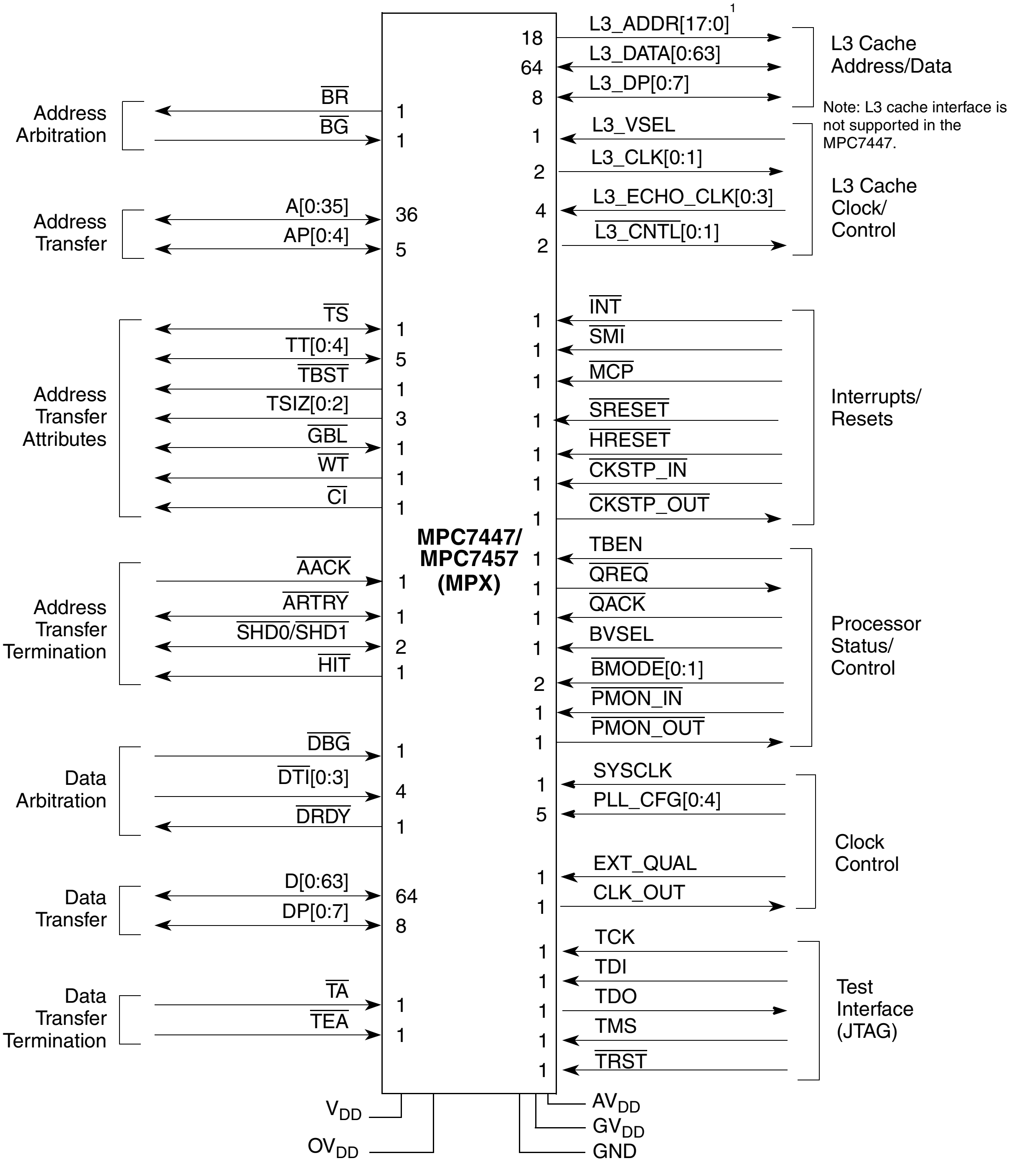


Figure 8-1. MPX Bus Signal Groups in the MPC7450, MPC7451, MPC7441, MPC7455, and MPC7445

Figure 8-2 illustrates the signal configuration in MPX bus mode for the MPC7447 and the MPC7457.



[1] For the MPC7457, there are 19 L3_ADDR signals, (L3_ADDR[0:18]).

Figure 8-2. MPX Bus Signal Groups in the MPC7447 and MPC7457

Figure 8-3 illustrates the signal configuration in MPX bus mode for the MPC7457 and the MPC7447.

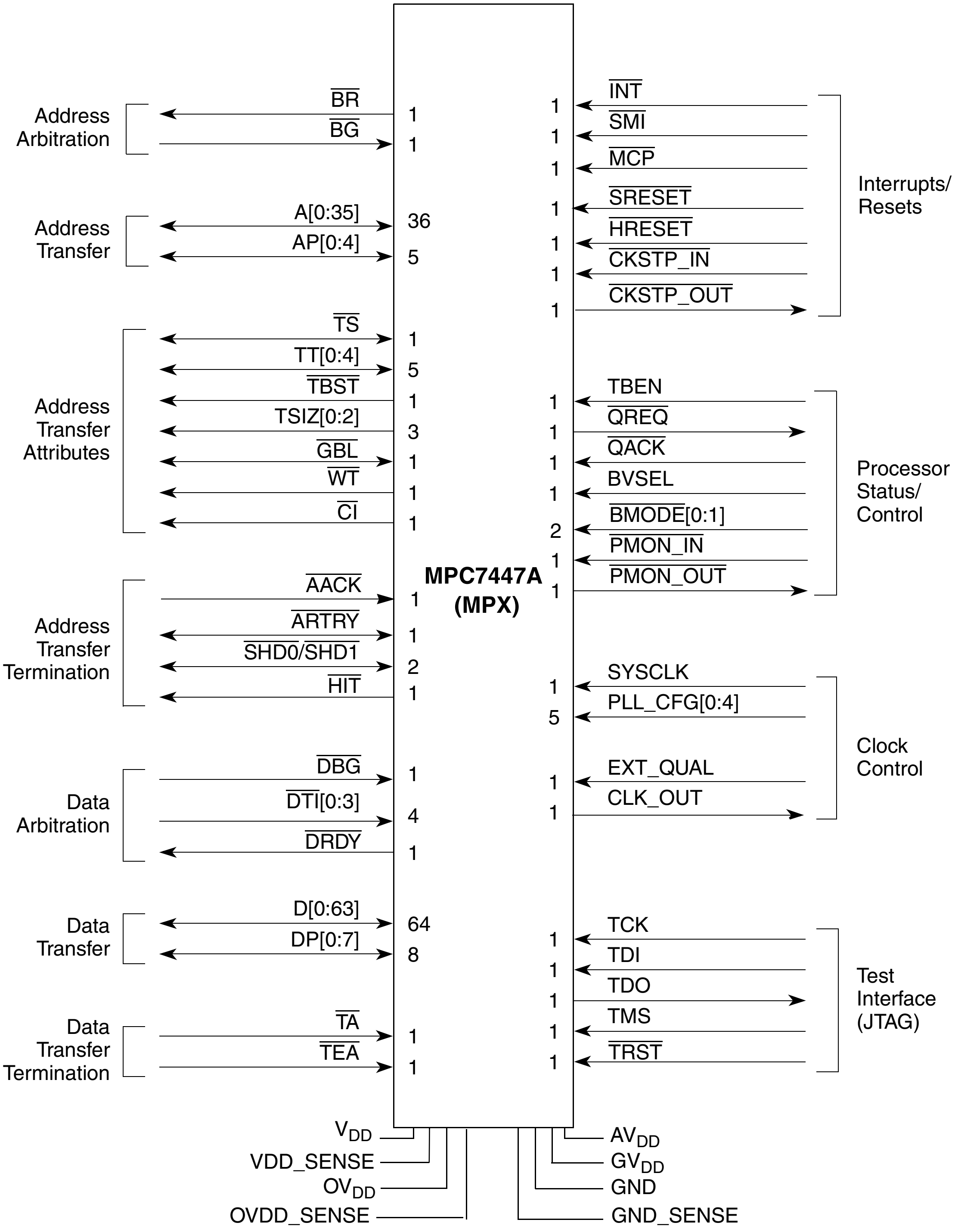


Figure 8-3. MPX Bus Signal Groups in MPC7447A

Figure 8-4 illustrates the MPC7448’s signal configuration in MPX bus mode

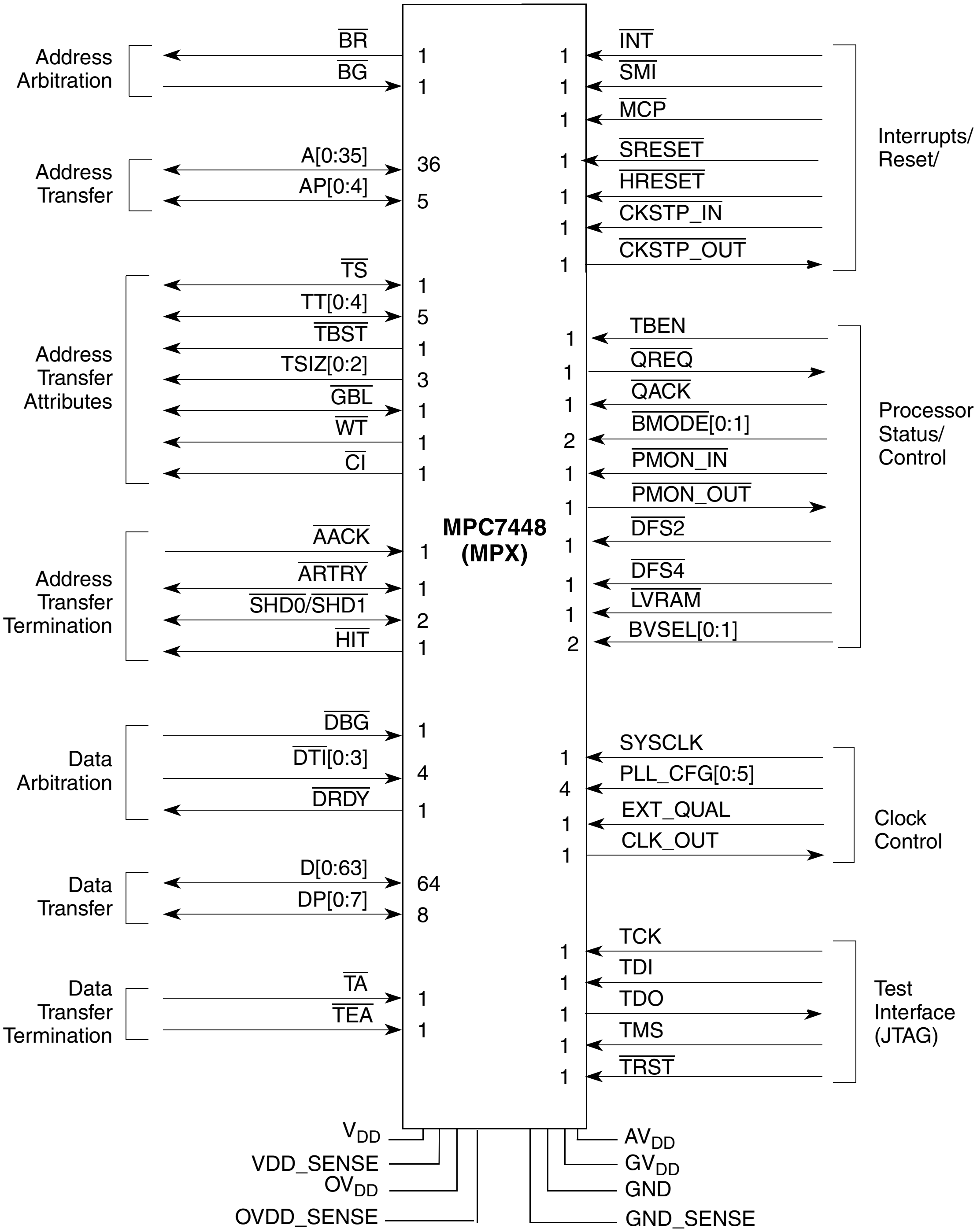


Figure 8-4. MPX Bus Signal Groups in MPC7448

Note that the following sections summarize MPX signal functions. Chapter 9, “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. The 60x bus protocol signals start in Section 8.3, “60x Bus Signal Configuration.”

8.2.5 Address Bus Arbitration Signals

The address arbitration signals are the input and output signals the MPC7450 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 9.3.1, “MPX Bus Address Bus Arbitration.”

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

Following are the state meaning and timing comments for the $\overline{\text{BR}}$ output signal on the MPC7450 in MPX bus mode.

State Meaning

Asserted—Indicates that the MPC7450 is requesting mastership of the address bus, that its pipeline depth allows it to start the transaction, and that it is waiting for a qualified $\overline{\text{BG}}$ to begin the address tenure. Note that $\overline{\text{BR}}$ may be asserted for one or more cycles, and then negated due to an internal cancellation of the bus request. See Section 9.3.1, “MPX Bus Address Bus Arbitration,” for more information.

Negated—Indicates that the MPC7450 is not requesting the address bus. The MPC7450 may have no bus operation pending, the address bus may be parked, or the $\overline{\text{ARTRY}}$ input may have been asserted on the previous bus clock cycle.

Timing Comments

Assertion—Occurs when the MPC7450 is not parked, it does not have a qualified bus grant, and a bus transaction is needed

Negation—Occurs the cycle after a qualified $\overline{\text{ARTRY}}$ on the bus unless the MPC7450 asserted the $\overline{\text{ARTRY}}$ and is required to perform a snoop copyback; may also occur on any cycle if the request is internally cancelled before a qualified $\overline{\text{BG}}$.

High Impedance—Occurs during a hard reset or checkstop condition.

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

Following are the state meaning and timing comments for the $\overline{\text{BG}}$ output signal on the MPC7450 in MPX bus mode.

State Meaning

Asserted—Indicates that the MPC7450 may, with the proper qualification, begin a bus transaction. A qualified bus grant is determined from the bus state as follows:

QBG = $\overline{\text{BG}}$ • ¬$\overline{\text{ARTRY}}$ • ¬$\overline{\text{TS}}$ • ¬(latched state variables)

Note that the assertion of $\overline{\text{BR}}$ is not required for a qualified bus grant. Because $\overline{\text{AACK}}$ is not in the qualified bus grant equation, the bus arbiter must negate $\overline{\text{BG}}$ in every cycle the arbiter is delaying $\overline{\text{AACK}}$.

Negated—Indicates that the MPC7450 is not granted next address bus ownership.

Timing Comments Assertion—May occur on any cycle. Because $\overline{\text{AACK}}$ is not in the qualified bus grant equation, the bus arbiter must negate $\overline{\text{BG}}$ in every cycle the arbiter is delaying $\overline{\text{AACK}}$ to prevent a qualified bus grant.

Negation—May occur whenever the MPC7450 must be prevented from starting a bus transaction. The MPC7450 may still assume address bus ownership on the cycle $\overline{\text{BG}}$ is negated if $\overline{\text{BG}}$ was asserted in the previous cycle with the other bus grant qualifications. Negation must occur in every cycle the arbiter delays $\overline{\text{AACK}}$.

8.2.6 Address Bus and Parity in MPX Bus Mode

The following sections describe the address bus and parity signals used to transmit the address and to generate and monitor parity for the address transfer. The address bus driven mode is enabled with the assertion of $\overline{\text{BMODE0}}$ *after* $\overline{\text{HRESET}}$ negation (assertion of $\overline{\text{BMODE0}}$ during $\overline{\text{HRESET}}$ negation sets the MPC7450’s bus mode). Note that this selection is reflected in the read-only ABD bit in MSSCR0. See Section 2.2.5.3, “Memory Subsystem Control Register (MSSCR0).”

8.2.6.1 Address Bus (A[0:35])

The address bus (A[0:35]) consists of 36 signals that are both input and output signals. A[0:3] are always driven as a zero when extended addressing is disabled through HID0[XAEN] (extended addressing is not enabled, HID0[XAEN] = 0). See Section 9.3.2, “MPX Bus Address Transfer,” for more information on extended addressing.

8.2.6.1.1 Address Bus (A[0:35])—Output

Following are the state meaning and timing comments for the address bus A[0:35] as output signals on the MPC7450 in MPX bus mode.

State Meaning Asserted/Negated—Represents the physical address of the data to be transferred. On burst transfers, the address bus presents the double-word-aligned address 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 9.3.2, “MPX Bus Address Transfer.”

Timing Comments Assertion/Negation—Occurs on the bus clock cycle that $\overline{\text{TS}}$ is asserted; remains asserted for the duration of the address tenure.

High Impedance—Occurs one bus clock cycle following the assertion of $\overline{\text{AACK}}$ unless address bus streaming is occurring and the MPC7450 qualified a $\overline{\text{BG}}$ on the previous cycle.

Note that if MSSCR0[ABD] is set the address bus is always driven on the bus clock cycle after a qualified bus grant is asserted to the processor, regardless of whether the MPC7450 has a queued transaction.

8.2.6.1.2 Address Bus (A[0:35])—Input

Following are the state meaning and timing comments for the address bus A[0:35] as input signals on the MPC7450 in MPX bus mode.

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

Timing Comments Assertion/Negation—Must be valid on the same bus clock cycle as the assertion of $\overline{\text{TS}}$; it is sampled by MPC7450 only on this cycle.

Note that unused address signals cannot be left floating. If any address signals are unused by the system, they must be driven by the system during the address tenure of the transaction to be snooped, or tied low with a pull-down resistor.

High Impedance—Occurs on the bus clock cycle after the assertion of $\overline{\text{AACK}}$ unless address bus streaming is occurring and the MPC7450 qualified a $\overline{\text{BG}}$ on the previous cycle.

8.2.6.2 Address Bus Parity (AP[0:4])

The address bus parity (AP[0:4]) signals, both input and output, reflect one bit of odd-byte parity for each of the 4 bytes and the one extra nibble of address when a valid address is on the bus.

8.2.6.2.1 Address Bus Parity (AP[0:4])—Output

Following are the state meaning and timing comments for AP[0:4] as output signals on the MPC7450.

State Meaning Asserted/Negated—Represents odd parity for each of the 4 bytes and the one extra nibble of the physical address for a transaction. Odd parity means that an odd number of bits, including the parity bit, are driven high. Address parity is generated by the MPC7450 when it is the address bus master.

Table 8-4 shows the address parity signal assignments. For more information, see Section 9.3.2.3, "Address Bus Parity."

Table 8-4. Address Parity Bit Assignments

Address Parity Bit	Corresponding Address Bus Signals
AP0	A[0:3]
AP1	A[4:11]
AP2	A[12:19]
AP3	A[20:27]
AP4	A[28:35]

Timing Comments Assertion/Negation—The same as A[0:35].

High Impedance—The same as A[0:35].

8.2.6.2.2 Address Bus Parity (AP[0:4])—Input

Following are the state meaning and timing comments for AP[0:4] as input signals on the MPC7450.

State Meaning	Asserted/Negated—Represents odd parity for each of the 4 bytes and one nibble of the physical address for snooping operations (if enabled through HID1). Detected even parity causes the processor to take a machine check exception or enter the checkstop state if address parity checking is enabled (HID1[EBA] = 1); see Section 2.2.5.2, “Hardware Implementation-Dependent Register 1 (HID1).”
Timing Comments	Assertion/Negation—The same as A[0:35]. Note that unused address parity signals cannot be left floating. If the address bits corresponding to an address parity bit are always driven to a 0, then the parity bit must be driven to a 1. If the address bits are connected to pull-down resistors, then the corresponding address parity bit should be attached to a pull-up resistor.

8.2.7 Address Transfer Attribute Signals in MPX Bus Mode

The transfer attribute signals are a set of signals that characterize the following:

- Size of the transfer
- Whether it is a read or write operation
- Whether it is a burst or single-beat transfer

For a detailed description of how these signals interact, see Section 9.3.2, “MPX Bus Address Transfer.”

8.2.7.1 Transfer Start ($\overline{\text{TS}}$)

The address transfer start ($\overline{\text{TS}}$) signal indicates that an address bus transfer has begun and is both an input and an output signal on the MPC7450.

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

Following are the state meaning and timing comments for $\overline{\text{TS}}$ as an output signal.

State Meaning	Asserted—Indicates that the MPC7450 has begun a 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—May occur on any cycle following a qualified $\overline{\text{BG}}$. Remains asserted for one clock only. Negation—Occurs one bus clock cycle after $\overline{\text{TS}}$ is asserted and continues negated through the cycle of $\overline{\text{AACK}}$. High Impedance—Occurs the cycle after $\overline{\text{AACK}}$, unless address bus streaming is occurring and the MPC7450 qualified a $\overline{\text{BG}}$ on the previous cycle.

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

Following are the MPC7450’s state meaning and timing comments for $\overline{\text{TS}}$ as an 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 Section 8.2.7.5, “Global (GBL).” Negated—Indicates that no bus transaction is occurring.
Timing Comments	Assertion—Must be asserted for one cycle only. Can occur on any bus clock cycle following a qualified $\overline{\text{BG}}$ that is accepted by the processor. Negation—Must occur one bus clock cycle after $\overline{\text{TS}}$ is asserted.

8.2.7.2 Transfer Type (TT[0:4])

The transfer type (TT[0:4]) signals consist of five input/output signals on the MPC7450. For a complete description of the TT[0:4] signals for MPX bus mode, see Section 9.3.2.4, “Address Transfer Attributes.”

8.2.7.2.1 Transfer Type (TT[0:4])—Output

Following are the state meaning and timing comments for TT[0:4] as output signals on the MPC7450 in MPX bus mode. Note that there is a new transfer type called read claim (RCLAIM; TT[0:4] = 0b01111) defined for MPX bus mode that is used for accesses generated by touch-for-store instructions.

State Meaning	Asserted/Negated—Indicates the type of transfer in progress.
Timing Comments	Assertion/Negation—The same as A[0:35].

8.2.7.2.2 Transfer Type (TT[0:4])—Input

Following are the state meaning and timing comments for TT[0:4] as input signals on the MPC7450 in MPX bus mode.

State Meaning	Asserted/Negated—Indicates the type of transfer in progress.
Timing Comments	Assertion/Negation—The same as A[0:35].

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

The transfer burst ($\overline{\text{TBST}}$) signal is an output signal on the MPC7450 that indicates a burst transfer is in progress. Following are the state meaning and timing comments for the $\overline{\text{TBST}}$ output signal on the MPC7450 in MPX bus mode.

State Meaning	Asserted—Indicates that a burst transfer is in progress. For transactions initiated by external control instructions (**eciwx** and **ecowx**), $\overline{\text{TBST}}$ forms part of the 4-bit Resource ID field on the bus as follows: $\overline{\text{TBST}}$ || TSIZ[0:2] ← EAR[28–31] Negated—Indicates that a burst transfer is not in progress.
Timing Comments	Assertion/Negation—The same as A[0:35]. High Impedance—The same as A[0:35].

8.2.7.4 Transfer Size (TSIZ[0:2])—Output

The TSIZ[0:2] signals specify the data transfer size of a transaction. For memory accesses, these signals along with $\overline{\mathrm{TBST}}$, indicate the data transfer size for the current bus operation. See Section 9.3.2.4.1, "Transfer Type (TT[0:4]) Signals." Also, Section 9.3.2.6, "Effect of Alignment in Data Transfers," shows how the transfer size signals are used with the address signals for aligned and misaligned transfers. Note that the MPC7450 does not generate all possible TSIZ[0:2] encodings.

Following are the state meaning and timing comments for the transfer size (TSIZ[0:2]) output signals on the MPC7450 in MPX bus mode.

State Meaning	Asserted—Indicates the size of the transfer in progress. Asserted/Negated—For transactions initiated by external control instructions (**eciwx** and **ecowx**), the TSIZ[0:2] signals form part of the 4-bit resource ID field (they are used to output bits 29–31 of the external access register (EAR)) on the bus as follows: $\overline{\mathrm{TBST}}$ || TSIZ[0:2] ← EAR[28–31]
Timing Comments	Assertion/Negation—The same as A[0:35]. High Impedance—The same as A[0:35].

8.2.7.5 Global ($\overline{\mathrm{GBL}}$)

The global ($\overline{\mathrm{GBL}}$) signal is an input and output signal on the MPC7450.

8.2.7.5.1 Global ($\overline{\mathrm{GBL}}$)—Output

Following are the state meaning and timing comments for $\overline{\mathrm{GBL}}$ as an output signal in MPX bus mode.

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 during certain data cache, memory synchronization, TLB management, and external control operations as described in Table 3-27). Thus, this transaction must be snooped. Negated—Indicates that a transaction is not global and does not need to be snooped by other masters.
Timing Comments	Assertion/Negation—The same as A[0:35].

8.2.7.5.2 Global ($\overline{\mathrm{GBL}}$)—Input

Following are the state meaning and timing comments for $\overline{\mathrm{GBL}}$ as an input signal in MPX bus mode.

State Meaning	Asserted—Indicates that a transaction must be snooped by the MPC7450. Negated—Indicates that a transaction must not be snooped by the MPC7450.
Timing Comments	Assertion/Negation—The same as A[0:35].

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

The $\overline{\text{WT}}$ signal is an output signal on the MPC7450. Following are the state meaning and timing comments for the $\overline{\text{WT}}$ signal.

State Meaning	Asserted—Indicates that a single-beat write transaction is write-through, reflecting the value of the W bit for the block or page that contains the address of the current transaction (except during certain data cache, memory synchronization, TLB management, and external control operations as described in Table 3-27). Negated—Indicates that a write transaction is not write-through. The MPC7450 negates $\overline{\text{WT}}$ for instruction fetches. Note that this is different from previous processors.
Timing Comments	Assertion/Negation—The same as A[0:35].

8.2.7.7 Cache Inhibit ($\overline{\text{CI}}$)—Output

The $\overline{\text{CI}}$ signal is an output signal on the MPC7450. Following are the state meaning and timing comments for the $\overline{\text{CI}}$ signal as an output.

State Meaning	Asserted—Indicates that a single-beat transfer is not cached, reflecting the setting of the I bit for the block or page that contains the address of the current transaction (except during certain data cache, memory synchronization, TLB management, and external control operations as described in Table 3-27). $\overline{\text{CI}}$ is also asserted for reads and writes if the L1 cache is disabled. Negated—Indicates that a burst transfer allocates an MPC7450 data cache block.
Timing Comments	Assertion/Negation—The same as A[0:35].

8.2.8 MPX 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 must be terminated. For detailed information about how these signals interact, see Section 9.3.3, “MPX Bus Address Tenure Termination.”

8.2.8.1 Address Acknowledge ($\overline{\text{AACK}}$)—Input

The address acknowledge ($\overline{\text{AACK}}$) signal is an input signal on the MPC7450. Following are the state meaning and timing comments for the $\overline{\text{AACK}}$ signal in MPX bus mode.

State Meaning	Asserted—Indicates that the address tenure of a transaction should be terminated. On the following bus clock cycle, the MPC7450, as address bus master, releases the address and attribute signals to high impedance (unless the MPC7450 received another qualified bus grant and is in address streaming mode), and samples $\overline{\text{ARTRY}}$ to determine a qualified $\overline{\text{ARTRY}}$ condition. As a snooping device, the MPC7450 requires an assertion of $\overline{\text{AACK}}$ for every assertion of $\overline{\text{TS}}$ that it detects. Negated—(During an address tenure) 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; 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 $\overline{\text{AACK}}$. Because $\overline{\text{AACK}}$ is not in the qualified bus grant equation, the bus arbiter must negate $\overline{\text{BG}}$ in every cycle the arbiter is delaying $\overline{\text{AACK}}$ to prevent a qualified bus grant when a delayed $\overline{\text{AACK}}$ is desired.

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

8.2.8.2 Address Retry ($\overline{\text{ARTRY}}$)

The address retry ($\overline{\text{ARTRY}}$) signal is both an input and output signal on the MPC7450.

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

Following are the state meaning and timing comments for $\overline{\text{ARTRY}}$ as an output signal in MPX bus mode.

State Meaning Asserted—Indicates that the MPC7450, as a snooping device, detects a condition in which a snooped address tenure must be retried. If the MPC7450 needs to update memory as a result of the snoop that caused the retry, the MPC7450 asserts $\overline{\text{BR}}$ in the bus clock cycle following the assertion of $\overline{\text{ARTRY}}$.

High Impedance—Indicates that the MPC7450 does not need the snooped address tenure to be retried.

Timing Comments Assertion—Asserted as early as the second bus cycle following the assertion of $\overline{\text{TS}}$ if a retry is required. $\overline{\text{ARTRY}}$ will remain asserted until the cycle following the assertion of $\overline{\text{AACK}}$. Note that the MPC7450 requires a minimum of five core cycles to process a snoop and generate a response after latching $\overline{\text{TS}}$ and associated transfer attributes. As a result, if the processor core frequency is less than five times the system bus frequency, $\overline{\text{ARTRY}}$ is asserted later than the second bus cycle following the assertion of $\overline{\text{TS}}$.

Negation/High Impedance—Driven asserted until the bus clock cycle following the assertion of $\overline{\text{AACK}}$. Because this signal may be simultaneously driven by multiple devices, it negates in a unique fashion. First the output buffer goes to high impedance for a fraction of a bus clock cycle (dependent on the clock mode—minimum of one-half of a bus clock cycle), then it is driven negated for one bus clock cycle before returning to high impedance.

This special method of negation may be disabled by setting the precharge disable bit (HID1[PAR]).

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

Following are the state meaning and timing comments for the $\overline{\text{ARTRY}}$ input signal in MPX bus mode.

State Meaning Asserted—If the MPC7450 is the address bus master, $\overline{\text{ARTRY}}$ indicates that the MPC7450 must retry and rerun the entire transaction (address and data tenure). The MPC7450 does not support transfer of data before the $\overline{\text{ARTRY}}$ window. If the

MPC7450 is in address streaming mode and has started the next transaction, it will also be aborted.

If the MPC7450 is not the address bus master, this input indicates that the MPC7450 must immediately negate $\overline{\text{BR}}$ to allow an opportunity for a copyback operation to main memory after a snooping bus master asserts $\overline{\text{ARTRY}}$. 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.

Note that the MPC7450 ignores the $\overline{\text{BG}}$ signal on the cycle in which $\overline{\text{ARTRY}}$ is detected and the cycle following the assertion of $\overline{\text{ARTRY}}$.

Negated/High Impedance—Indicates that the MPC7450 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 remain asserted until the clock cycle following the assertion of $\overline{\text{AACK}}$.

Negation/High Impedance—Must occur two bus clock cycles after the assertion of $\overline{\text{AACK}}$.

Note that during the second bus clock cycle after the assertion of $\overline{\text{AACK}}$, masters release $\overline{\text{ARTRY}}$ to high impedance and then negate it. Thus, care must be taken when sampling $\overline{\text{ARTRY}}$ during this clock period as it could be sampled in an indeterminate state.

8.2.8.3 Shared ($\overline{\text{SHD0}}$, $\overline{\text{SHD1}}$) Signals

The $\overline{\text{SHD0}}$ and $\overline{\text{SHD1}}$ signals act together to indicate a shared snoop response. The MPX bus mode interface allows a given master to drive a new address tenure every other cycle, so the shared signal must be able to be driven every other cycle. But, because it must be actively negated and might be driven by multiple masters at any given time, electrical requirements dictate that two shared signals be implemented. When signaling a snoop response of shared, the MPC7450 must assert $\overline{\text{SHD0}}$ unless $\overline{\text{SHD0}}$ was asserted in any of the three cycles prior to the snoop response window for the current transaction. In that case, the MPC7450 asserts $\overline{\text{SHD1}}$. Thus, both $\overline{\text{SHD0}}$ and $\overline{\text{SHD1}}$ can be released to high-impedance, driven negated, then released to high-impedance again before they need to be reasserted. When the MPC7450 is a bus master, the MPC7450 considers the snoop response to be shared if either $\overline{\text{SHD0}}$ or $\overline{\text{SHD1}}$ is asserted. Note that the $\overline{\text{SHD1}}$ signal is only provided in MPX bus mode.

8.2.8.3.1 Shared ($\overline{\text{SHD0}}$, $\overline{\text{SHD1}}$)—Output

If $\overline{\text{SHD0}}$ was not asserted the cycle before or the cycle of $\overline{\text{TS}}$, $\overline{\text{SHD0}}$ should be asserted to indicate a shared response. But if $\overline{\text{SHD0}}$ was asserted in any of the three cycles before the snoop response window for the current transaction then $\overline{\text{SHD1}}$ is used to indicate a shared response in this cycle. Following are the state meaning and timing comments for $\overline{\text{SHD0}}$ and $\overline{\text{SHD1}}$ as output signals in MPX bus mode.

State Meaning Asserted—If $\overline{\text{ARTRY}}$ is negated, $\overline{\text{SHD0}}$ and $\overline{\text{SHD1}}$ indicate that the MPC7450 had a cache hit on a shared block or the reservation address.

If $\overline{\text{ARTRY}}$ is asserted, the $\overline{\text{SHD0}}$ and $\overline{\text{SHD1}}$ are don’t care.

Negated/High Impedance—Indicates that the processor did not contain the data or has invalidated the snooped address.

Timing Comments Assertion/Negation—If the MPC7450 needs to assert a shared snoop response and $\overline{\text{SHD0}}$ was not asserted in the cycle before or the cycle of $\overline{\text{TS}}$, it should be asserted in the snoop response window to indicate a shared response. If $\overline{\text{SHD0}}$ was asserted the cycle before or the cycle of $\overline{\text{TS}}$, then $\overline{\text{SHD1}}$ must be used to indicate a shared response.A master observing the snoop response must consider the shared response asserted if either $\overline{\text{SHD0}}$ or $\overline{\text{SHD1}}$ is asserted.

High Impedance—The timing of $\overline{\text{SHD0}}$ and $\overline{\text{SHD1}}$ for the release to high-impedance, negating, and re-release to high-impedance, may vary. To ensure compatibility with the standard 60x interface in which $\overline{\text{SHD}n}$ might need to be asserted up to every three bus cycles, the MPC7450 implements the 60x-style timing for both $\overline{\text{SHD0}}$ and $\overline{\text{SHD1}}$; that is $\overline{\text{SHD0}}$ and $\overline{\text{SHD1}}$ have the same timing as $\overline{\text{ARTRY}}$, in which the signal is released to high-impedance for a fraction of a cycle, then negated for up to an entire cycle (crossing a bus cycle boundary) before being released to high-impedance again. Note that future implementations with the MPX bus protocol may define this timing differently. The MPC7450 does not assert either $\overline{\text{SHD0}}$ or $\overline{\text{SHD1}}$ any more often than every fourth bus clock cycle.

8.2.8.3.2 Shared ($\overline{\text{SHD0}}$, $\overline{\text{SHD1}}$)—Input

Following are the state meaning and timing comments for $\overline{\text{SHD0}}$ and $\overline{\text{SHD1}}$ as input signals in MPX bus mode.

State Meaning Asserted—If $\overline{\text{ARTRY}}$ is negated, the MPC7450 uses $\overline{\text{SHD0}}$ and $\overline{\text{SHD1}}$ to allocate the incoming cache block as shared (S) for a self-generated transaction. Applies only to rclaim, read, and read atomic transactions.

If $\overline{\text{ARTRY}}$ is asserted, $\overline{\text{SHD0}}$ and $\overline{\text{SHD1}}$ are ignored as an input.

Negated—If $\overline{\text{ARTRY}}$ is negated and $\overline{\text{SHD0}}$ and $\overline{\text{SHD1}}$ are negated, the MPC7450 allocates the incoming cache block as exclusive (E) for a self-generated read or read-atomic transaction or for an rclaim transaction.

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

8.2.8.4 Snoop Hit ($\overline{\text{HIT}}$)—Output

The snoop response of the MPC7450 (or local bus slave) uses the $\overline{\text{HIT}}$ output signal to communicate to the system arbiter whether or not data intervention occurs for the current transaction. See Section 9.3, "MPX Bus Address Tenure," and Section 9.4, "MPX Bus Data Tenure," for more detailed information about the data-only transactions used by the MPC7450 in MPX bus mode for data intervention.

This signal is a snoop response, valid in the $\overline{\text{ARTRY}}$ window (the cycle after $\overline{\text{AACK}}$) that indicates that the MPC7450 as a snooper will supply intervention data. That is, the MPC7450 has found the data in its cache that has been requested by another master's bus transaction. Instead of asserting $\overline{\text{ARTRY}}$ and flushing the data to memory, the MPC7450 asserts $\overline{\text{HIT}}$ the cycle after $\overline{\text{AACK}}$ to indicate that it can supply the data directly to the other master.

The MPC7450 does not implement the optional protocol of the MPX bus to communicate to the system whether or not the intervention data needs to be forwarded to memory. The system needs to identify which transactions it is required to snarf and which transactions it is not required to snarf.

It is possible for the MPC7450 to assert simultaneously both $\overline{\text{ARTRY}}$ and $\overline{\text{HIT}}$ for the same snoop response. When simultaneously asserted, $\overline{\text{ARTRY}}$ supersedes $\overline{\text{HIT}}$ and $\overline{\text{HIT}}$ should be ignored by the system.

Following are the state meaning and timing comments for the $\overline{\text{HIT}}$ signal.

State Meaning

Asserted—The MPC7450 has the requested data in its cache and will supply it through a data-only transaction.

Negated—The MPC7450 cannot provide data for a snoop request through the $\overline{\text{HIT}}$ intervention protocol.

Timing Comments

Asserted— $\overline{\text{HIT}}$ is driven the cycle after $\overline{\text{AACK}}$.

$\overline{\text{HIT}}$ is held asserted for one cycle beyond the assertion of $\overline{\text{AACK}}$ if the snoop hit data is modified and must be forwarded to memory.

Negated—$\overline{\text{HIT}}$ is negated two cycles after $\overline{\text{AACK}}$.

8.2.9 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 9.4.1, “MPX Bus Data Bus Arbitration.”

8.2.9.1 Data Bus Grant ($\overline{\text{DBG}}$)—Input

The data bus grant ($\overline{\text{DBG}}$) signal is an input-only signal on the MPC7450. Following are the state meaning and timing comments for the $\overline{\text{DBG}}$ signal.

State Meaning

Asserted—Indicates that the MPC7450 may assume ownership of the data bus with the proper qualification:

QDBG = $\overline{\text{DBG}}$ & ¬($\overline{\text{ARTRY}}$ & retriable) & ¬(state_variables)

where:

- $\overline{\text{ARTRY}}$ is only for the address bus tenure associated with the data bus tenure about to be granted (that is, not from another address tenure due to address pipelining).
- Retriable indicates whether the current transaction can still be retried.
- State variables include whether or not:
  - —The data bus is being used by this master.
  - —The master has back-to-back burst accesses in progress.
  - —The processor is currently receiving the last $\overline{\text{TA}}$ for the current burst.

Thus, a qualified data bus grant occurs when:

- $\overline{\text{DBG}}$ is asserted
- $\overline{\text{ARTRY}}$ was negated in the address retry window for the address phase of this transaction
- The MPC7450 is ready to begin a data transaction

Note that data streaming is allowed in MPX bus mode.

Negated—Indicates that the MPC7450 must hold off its data tenures.

Timing Comments Assertion—May occur any time to indicate the MPC7450 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 MPC7450 cannot assume data bus mastership.

8.2.9.2 Data Transaction Index (DTI[0:3])—Input

The MPC7450 can be configured to support a generalized reordering scheme using the new 4-bit data transfer index (DTI[0:3]) input signals.

The DTI signals can be bused or point-to-point. They must be driven valid by the system arbiter on the cycle before a data bus grant ($\overline{\text{DBG}}$). They are sampled on each bus clock cycle by the MPC7450 and are qualified by the assertion of $\overline{\text{DBG}}$ on the following cycle.

The data transfer index is a pointer into the MPC7450’s queue of outstanding transactions, indicating which transaction is to be serviced by the subsequent data tenure. The number of outstanding transactions is configurable to 2–8 or 16. The default is 8 outstanding transactions.

Data tenure reordering can be disabled by clearing DTI[0:3] to 0b0000. This setting causes the MPC7450 to always select the oldest transaction in the outstanding transaction queue. See Section 9.4.2.4.5, “Ordering of Data-Only Transactions.”

Following are the state meaning and timing comments for the DTI[0:3] signals.

State Meaning Asserted—The DTI[0:3] signals act as a pointer into the queue of outstanding transactions for the MPC7450, indicating which transaction is to be served by the subsequent data tenure. For example, DTI = 0b0000 means that the oldest transaction is to be serviced, DTI = 0b0001 means the second oldest transaction is to be serviced up to DTI = 0b1111 meaning the 16th oldest transaction is to be serviced.

Negated—DTI = 0b0000 indicates that the MPC7450 must run the data bus tenures in the same order as the address tenures. DTI[0–3] must be driven negated in 60x mode.

Timing Comments Assertion/Negation—Sampled each cycle and qualified by a qualified $\overline{\text{DBG}}$ in the following cycle.

8.2.9.3 Data Ready ($\overline{\text{DRDY}}$)—Output

The data ready ($\overline{\text{DRDY}}$) signal is a point-to-point output signal from the MPC7450 to the system arbiter. It functions as a data bus request indicating to the arbiter that data for an outstanding data intervention transaction previously signaled with a $\overline{\text{HIT}}$ is ready. The arbiter responds by granting the data bus. Note that the EIDIS bit of MSSCR0 disables data intervention for the MPC7450 caches. See Section 2.2.5.3, “Memory Subsystem Control Register (MSSCR0).” Also, see Section 9.4, “MPX Bus Data Tenure,” for more information about the data intervention functionality. Following are the state meaning and timing comments for the $\overline{\text{DRDY}}$ signal.

State Meaning	Asserted—The MPC7450 has data ready for a pending bus operation initiated elsewhere in the system (for which the MPC7450 has previously signaled $\overline{\text{HIT}}$ during the snoop response window), and the MPC7450 is requesting the data bus in order to service that bus operation.
	Negated—The MPC7450 is not requesting the data bus to service an outstanding bus request.
Timing Comments	Asserted—$\overline{\text{DRDY}}$ is asserted no earlier than $\overline{\text{HIT}}$ and no earlier than two cycles before the MPC7450 is able to drive the data (because $\overline{\text{DRDY}}$ may be followed immediately by $\overline{\text{DBG}}$ and then $\overline{\text{TA}}$).
	Negated—$\overline{\text{DRDY}}$ is negated on the cycle after it is asserted unless another $\overline{\text{DRDY}}$ is asserted for the next transaction. $\overline{\text{DRDY}}$ may be fully pipelined on back-to-back cycles when multiple hits are outstanding.

8.2.10 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. They are also used for data-only transactions. For a detailed description of how the data transfer signals interact, see Section 9.4.2.4.1, “Data-Only Transaction Protocol.”

8.2.10.1 Data Bus (D[0:63])

The data bus (D[0:63]) consists of 64 signals that are both inputs and outputs on the MPC7450. The data bus is driven once for cache-inhibited or write-through transactions of 64 bits or less, two times for $\overline{\text{CI}}$ or $\overline{\text{WT}}$ AltiVec quad-word loads and stores, and four times for cache-line burst transactions. See Table 8-5 for the data bus lane assignments.

Table 8-5. Data Bus Lane Assignments

Data Bus Signals	Byte Lane
D[0:7]	0
D[8:15]	1
D[16:23]	2
D[24:31]	3
D[32:39]	4
D[40:47]	5

Table 8-5. Data Bus Lane Assignments (continued)

Data Bus Signals	Byte Lane
D[48:55]	6
D[56:63]	7

8.2.10.1.1 Data Bus (D[0:63])—Output

Following are the state meaning and timing comments for D[0:63] as output signals.

State Meaning Asserted/Negated—Represent the state of data during data-write (including data-only [data intervention]) transactions. Byte lanes not selected for data transfer do not supply valid data.

Timing Comments Assertion/Negation—Initial beat occurs one cycle after a qualified $\overline{\mathrm{DBG}}$ is sampled, and, for bursts, changes one bus cycle following each assertion of $\overline{\mathrm{TA}}$.

High Impedance—Occurs on the bus clock cycle after the final assertion of $\overline{\mathrm{TA}}$, following the assertion of $\overline{\mathrm{TEA}}$, or in certain $\overline{\mathrm{ARTRY}}$ cases

8.2.10.1.2 Data Bus (D[0:63])—Input

Following are the state meaning and timing comments for D[0:63] as input signals.

State Meaning Asserted/Negated—Represent the state of data during a data read transaction.

Timing Comments Assertion/Negation—Data must be valid on the same bus clock cycle that $\overline{\mathrm{TA}}$ is asserted.

8.2.10.2 Data Bus Parity (DP[0:7])

The eight data bus parity (DP[0:7]) signals on the MPC7450 are both input and output. They can also be used for data-only transactions.

8.2.10.2.1 Data Bus Parity (DP[0:7])—Output

Following are the state meaning and timing comments for DP[0:7] as output signals.

State Meaning Asserted/Negated—Represents odd parity for each of the eight bytes during data write transactions. Odd parity means that an odd number of bits, including the parity bit, are driven high. All eight parity bits are driven with valid parity on all bus operations. HID1[EBA] and HID1[EBD] control whether control whether the processor will check address and data parity respectively. The MPC7450 always generates parity regardless of whether checking is enabled or disabled. The signal assignments are listed in Table 8-6.

Timing Comments Assertion/Negation—The same as (D[0:63])
High Impedance—The same as (D[0:63])

Table 8-6. DP[0:7] Signal Assignments

Signal Name	Signal Assignments
DP0	D[0:7]
DP1	D[8:15]
DP2	D[16:23]
DP3	D[24:31]
DP4	D[32:39]
DP5	D[40:47]
DP6	D[48:55]
DP7	D[56:63]

8.2.10.2.2 Data Bus Parity (DP[0:7])—Input

Following are the state meaning and timing comments for DP[0:7] as 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 machine check exception if data parity errors are enabled in the machine-specific HID1 register.

Timing Comments Assertion/Negation—The same as (D[0:63])

8.2.11 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 9.4.3, “MPX Bus Data Tenure Termination.”

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

Following are the state meaning and timing comments for the $\overline{\text{TA}}$ signal.

State Meaning Asserted—Indicates that a single-beat data transfer or a data beat in a burst transfer completed successfully. On the following cycle, the MPC7450 terminates the data beat, or if a burst, advances to the next data beat. If it is the last or only data beat, MPC7450 also terminates the data tenure. Note that $\overline{\text{TA}}$ must be asserted for each data beat in a burst transaction.

Negated—(During a data tenure) indicates that, until $\overline{\text{TA}}$ is asserted, the MPC7450 must continue to drive the data (insert wait states) for the current write or must wait to sample the data for reads. Note that it is the responsibility of the system to ensure that $\overline{\text{TA}}$ is negated by the start of the next data bus tenure.

Timing Comments Assertion—Must not occur before the last cycle of the snoop response window (cycle after AACK). The system can withhold assertion of $\overline{\text{TA}}$ to indicate that the MPC7450 should insert wait states to extend the duration of the data beat, for details see Section 9.4.2.2.1, “Data Streaming in MPX Bus Mode.”

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.

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

Following are the state meaning and timing comments for the $\overline{\text{TEA}}$ signal.

State Meaning Asserted—Indicates that a data bus error occurred. On the following cycle, the MPC7450 must terminate the current data tenure (assertion of $\overline{\text{TA}}$ is ignored). 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 4.6.2.2, “Checkstop State (MSR[ME] = 0).” For reads, assertion of $\overline{\text{TEA}}$ does not invalidate data entering the GPRs or the caches.

Negated—Indicates that no bus error was detected.

Timing Comments Assertion—May be asserted on any bus clock cycle during a normal data tenure, from the cycle following a qualified data bus grant to the cycle of the final $\overline{\text{TA}}$. $\overline{\text{TEA}}$ should be asserted for one cycle only. $\overline{\text{TEA}}$ must not be asserted before the last cycle of the snoop response window. If $\overline{\text{TEA}}$ is asserted at the same time as $\overline{\text{ARTRY}}$, then $\overline{\text{ARTRY}}$ takes precedence and the address tenure will be rerun. It is the responsibility of the system to ensure that this scenario does not recur (causing a deadlock).

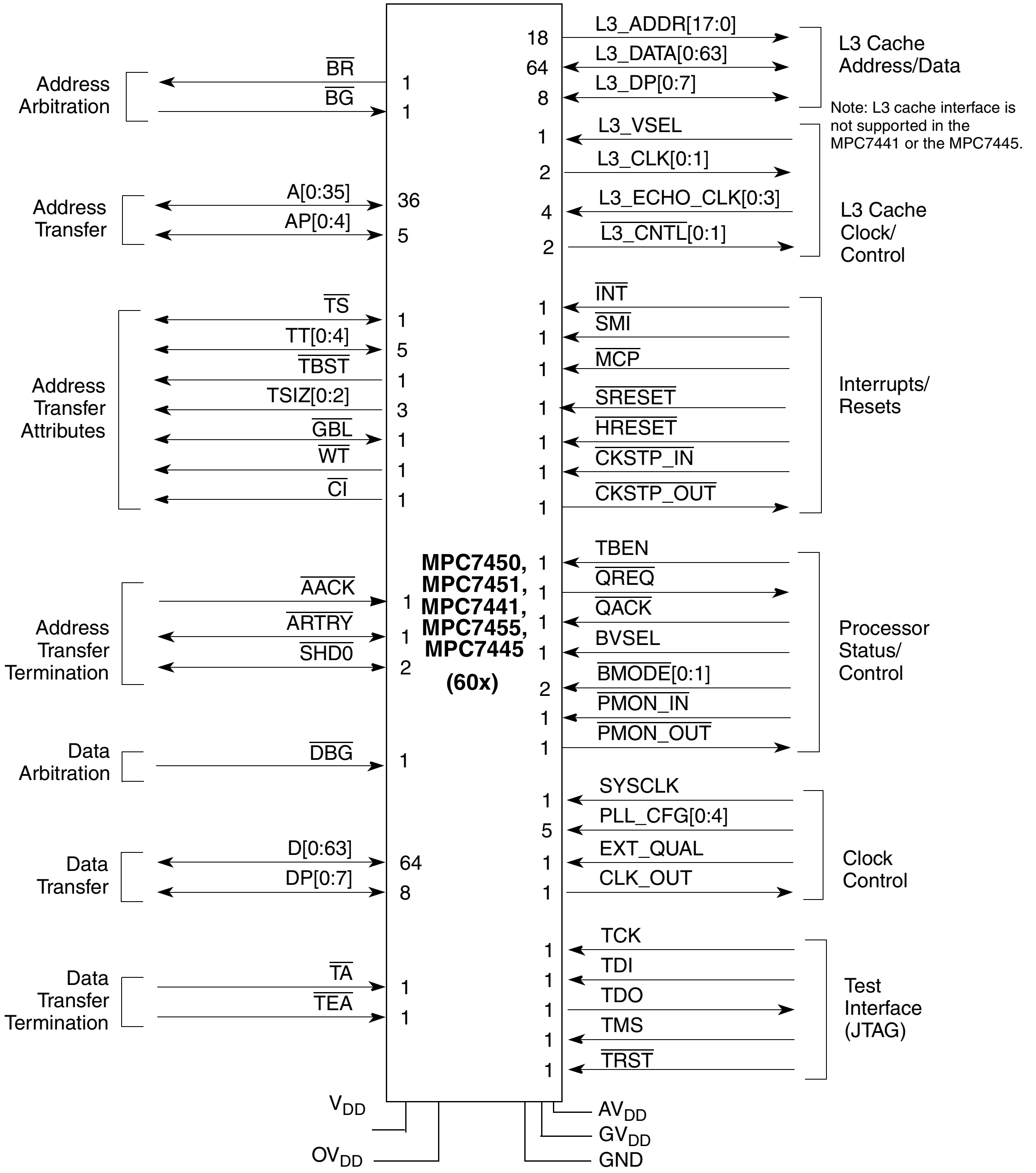
Negation—$\overline{\text{TEA}}$ must be negated one cycle after it is asserted. Note that it is the responsibility of the system to ensure that $\overline{\text{TEA}}$ is negated by the start of the next data bus tenure.

8.3 60x Bus Signal Configuration

The signals that implement the 60x bus protocol on the MPC7450 are very similar to those of MPX bus mode, with the exceptions noted in the following subsections.

8.3.1 60x Bus Mode Functional Groupings

Figure 8-5 shows how the signals are grouped in the 60x bus mode for the MPC7450, MPC7451, MPC7441, MPC7455, and MPC7445. A pinout showing pin numbers is included in the hardware specifications. Note that the left side of each figure depicts the signals that implement the 60x bus protocol and the right side of each figure shows the remaining signals on the MPC7450 (not part of the bus protocol).

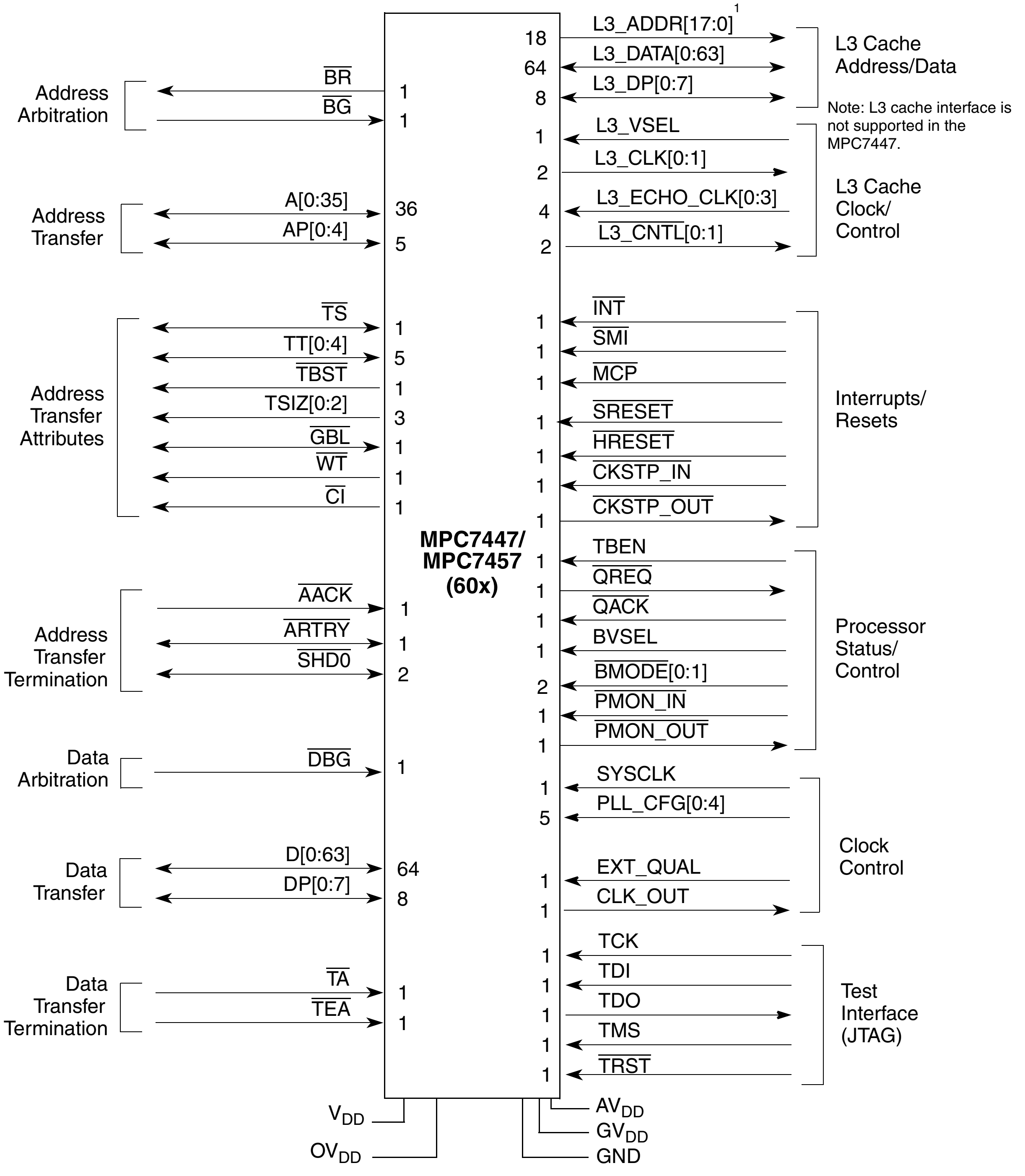


Note: L3 cache interface is not supported in the MPC7441 or the MPC7445.

Note: The DTI[0:3] signals are not functional in 60x mode.

Figure 8-5. 60x Bus Signal Groups in the MPC7450, MPC7451, MPC7441, MPC7455, and MPC7445

Figure 8-6 illustrates the signal configuration in 60x bus mode for the MPC7447 and the MPC7457.

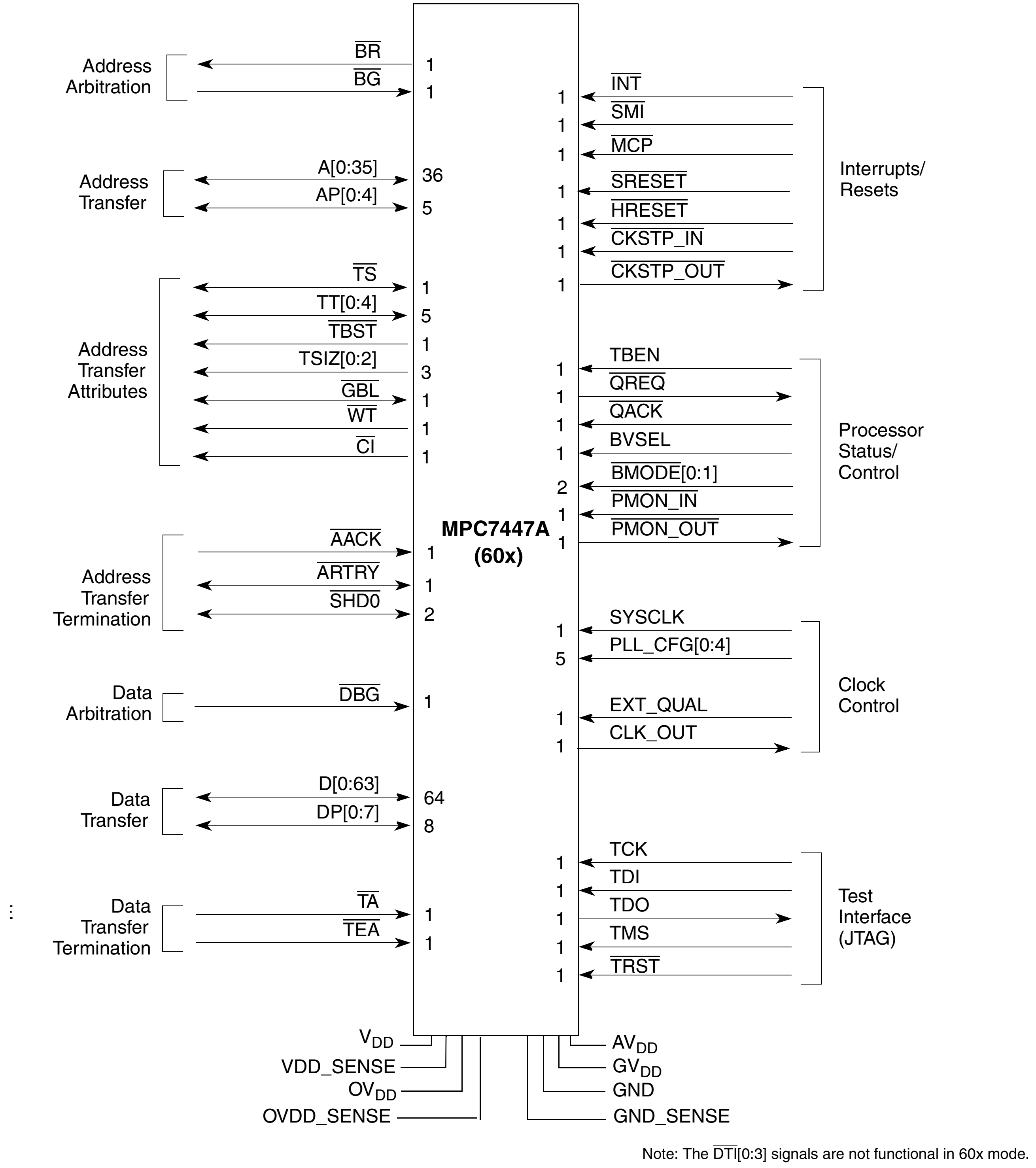


[1] For the MPC7457, there are 19 L3_ADDR signals, (L3_ADDR[0:18]).

Note: The DTI[0:3] signals are not functional in 60x mode.

Figure 8-6. 60x Bus Signal Groups in the MPC7447 and the MPC7457

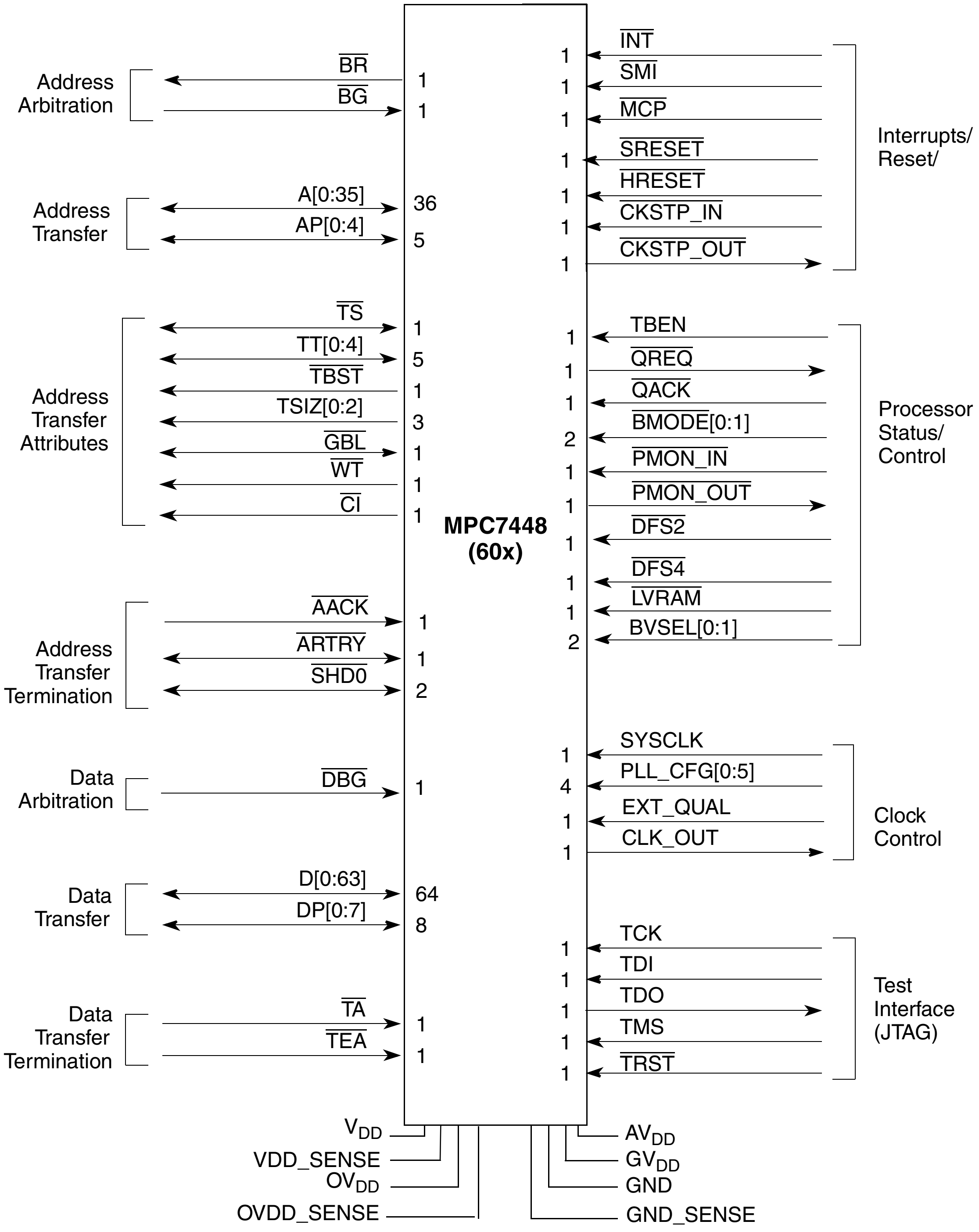
Figure 8-7 illustrates the signal configuration in 60x bus mode for the MPC7447A.



Note: The DTI[0:3] signals are not functional in 60x mode.

Figure 8-7. 60x Bus Signal Groups in the MPC7447A

Figure 8-8 illustrates the signal configuration in 60x bus mode for the MPC7448.



Note: The DTI[0:3] signals are not functional in 60x mode.

Figure 8-8. 60x Bus Signal Groups in the MPC7448

8.3.2 60x Address Bus Arbitration Signals

The address arbitration signals are the input and output signals the MPC7450 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 9.6.1, “60x Bus Address Bus Arbitration.”

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

Following are the state meaning and timing comments for the $\overline{\text{BR}}$ output signal on the MPC7450 in 60x bus mode.

State Meaning Asserted—Same as MPX bus interface.

Negated—Same as MPX bus interface.

Timing Comments Assertion—Same as MPX bus interface.

Negation—Same as MPX bus interface, except that the $\overline{\text{BR}}$ signal is negated during $\overline{\text{TS}}$.

High Impedance—Same as MPX bus interface.

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

Following are the state meaning and timing comments for the $\overline{\text{BG}}$ input signal.

State Meaning Asserted—Indicates that the MPC7450 may, with proper qualification, assume mastership of the address bus. The conditions for a qualified bus grant are described in Section 9.6.1, “60x Bus Address Bus Arbitration.”

Negated— Indicates that the MPC7450 is not the next potential address bus master.

Timing Comments Assertion—May occur at any time to indicate the MPC7450 can use the address bus. In 60x bus mode, the MPC7450 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 MPC7450 cannot use the bus. The MPC7450 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 MPC7450 that it could take mastership (if qualified).

8.3.3 Address Bus and Parity in 60x Bus Mode

The address bus (A[0:35]) consists of 36 signals that are both input and output signals. The following sections describe the address bus and parity signals in 60x bus mode. For a detailed description of how these signals interact, refer to Section 9.6.2, “60x Bus Address Transfer.”

8.3.3.1 Address Bus (A[0:35])—Output

Following are the state meaning and timing comments for the address bus A[0:35] as output signals on the MPC7450 in 60x bus mode.

State Meaning	Asserted/Negated—Same as MPX bus interface.
Timing Comments	Assertion/Negation—Same as MPX bus interface.
	High Impedance—Occurs one bus clock cycle after $\overline{\text{AACK}}$ is asserted.

8.3.3.2 Address Bus (A[0:35])—Input

Following are the state meaning and timing comments for the address bus A[0:35] as input signals on the MPC7450 in 60x bus mode.

State Meaning	Asserted/Negated—Same as MPX bus interface.
Timing Comments	Assertion/Negation—Same as MPX bus interface.

8.3.3.3 Address Parity (AP[0:4])—Output

Following are the state meaning and timing comments for AP[0:4] as output signals on the MPC7450. See Table 8-4.

State Meaning	Asserted/Negated—Same as A[0:35].
Timing Comments	Assertion/Negation—Same as A[0:35].

8.3.3.4 Address Parity (AP[0:4])—Input

Following are the state meaning and timing comments for AP[0:4] as input signals on the MPC7450.

State Meaning	Asserted/Negated—Same as A[0:35].
Timing Comments	Assertion/Negation—Same as A[0:35].

8.3.4 Address Transfer Attribute Signals in 60x Bus Mode

The transfer attribute signal functions in 60x bus mode are very similar to that of MPX bus mode, with the exceptions noted in the following subsections.

8.3.4.1 Transfer Start ($\overline{\text{TS}}$)

The transfer start ($\overline{\text{TS}}$) signal is both an input and output signal on the MPC7450.

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

Following are the state meaning and timing comments for $\overline{\text{TS}}$ an output signal on the MPC7450.

State Meaning	Asserted—Same as MPX bus interface. Negated—Same as MPX bus interface.
Timing Comments	Assertion—Same as MPX bus interface. Negation—Same as MPX bus interface. High Impedance—Same as MPX bus interface.

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

Following are the state meaning and timing comments for $\overline{\text{TS}}$ as an input signal on the MPC7450.

State Meaning	Asserted—Same as MPX bus interface. Negated—Same as MPX bus interface.
Timing Comments	Assertion—May occur on any cycle following a qualified $\overline{\text{BG}}$. Negation— Same as MPX bus interface.

8.3.4.2 Transfer Type (TT[0:4])

The transfer type (TT[0:4]) signals consist of five input/output signals on the MPC7450. For a complete description of the TT[0:4] signals for 60x bus mode, see Section 9.3.2.4, “Address Transfer Attributes.”

8.3.4.2.1 Transfer Type (TT[0:4])—Output

Following are the state meaning and timing comments for TT[0:4] as output signals on the MPC7450 in 60x bus mode.

State Meaning	Asserted/Negated—Indicates the type of transfer in progress.
Timing Comments	Assertion/Negation—Same as A[0:35]. High Impedance—Same as A[0:35]. For details on timing for the TT[0:4] signals for 60x bus mode see Section 9.8, “60x Bus Timing Examples.”

8.3.4.2.2 Transfer Type (TT[0:4])—Input

Following are the state meaning and timing comments for TT[0:4] as input signals on the MPC7450 in 60x bus mode.

State Meaning	Asserted/Negated–Indicates the type of transfer in progress.
Timing Comments	Assertion/Negation—Same as A[0:35]. For details on timing for the TT[0:4] signals for 60x bus mode see Section 9.8, “60x Bus Timing Examples.”

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

The transfer burst ($\overline{\text{TBST}}$) signal is an output-only signal on the MPC7450. Following are the state meaning and timing comments for the transfer burst $\overline{\text{TBST}}$ output signal in 60x bus mode.

State Meaning	Asserted—Same as MPX bus interface.
	Negated—Same as MPX bus interface.
Timing Comments	Assertion/Negation—Same as A[0:35].
	High Impedance—Same as A[0:35].

8.3.4.4 Transfer Size (TSIZ[0:2])—Output

Following are the state meaning and timing comments for the transfer size TSIZ[0:2] output signals on the MPC7450 in 60x bus mode.

State Meaning	Asserted/Negated—Same as MPX bus interface.
Timing Comments	Assertion/Negation—Same as A[0:35].
	High Impedance—Same as A[0:35].

8.3.4.5 Global ($\overline{\text{GBL}}$)

The global ($\overline{\text{GBL}}$) signal is an input/output signal on the MPC7450.

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

Following are the state meaning and timing comments for $\overline{\text{GBL}}$ as an output signal on the MPC7450 in 60x bus mode.

State Meaning	Asserted—Same as MPX bus interface.
	Negated—Same as MPX bus interface.
Timing Comments	Assertion/Negation—Same as A[0:35].
	High Impedance—The same as A[0:35].

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

Following are the state meaning and timing comments for $\overline{\text{GBL}}$ as an input signal on the MPC7450 in 60x bus mode.

State Meaning	Asserted—Same as MPX bus interface.
	Negated—Same as MPX bus interface.
Timing Comments	Assertion/Negation—Same as A[0:35].

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

Following are the state meaning and timing comments for the write-through $\overline{\text{WT}}$ output signal on the MPC7450 in 60x bus mode.

State Meaning	Asserted/Negated—Same as MPX bus interface.
Timing Comments	Assertion/Negation—Same as A[0:35]. High Impedance—Same as A[0:35].

8.3.4.7 Cache Inhibit ($\overline{\text{CI}}$)—Output

The cache inhibit ($\overline{\text{CI}}$) signal is an output signal on the MPC7450 in 60x bus mode and following are its state meaning and timing comments.

State Meaning	Asserted—Same as MPX bus interface. Negated—Same as MPX bus interface.
Timing Comments	Assertion/Negation—The same as A[0:35]. High Impedance—The same as A[0:35].

8.3.5 60x Address Transfer Termination Signals

The address transfer termination signal functions in 60x bus mode are very similar to that of MPX bus mode, with the exceptions noted in the following subsections. For detailed information about how these signals interact, see Section 9.6.3, “60x Bus Address Transfer Termination.”

8.3.5.1 Address Acknowledge ($\overline{\text{AACK}}$)—Input

The address acknowledge ($\overline{\text{AACK}}$) signal is an input-only signal on the MPC7450. 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 is released to high-impedance on the next bus clock cycle. Note that the address tenure does not terminate until the assertion of $\overline{\text{AACK}}$. As a snooping device, the MPC7450 requires that $\overline{\text{AACK}}$ be asserted for every assertion of $\overline{\text{TS}}$ that it detects. Negated—Same as MPX bus interface.
Timing Comments	Assertion—May occur as early as the bus clock cycle after $\overline{\text{TS}}$ is asserted; assertion can be delayed to allow adequate address access time for slow devices. For example, if an implementation supports slow snooping devices, an external arbiter can postpone the assertion of $\overline{\text{AACK}}$. Negation—Same as MPX bus interface.

8.3.5.2 Address Retry ($\overline{\text{ARTRY}}$)

The address retry ($\overline{\text{ARTRY}}$) signal is both an input and output signal on the MPC7450 in 60x bus mode.

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

Following are the state meaning and timing comments for $\overline{\text{ARTRY}}$ as an output signal in 60x bus mode.

State Meaning	Asserted—Same as MPX bus interface. Negation/High Impedance—Same as MPX bus interface.
Timing Comments	Assertion—Same as MPX bus interface.

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

Following are the state meaning and timing comments for $\overline{\text{ARTRY}}$ as an input signal in 60x bus mode.

State Meaning	Asserted—If the MPC7450 is the address bus master, $\overline{\text{ARTRY}}$ indicates that the MPC7450 must retry the preceding address tenure and immediately negate $\overline{\text{BR}}$ (if asserted). If the associated data tenure has already started, the MPC7450 also aborts the data tenure immediately, even if data has been received. If the MPC7450 is not the address bus master, this input indicates that the MPC7450 must immediately negate $\overline{\text{BR}}$ to allow an opportunity for a copyback operation to main memory after a snooping bus master asserts $\overline{\text{ARTRY}}$. 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. Note that the MPC7450 ignores the $\overline{\text{BG}}$ signal on the cycle in which $\overline{\text{ARTRY}}$ is detected and the cycle following the assertion of $\overline{\text{ARTRY}}$. Negated—Same as MPX bus interface.
Timing Comments	Assertion—Same as MPX bus interface. Negation/High Impedance—Same as MPX bus interface.

8.3.5.3 Shared ($\overline{\text{SHD0}}$)

The shared $\overline{\text{SHD0}}$ signal is both an input and an output on the MPC7450 in 60x bus mode and functions similarly to $\overline{\text{SHD0}}$ and $\overline{\text{SHD1}}$ in MPX bus mode. Because the 60x protocol does not allow a given master to drive a new address tenure every other cycle as does the MPX protocol, only one snoop response signal, $\overline{\text{SHD0}}$, is necessary.

8.3.5.3.1 Shared (SHD0)—Output

Following are state and timing descriptions for shared (SHD0) as an output signal.

State Meaning	Asserted—If ARTRY is negated, SHD0 indicates that after this transaction completes successfully, the MPC7450 will keep a valid shared copy of the address or that a reservation exists on this address. If SHD0 and ARTRY are asserted for a snooping master, the snoop hit modified data is pushed as the master's next address transaction.
	Negated/High Impedance—After this address is transferred, the processor no longer has a valid copy of the snooped address.
Timing Comments	Assertion/Negation—Same as ARTRY.
	High Impedance—Same as ARTRY.

8.3.5.3.2 Shared (SHD0)—Input

Following are state and timing descriptions for (SHD0) as an input signal.

State Meaning	Asserted—If ARTRY is negated, the MPC7450 allocates the incoming cache block as shared (S) for a self-generated transaction. Applies only to read and read atomic transactions.
	If ARTRY is asserted, SHD0 is ignored as an input.
	Negated—If ARTRY is negated and SHD0 is negated, the MPC7450 allocates the incoming cache block as exclusive (E) for a self-generated read or read-atomic transaction.
Timing Comments	Assertion/Negation—Same as ARTRY

8.3.6 Data Bus Arbitration Signals

The data bus arbitration signals for 60x bus mode operate similarly to MPX bus mode except as noted in the following subsections. See Section 9.7.1, "60x Bus Data Bus Arbitration," for more information about data bus arbitration in 60x bus mode.

8.3.6.1 Data Bus Grant (DBG)—Input

The data bus grant (DBG) signal is an input signal on the MPC7450. Following are the state meaning and timing comments for the DBG signal in 60x bus mode.

State Meaning	Asserted—Same as MPX bus interface, except that data streaming is not allowed in 60x bus mode.
	Negated—Same as MPX bus interface.
Timing Comments	Assertion—Same as MPX bus interface.
	Negation—Same as MPX bus interface.

8.3.6.2 Data Transaction Index (DTI[0:3])—Input

In the MPC7450’s implementation of 60x bus protocol, out-of-order transactions are not supported. Therefore DTI[0:3] signals have no functionality in 60x mode and must be pulled low.

8.3.7 Data Transfer Signals in 60x Bus Mode

The data transfer signals in 60x bus mode transmit data and generate and monitor parity for the data transfer similarly to those in MPX bus mode, except that they are not used for data-only (intervention) transactions. For a detailed description of how the data transfer signals interact in 60x bus mode, see Section 9.7.2, “60x Bus Data Transfers.”

8.3.7.1 Data Bus (D[0:63])

The following subsections describe the operation of the data bus signals as inputs and outputs in 60x bus mode.

8.3.7.1.1 Data Bus (D[0:63])—Output

Following are the state meaning and timing comments for the D[0:63] signals as outputs in 60x bus mode.

State Meaning	Asserted/Negated—Represent the state of data during a data write transaction (excluding data-only transactions that are unsupported in 60x mode). Byte lanes not selected for data transfer do not supply valid data.
Timing Comments	Assertion/Negation—Initial beat occurs one bus clock cycle after a qualified $\overline{\text{DBG}}$ is sampled, and, for bursts, transitions on the bus in the clock cycle following each assertion of $\overline{\text{TA}}$. High Impedance—Same as MPX bus interface.

8.3.7.1.2 Data Bus (D[0:63])—Input

Following are the state meaning and timing comments for the D[0:63] signals as inputs in 60x bus mode.

State Meaning	Asserted/Negated—Same as MPX bus interface, except that data-only transactions are not supported in 60x mode.
Timing Comments	Assertion/Negation—Same as MPX bus interface.

8.3.7.2 Data Bus Parity (DP[0:7])

The following subsections describe the operation of the data bus parity signals (DP[0:7]) as inputs and outputs in 60x bus mode.

8.3.7.2.1 Data Bus Parity (DP[0:7])—Output

Following are the state meaning and timing comments for the DP[0:7] signals as outputs in 60x bus mode.

State Meaning	Asserted/Negated—Same as MPX bus interface, except that data-only transactions are not supported in 60x mode.
	High Impedance—Same as MPX bus interface.
Timing Comments	Assertion/Negation—Same as D[0:63].
	High Impedance—Same as D[0:63].

8.3.7.2.2 Data Bus Parity (DP[0:7])—Input

Following are the state meaning and timing comments for the DP[0:7] signals as inputs in 60x bus mode.

State Meaning	Asserted/Negated—Same as MPX bus interface, except that data-only transactions are not supported in 60x mode.
Timing Comments	Assertion/Negation—Same as D[0:63].

8.3.8 Data Transfer Termination Signals in 60x Bus Mode

The function of the data termination signals in 60x bus mode is similar to that in MPX bus mode. The differences are described in the following subsections. For a detailed description of how these signals interact in 60x bus mode, see Section 9.7.3, “60x Bus Data Tenure Termination.”

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

Following are the state meaning and timing comments for the $\overline{\text{TA}}$ signal.

State Meaning	Asserted—Same as MPX bus interface.
	Negated—Same as MPX bus interface.
Timing Comments	Assertion—Same as MPX bus interface.
	Negation—Same as MPX bus interface.

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

Following are the state meaning and timing comments for the $\overline{\text{TEA}}$ signal.

State Meaning	Asserted—Same as the MPX bus interface.
	Negated—Same as MPX bus interface.
Timing Comments	Assertion—Same as MPX bus interface.
	Negation—Same as MPX bus interface.

8.4 Non-Protocol Signal Descriptions

The following sections describe the signals on the MPC7450 that do not specifically implement the MPX or 60x bus protocols. These signals include the L3 interface signals, the interrupt and reset signals, processor status and control signals, clock control signals, and JTAG test signals. Note that the L3 interface signals are not present on the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448.

8.4.1 L3 Cache Address/Data

The MPC7450's dedicated L3 cache interface provides all the signals required for the support of up to 2 Mbytes of DDR, late write, and PB2 SRAM for data storage as L3 cache or as private memory. The use of the L3 data parity (L3_DP[0:7]) signals is optional and depends on the SRAMs selected for use with the MPC7450. Note that the least-significant bit of the L3 address (L3_ADDR[17:0]) is identified as bit 0 and that the most-significant bit is identified as bit 17. Note that for the MPC7457, there is an additional signal (L3_ADDR[18:0]), so the most-significant bit is identified as bit 18. See Section 3.7, "L3 Cache Interface," for more information on the operation of the L3 interface and the interactions of these signals. These L3 cache signals are not present on the MPC7441, MPC7445, MPC7447, MPC7447A, or MPC7448.

8.4.1.1 L3 Address (L3_ADDR[17:0])—Output

Following are the state meaning and timing comments for the L3 address output signals. For the MPC7457 there is one more output signal (L3_ADDR[18:0]).

State Meaning Asserted/Negated—Represents the address of the data to be transferred to the L3 cache. The L3 address bus is configured with bit 0 as the least-significant bit. The L3 address signals reflect the real address for various L3 cache sizes as shown in Table 8-7. Note that the L3 address does not correspond bit-for-bit with the real address.

Table 8-7. L3 Address to Physical Address Bit Mapping

L3 Address Bit	Private Memory Size and Corresponding Physical Address Bit[1]		
	1M	2M	4M[2]
L3_ADDR[18][3]	—	PM[4]	17
L3_ADDR[17]	PM[5]	18	18
L3_ADDR[16]	19	19	19
L3_ADDR[15]	20	20	20
L3_ADDR[14]	21	21	21
L3_ADDR[13]	22	22	22
L3_ADDR[12]	23	23	23
L3_ADDR[11]	24	24	24
L3_ADDR[10]	25	25	25
L3_ADDR[9]	26	26	26

Table 8-7. L3 Address to Physical Address Bit Mapping (continued)

L3 Address Bit	Private Memory Size and Corresponding Physical Address Bit[1]		
	1M	2M	4M[2]
L3_ADDR[8]	27	27	27
L3_ADDR[7]	28	28	28
L3_ADDR[6]	29	29	29
L3_ADDR[5]	30	30	30
L3_ADDR[4]	16 (Way 2)	15 (Way 2)	14
L3_ADDR[3]	17 (Way 1)	16 (Way 1)	15
L3_ADDR[2]	18 (Way 0)	17 (Way 0)	16
L3_ADDR[1]	31	31	31
L3_ADDR[0]	32	32	32

[1] Entries in parentheses indicate the function of these bits when the L3 is operated in cache mode. These bits reflect which of the eight ways is being accessed. When operating in private memory mode, the bits reflect the stated bits of the physical address.

[2] MPC7457-specific; not supported on other devices. Note that because the MPC7457 does not support 4M of L3 as cache, 4M of SRAM must be configured as either 4M of private memory or 2M of private memory and 2M of cache; see the *MPC7450 RISC Microprocessor Family Reference Manual* for more information.

[3] MPC7457-specific; not implemented on other devices.

[4] For MPC7457 only, the value of this bit depends on the size of the SRAM and the private memory space. For 4M of SRAM configured as 2M of cache and 2M of private memory, this bit will be driven low (0b0) for cache accesses and high (0b1) for private memory accesses. For 2M of SRAM configured as private memory (only), this bit is always driven low (0b0).

[5] The value of this bit depends on the size of the SRAM and the private memory space. For 2M of SRAM configured as 1M of cache and 1M of private memory, this bit will be driven low (0b0) for cache accesses and high (0b1) for private memory accesses. For 1M of SRAM configured as private memory (only), this bit is always driven low (0b0).

Timing Comments Assertion/Negation—Driven valid by the MPC7450 during read and write operations; driven with static data when the L3 cache memory is not being accessed.

8.4.1.2 L3 Data (L3_DATA[0:63])

The L3 data bus (L3_DATA[0:63]) consists of 64 signals that are both input and output on the MPC7450. The L3_DATA[0:63] are tristated during system reset. These L3 cache signals are not present on the MPC7441, MPC7445, MPC7447, MPC7447A, or MPC7448.

8.4.1.2.1 L3 Data (L3_DATA[0:63])—Output

Following are the state meaning and timing comments for the L3 data output signals.

State Meaning	Asserted/Negated—Represents the state of data during a data write transaction. Data is always transferred in full double words.
Timing Comments	Assertion/Negation—Driven valid by MPC7450 during write operations; driven with static data when the L3 cache memory is not being accessed by a read operation. High Impedance—Occurs for at least one cycle when the MPC7450 transitions between read and write operations to the L3 cache memory.

8.4.1.2.2 L3 Data (L3_DATA[0:63])—Input

Following are the state meaning and timing comments for the L3 data input signals.

State Meaning	Asserted/Negated—Represents the state of data during a data read transaction. Data is always transferred in full double words. Each transaction is split up into four 2-byte data lanes. For DDR SRAM, each data lane is synchronized to the SRAM-supplied clock connected to the corresponding L3_ECHO_CLK signal. For PB2 and late-write SRAM (which do not provide a return clock), the external SRAM supplies data in 4-byte groups synchronized to a feedback loop. Two data lanes are synchronized by a loop from L3_ECHO_CLK[1] to L3_ECHO_CLK[0], and the other two data lanes are synchronized by a loop from L3_ECHO_CLK[3] to L3_ECHO_CLK[2].
Timing Comments	Assertion/Negation—Driven valid by L3 cache memory during read operations.

8.4.1.3 L3 Data Parity (L3_DP[0:7])

The eight data bus parity (L3_DP[0:7]) signals on the MPC7450 are both output and input signals.These L3 cache signals are not present on the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448.

8.4.1.3.1 L3 Data Parity (L3_DP[0:7])—Output

Following are the state meaning and timing comments for the L3 data parity signals as outputs.

State Meaning	Asserted/Negated—Represents odd parity for each of the 8 bytes of L3 cache data during write transactions. Odd parity means that an odd number of bits, including the parity bit, are driven high. L3_DP0 is associated with bits 0–7 (byte lane 0) of the L3_DATA bus.
Timing Comments	Assertion/Negation—The same as L3_DATA[0:63]. High Impedance—The same as L3_DATA[0:63].

8.4.1.3.2 L3 Data Parity (L3_DP[0:7])—Input

Following are the state meaning and timing comments for the L3 data parity signals as inputs.

State Meaning	Asserted/Negated—Represents odd parity for each byte of L3 cache read data.
Timing Comments	Assertion/Negation—Same as L3_DATA[0:63].

8.4.2 L3 Cache Clock/Control

The following sections describe the L3 clock and control signals. These L3 cache signals are not present on the MPC7441, MPC7445, MPC7447, MPC7447A, or MPC7448.

8.4.2.1 L3 Clock (L3_CLK[0:1])—Output

Following are the state meaning and timing comments for the L3_CLK[0:1] signals.

State Meaning	Asserted/Negated—Clock output for L3 cache memory devices.
Timing Comments	Assertion/Negation—Refer to the hardware specifications for timing comments.

8.4.2.2 L3 Clock Synchronization (L3_ECHO_CLK[0:3])

The four L3 clock synchronization (L3_ECHO_CLK[0:3]) signals on the MPC7450 are both input and output signals depending on the type of SRAM used.

8.4.2.2.1 L3 Clock Synchronization (L3_ECHO_CLK[1,3])—Output

With PB2 and late-write SRAM, L3_ECHO_CLK[1] and L3_ECHO_CLK[3] are used as output signals. Following are the state meaning and timing comments for the L3_ECHO_CLK[1,3] signals. Note that L3_ECHO_CLK[0,2] are inputs only and never outputs.

State Meaning	Asserted/Negated—Clock outputs for read data synchronization. One signal is provided for each external SRAM.
Timing Comments	Assertion/Negation—Occur synchronously with the L3_CLK[0:1]. Refer to the hardware specifications.

8.4.2.2.2 L3 Clock Synchronization (L3_ECHO_CLK[0:3])—Input

All four (L3_ECHO_CLK[0:3]) signals are used as inputs with MSUG2 DDR SRAM. For MSUG2 DDR, L3_ECHO_CLK[0:1] must be paired with the SRAM providing L3_DATA[0:31]/L3_DP[0:3], and L3_ECHO_CLK[2:3] is connected to the to SRAM providing L3_DATA[32:63]/L3_DP[4:7]. With PB2 and late-write SRAM, L3_ECHO_CLK[0] and L3_ECHO_CLK[2] are used as input signals. Following are the state meaning and timing comments for the L3_ECHO_CLK[0:3] signals.

State Meaning	Asserted/Negated—Clock inputs for read data synchronization. One pair of signals is provided for each external SRAM with MSUG2 DDR SRAM, one signal is provided for external SRAM with PB2 and late-write SRAM.
Timing Comments	Assertion/Negation—Occurs asynchronously with the L3_CLK[0:1] and with each SRAM. Refer to the hardware specifications.

8.4.2.3 L3 Control (L3_CNTL[0:1])

The L3_CNTL[0:1] signals can have various functionality depending on the type of SRAM. Table 8-8 provides a summary of how the L3_CNTL[0:1] signals function.

Table 8-8. Function of L3_CNTL[0:1] Signal

L3CR[L3RT] Setting	L3 SRAM Type	L3_CNTRL0	L3_CNTRL1
00	MUGS2 DDR SRAM	Load new address (L3ADS)	Write operation (L3WE)
01	Late-write SRAM	Chip enable (L3CE)	Write operation (L3WE)
10	Reserved	—	—
11	PB2 SRAM	Chip enable (L3CE)	Write operation (L3WE)

For further details on timing for: DDR SRAM refer to Section 3.7.9.1, “MSUG2 DDR Interface Timing,” late-write SRAM refer to Section 3.7.9.2, “Late-Write SRAM Timing,” and PB2 SRAM refer to Section 3.7.9.3, “Pipelined Burst SRAM.”

8.4.2.3.1 L3 Control (L3_CNTL0)—Output

Following are the state meaning and timing comments for the L3_CNTL0 signal.

State Meaning Asserted/Negated—For DDR SRAM, this indicates to load a new address. For PB2 and late-write SRAM, this indicates to enable the SRAM.

Timing Comments Assertion/Negation—Occurs synchronously with L3_CLK[0:1].

8.4.2.3.2 L3 Control (L3_CNTL1)—Output

Following are the state meaning and timing comments for the L3_CNTL1 signal.

State Meaning Asserted/Negated—For DDR SRAM, PB2, and late-write SRAM this indicates that a write operation is occurring.

Timing Comments Assertion/Negation—Occurs synchronously with L3_CLK[0:1].

8.4.2.4 L3 Voltage Select (L3_VSEL)—Input

The MPC7450 provides several I/O voltages to support both compatibility with existing systems and migration to future systems. The state of the L3VSEL signal is sampled before and after HRESET negation to represent a 2-bit value that is used to select among the various supported bus voltages. See the hardware specifications for specific information on the L3VSEL signal and supported bus voltages. Following are the state meaning and timing comments for the L3VSEL signal.

State Meaning Assertion/Negation—Selects the high voltage level for all L3 interface signals. See the hardware specifications for more information.

Note that this input contains an internal pull-up resistor to ensure that an unterminated input appears as a high signal level to the test logic.

Timing Comments Assertion/Negation—Must remain asserted or negated during normal operation.

8.4.3 Interrupts/Reset Signals

Most system status signals are input signals that indicate when exceptions are received, when checkstop conditions have occurred, and when the MPC7450 must be reset. The MPC7450 generates the output signal $\overline{\text{CKSTP\_OUT}}$ when it detects a checkstop condition. For a detailed description of these signals, see Section 9.9, “Reset, Interrupt, Checkstop, and Power Management Signal Interactions.”

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

The interrupt ($\overline{\text{INT}}$) signal is an input signal on the MPC7450. Following are the state meaning and timing comments for the $\overline{\text{INT}}$ signal.

State Meaning	Asserted—Indicates that the MPC7450 should take an external interrupt if enabled in the MSR. Refer to Chapter 4, “Exceptions,” for more information on interrupt operation and interrupt vector assignments. Negated—Indicates that the interrupt is not being requested.
Timing Comments	Assertion—May occur at any time asynchronously to SYSCLK; the $\overline{\text{INT}}$ input is level-sensitive. Negation—Must not occur until after the interrupt is taken.

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

The system management interrupt ($\overline{\text{SMI}}$) signal is an input signal on the MPC7450. Following are the state meaning and timing comments for the $\overline{\text{SMI}}$ signal.

State Meaning	Asserted—Indicates that the MPC7450 should take a system management interrupt if enabled in the MSR. Refer to Chapter 4, “Exceptions,” for more information on interrupt operation and interrupt vector assignments. Negated—Indicates that the interrupt is not being requested.
Timing Comments	Assertion—May occur at any time asynchronously to SYSCLK; The $\overline{\text{SMI}}$ input is level-sensitive. Negation—Must not occur until after the interrupt is taken.

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

The machine check ($\overline{\text{MCP}}$) signal is an input signal on the MPC7450. Following are the state meaning and timing comments for the $\overline{\text{MCP}}$ signal.

State Meaning	Asserted—Indicates that the MPC7450 should initiate a machine check interrupt or enter the checkstop state as directed by the MSR. Negated—Indicates that machine check handling is not being requested.
Timing Comments	Assertion—May occur at any time asynchronously to SYSCLK; the $\overline{\text{MCP}}$ input is negative edge sensitive. Negation—May occur any time after the minimum $\overline{\text{MCP}}$ pulse width of two bus clocks has been met; see the hardware specifications.

8.4.3.4 Reset Signals

There are two reset signals on the MPC7450—hard reset ($\overline{\text{HRESET}}$) and soft reset ($\overline{\text{SRESET}}$) and they are described in the following subsections.

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

The soft reset input provides a “warm” reset capability. It functions as a non-maskable interrupt. 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 4.6.1, “System Reset Exception (0x00100).”
	Negated—Indicates that normal operation should proceed. See Section 9.9.1, “Reset Inputs.”
Timing Comments	Assertion—May occur at any time and may be asserted asynchronously to the MPC7450 input clock SYSCLK. The $\overline{\text{SRESET}}$ input is negative edge sensitive.
	Negation—May be negated two bus cycles after assertion.

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

The hard reset ($\overline{\text{HRESET}}$) signal must be used at power-on to properly reset the processor. The hard reset sequence includes the hardware initialization of the MPC7450’s circuitry. 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 negated to asserted. Causes a reset exception as described in Section 4.6.1, “System Reset Exception (0x00100).” During assertion output drivers are released to high impedance and the MPC7450 is held in an initialized state.
	Negated—Indicates that normal operation should proceed as defined by Section 4.6.1, “System Reset Exception (0x00100).”
Timing Comments	Assertion—May occur at any time and may be asserted asynchronously to the MPC7450 input clock SYSCLK; must be held asserted for a minimum of 255 clock cycles after the PLL lock time has been met. Refer to the hardware specifications for further timing comments.
	Negation—May occur any time after the minimum reset pulse width has been met.

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

Following are the state meaning and timing comments for the $\overline{\text{CKSTP\_IN}}$ signal.

State Meaning	Asserted—Indicates that the MPC7450 must terminate operation by internally gating off all clocks and releasing 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. $\overline{\text{CKSTP\_IN}}$ is not maskable.
	Negated—Indicates that normal operation should proceed. See Section 9.9.3, “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 CKSTP_OUT output signal has been asserted.

8.4.3.6 Checkstop Output (CKSTP_OUT)—Output

Note that the CKSTP_OUT signal is an open-drain type output and requires an external pull-up resistor (for example, 10 kΩ to V_{DD}) to assure proper negation. Following are the state meaning and timing comments for the CKSTP_OUT signal.

State Meaning	Asserted—Indicates that the MPC7450 has detected a checkstop condition and has ceased operation. Negated—Indicates that the MPC7450 is operating normally. See Section 9.9.3, “Checkstops.”
Timing Comments	Assertion—May occur at any time and may be asserted asynchronously to the MPC7450 input clocks. High Impedance— Occurs upon assertion of HRESET.

8.4.4 Processor Status/Control Signals

Processor status signals indicate the state of the processor. This includes the time base enable signal and machine quiesce control signals.

8.4.4.1 Timebase Enable (TBEN)—Input

The timebase enable (TBEN) signal is an input signal on the MPC7450. Following are the state meaning and timing comments for the TBEN signal. Note that in addition to the assertion of the TBEN signal, HID0[TBEN] must also be set in order for the time base and decrementer to operate.

State Meaning	Asserted—Indicates that the timebase and decrementer should continue clocking. This signal functions as a count enable control for the timebase and decrementer counter. Negated—Indicates that the timebase and decrementer should stop clocking.
Timing Comments	Assertion/Negation—May occur at any time asynchronously to SYSCLK

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

The quiescent request ($\overline{\text{QREQ}}$) signal is an output signal on the MPC7450. See Chapter 10, “Power and Thermal Management,” for more information about the power management modes of the MPC7450. Following are the state meaning and timing comments for the $\overline{\text{QREQ}}$ signal.

State Meaning	Asserted—Indicates that the MPC7450 is requesting all bus activity to terminate or pause so that it may enter a quiescent (low-power nap or sleep) state. Once in this state, the MPC7450 stops snooping further bus activity**.** Negated—Indicates that the MPC7450 is not requesting to enter nap or sleep mode.
Timing Comments	Assertion/Negation—May occur on any cycle. $\overline{\text{QREQ}}$ remains asserted for the duration of the nap or sleep mode.

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

The quiescent acknowledge ($\overline{\text{QACK}}$) signal is an input signal on the MPC7450. See Chapter 10, “Power and Thermal Management,” for more information about the power management modes of the MPC7450. Following are the state meaning and timing comments for the $\overline{\text{QACK}}$ signal.

State Meaning	Asserted—Indicates that all bus activity has terminated or paused and that the MPC7450 may enter nap or sleep mode. Negated—Indicates that the MPC7450 may not enter nap or sleep mode or that it must return to doze mode from nap mode in order to snoop.
Timing Comments	Assertion/Negation—May occur on any cycle following the assertion of $\overline{\text{QREQ}}$; 20 processor cycles after $\overline{\text{QACK}}$ assertion, a $\overline{\text{QACK}}$ negation for at least eight cycles ensures that the MPC7450 has returned to doze mode from nap mode. Refer to Figure 10-1 for a state diagram of the power states.

8.4.4.4 Bus Voltage Select (BVSEL)—Input

The MPC7450 provides several I/O voltages to support both compatibility with existing systems and migration to future systems. The state of the BVSEL signal is sampled before and after $\overline{\text{HRESET}}$ negation to represent a 2-bit value that is used to select among the various supported bus voltages. See the hardware specifications for specific information on the BVSEL signal and supported bus voltages. Following are the state meaning and timing comments for the BVSEL signal.

State Meaning	Asserted/Negated—Sampled before and after $\overline{\text{HRESET}}$ negation to select bus voltage
Timing Comments	
	Assertion/Negation—Must remain asserted or negated during normal operation. Note that this input contains an internal pull-up resistor to ensure that an unterminated input appears as a high signal level to the test logic.

8.4.4.5 BVSEL[0:1] (MPC7448 Specific)

The MPC7448 implements two bus voltage select signals, BVSEL[0:1]. The voltage values are sampled once on reset to select among the various supported bus voltages. See the hardware specifications for specific information on the BVSEL[0:1] signals and supported bus voltages. Following are the state meaning and timing comments for the BVSEL[0:1] signals.

State Meaning Asserted/Negated—Sampled at $\overline{\text{HRESET}}$ negation to select bus voltage

Timing Comments

Assertion/Negation—Must remain asserted or negated during normal operation.

These inputs contain an internal pull-up resistor to ensure that an unterminated input appears as a high signal level to the test logic.

8.4.4.6 DFS Divide-by-Two and Divide-by-Four ($\overline{\text{DFS2}}$ and $\overline{\text{DFS4}}$) (MPC7448-Specific)

In the MPC7448, dynamic frequency switching (DFS) divide-by-two and divide-by-four modes can be controlled by software through the DFS2 and DFS4 bits in HID1, and by hardware through the DFS2 and DFS4 signals. Refer to Section 2.2.5.2, “Hardware Implementation-Dependent Register 1 (HID1),” for more information. Table 8-9 shows the frequency definitions determined by $\overline{\text{DFS2}}$ and $\overline{\text{DFS4}}$.

Table 8-9. MPC7448 DFS Selection

$\overline{\text{DFS2}}$ Value	$\overline{\text{DFS4}}$ Value	Definition
0	0	Full frequency
1	0	1/4 frequency
0	1	1/2 frequency
1	1	DFS is software-controlled by writes to HID1

8.4.4.7 Low Voltage RAM ($\overline{\text{LVRAM}}$) (MPC7448-Specific)

Table 8-10 shows the mode definitions determined by $\overline{\text{LVRAM}}$, a signal provided in the MPC7448 to allow L2 operation at low core voltages. Refer to the *MPC7448 RISC Microprocessor Hardware Specifications* for specific information on the $\overline{\text{LVRAM}}$ signal.

Table 8-10. MPC7448 $\overline{\text{LVRAM}}$ Selection

$\overline{\text{LVRAM}}$ Value	Definition
0	LVRAM mode enabled
1	Enabling of LVRAM mode is software-controlled by L2CR[LVRAME]

8.4.4.8 Bus Mode Select ($\overline{\text{BMODE}}$[0:1])

The $\overline{\text{BMODE}}$[0:1] signals are used during $\overline{\text{HRESET}}$ assertion to select the bus interface mode. They are also sampled after $\overline{\text{HRESET}}$ negation to configure the MPC7450's address bus driven mode and processor identification.

Table 8-11 shows the configuration settings for $\overline{\text{BMODE}}$[0:1].

Table 8-11. BMODE Configuration

Signals		At $\overline{\text{HRESET}}$ Negation (Bit Values)		Bus Mode	After $\overline{\text{HRESET}}$ Negation	
$\overline{\text{BMODE0}}$	$\overline{\text{BMODE1}}$	$\overline{\text{BMODE0}}$	$\overline{\text{BMODE1}}$		ABD Mode Selected?	Processor ID
0	0	1	1	Reserved	Y	1
$\overline{\text{HRESET}}$	0	1	1	Reserved	N	1
¬$\overline{\text{HRESET}}$[1]	0	0	1	Reserved	Y	1
1	0	0	1	Reserved	N	1
0	$\overline{\text{HRESET}}$	1	1	Reserved	Y	0
$\overline{\text{HRESET}}$	$\overline{\text{HRESET}}$	1	1	Reserved	N	0
¬$\overline{\text{HRESET}}$	$\overline{\text{HRESET}}$	0	1	Reserved	Y	0
1	$\overline{\text{HRESET}}$	0	1	Reserved	N	0
0	¬$\overline{\text{HRESET}}$	1	0	MPX	Y	1
$\overline{\text{HRESET}}$	¬$\overline{\text{HRESET}}$	1	0	MPX	N	1
¬$\overline{\text{HRESET}}$	¬$\overline{\text{HRESET}}$	0	0	60x	Y	1
1	¬$\overline{\text{HRESET}}$	0	0	60x	N	1
0	1	1	0	MPX	Y	0
$\overline{\text{HRESET}}$	1	1	0	MPX	N	0
¬$\overline{\text{HRESET}}$	1	0	0	60x	Y	0
1	1	0	0	60x	N	0

[1] ¬$\overline{\text{HRESET}}$ is the inverse of $\overline{\text{HRESET}}$.

8.4.4.8.1 Bus Selection Mode ($\overline{\text{BMODE0}}$)—Input During $\overline{\text{HRESET}}$

If the $\overline{\text{BMODE0}}$ input signal is sampled asserted at $\overline{\text{HRESET}}$ negation, the MPX bus mode will be selected. However if the $\overline{\text{BMODE0}}$ input signal is negated at $\overline{\text{HRESET}}$ negation, the 60x bus mode is selected.

Following are the state meaning and timing comments for the $\overline{\text{BMODE0}}$ signals during $\overline{\text{HRESET}}$.

State Meaning Asserted—Sampled at $\overline{\text{HRESET}}$ negation to select the bus mode. If $\overline{\text{BMODE0}}$ is asserted at $\overline{\text{HRESET}}$ negation, MPX bus mode is selected. The state of $\overline{\text{BMODE0}}$ sampled at $\overline{\text{HRESET}}$ negation is stored and readable from the BMODE0 bit in MSSCR0. The state of MSSCR0[BMODE] is active high, meaning that if $\overline{\text{BMODE0}}$ is detected as asserted at the negation of $\overline{\text{HRESET}}$, MSSCR0[BMODE] = 1. Section 9.2, “MPX Bus Protocol,” describes the MPX bus mode operation on the MPC7450.

Negated—If $\overline{\text{BMODE0}}$ is detected as negated at the negation of $\overline{\text{HRESET}}$, 60x bus mode is selected. Additionally, if $\overline{\text{BMODE0}}$ remains negated after $\overline{\text{HRESET}}$ negation (in MPX bus mode), then the address bus driven mode is not selected.

Timing Comments Assertion/Negation—May be tied high to select 60x bus interface operation; may be tied to $\overline{\text{HRESET}}$ to select MPX bus interface operation (without address bus driven mode); may be tied low to select MPX bus plus address bus driven mode.

8.4.4.8.2 Address Bus Driven Mode ($\overline{\text{BMODE0}}$)—Input After $\overline{\text{HRESET}}$

When the $\overline{\text{BMODE0}}$ input signal is sampled asserted *after* $\overline{\text{HRESET}}$ is negated, then address bus driven mode is selected. If $\overline{\text{BMODE0}}$ is detected negated after $\overline{\text{HRESET}}$ is negated, then normal address bus driving mode (address bus not always driven) is selected.

This mode modifies the time that the address and attributes signals are actively driven. The address bus driven mode is stored and readable from the MSSCR0[ABD] bit. For MPX bus mode, see Section 9.3.2.1, “Address Bus Driven Mode,” for more information and for 60x bus mode see Section 9.6.2.1, “60x Address Bus Driven Mode.”

Following are the state meaning and timing comments for the $\overline{\text{BMODE0}}$ signal.

State Meaning Asserted—Sampled after $\overline{\text{HRESET}}$ negation, the assertion of $\overline{\text{BMODE0}}$ selects address bus driven mode. Address bus grant driven mode causes the MPC7450 to drive the address bus during every cycle after a qualified bus grant is sampled, and is independent of whether the MPC7450 has a bus transaction to run or not.

Negated—If $\overline{\text{BMODE0}}$ is negated after the negation of $\overline{\text{HRESET}}$, then the address bus driven mode is not selected. The MPC7450 drives the address bus from $\overline{\text{TS}}$ through $\overline{\text{AACK}}$.

Timing Comments Assertion/Negation—

–May be tied low to indicate a value of 1 on bit 0 (most significant bit) of the bus mode and address bus driven mode.

–May be tied to $\overline{\text{HRESET}}$ to indicate a value of 1 on bit 0 of the bus mode and normal address drive mode.

–May be tied to the inverse of $\overline{\text{HRESET}}$ to indicate a value of 0 on bit 0 of the bus mode and address bus driven mode.

–May be tied high to indicate a value of 0 on bit 0 of the bus mode and normal address drive mode.

8.4.4.8.3 Bus Selection Mode ($\overline{\text{BMODE1}}$)—Input During $\overline{\text{HRESET}}$

The $\overline{\text{BMODE1}}$ signal is an input signal on the MPC7450. It must be sampled negated during $\overline{\text{HRESET}}$ negation for a bus mode to be selected.

State Meaning Asserted—If $\overline{\text{BMODE1}}$ is detected as asserted at the negation of $\overline{\text{HRESET}}$, reserved bus mode is selected. This is for factory use only.

Negated—If BMODE1 is detected as negated at the negation of HRESET, bus selection is enabled. Depending on the setting of BMODE0, the MPC7450 will function in either MPX bus or 60x bus mode.

Timing Comments Assertion/Negation—

–May not be tied low.

–May not be tied to HRESET.

–May be tied to the inverse of HRESET to indicate a value of 0 on bit 1 of the bus mode and that this MPC7450 ID is 1.

–May be tied high to indicate a value of 0 on bit 1 of the bus mode and that this MPC7450 ID is 0.

8.4.4.8.4 Bus Selection Mode (BMODE1)—Input After HRESET

When the BMODE1 input signal is sampled asserted *after* HRESET is negated, the ID bit in the MSSCR0 can be used to denote one processor as the master in a multiprocessor system. Most multiprocessor systems accomplish this through software and do not need the ID bit.

State Meaning Asserted—If BMODE1 is detected as asserted after HRESET negation, MSSCR0[ID] (bit 26) will be set.

Negated—If BMODE1 is detected as negated after HRESET negation, MSSCR0[ID] (bit 26) will not be set.

Timing Comments Assertion/Negation—

–May not be tied low.

–May be tied high to indicate a value of 0 on bit 1 of the bus mode and that this MPC7450 ID is 0.

–May not be tied to HRESET.

–May be tied to the inverse of HRESET to indicate a value of 0 on bit 1 of the bus mode and that this MPC7450 ID is 1.

8.4.4.9 Performance Monitor In (PMON_IN)—Input

The enhanced mode (PMON_IN) signal is an input signal on the MPC7450. Following are the state meaning and timing comments for the PMON_IN signal.

State Meaning Asserted—Indicates that the performance monitor will log a performance monitor pin event, as described in PMC1, bit 7 in Table 11-9 (an event will only be logged if this event is enabled in the performance monitor control registers).

Negated— Indicates that the performance monitor will not log a performance monitor pin event.

Timing Comments Assertion— May occur at any time asynchronously to SYSCLK; it is falling-edge activated.

Negation— May occur at any time after the minimum PMON_IN pulse width of two bus clocks has been met.

8.4.4.10 Performance Monitor Out ($\overline{\text{PMON\_OUT}}$)—Output

The enhanced mode ($\overline{\text{PMON\_OUT}}$) signal is an output signal on the MPC7450. Following are the state meaning and timing comments for the $\overline{\text{PMON\_OUT}}$ signal.

State Meaning	Asserted—Indicates that the performance monitor threshold or negative counter condition has been reached. Negated—Indicates that the performance monitor is either not active or the programmed threshold or negative counter condition has not been reached.
Timing Comments	Assertion— May occur at any time. Note that this pin is not affected by MMCR0[5].

8.4.5 Clock Control Signals

The MPC7450 clock signal inputs 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 hardware specifications for the exact timing relationships of the clock signals and other signals.

8.4.5.1 System Clock (SYSCLK)—Input

The MPC7450 requires a single system clock (SYSCLK) input. This input sets the frequency of operation for the bus interface. Internally, the MPC7450 uses a phase-locked loop (PLL) circuit to generate a master clock for all 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 of the SYSCLK frequency as defined in the hardware specifications, allowing the CPU core to operate at an equal or greater frequency than the bus interface.

Following are the state meaning and timing comments for the SYSCLK signals.

State Meaning	Asserted/Negated—The SYSCLK input is the primary clock input for the MPC7450 and represents the bus clock frequency for MPC7450 bus operation. Internally, the MPC7450 may be operating at an integer or half-integer multiple of the bus clock frequency.
Timing Comments	Duty cycle—Refer to the hardware specifications for timing comments and supported ratios. Loose duty cycle is allowed. 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.

8.4.5.2 PLL Configuration (PLL_CFG[0:4])—Input

The PLL (phase-locked loop) is configured by the PLL_CFG[0:4] signals. The MPC7448 has a sixth PLL configuration signal, PLL_CFG[5]. For a given SYSCLK (bus) frequency, the PLL configuration signals set the internal CPU frequency of operation. See the hardware specifications for PLL configuration information.

Following are the state meaning and timing comments for the PLL_CFG signals.

State Meaning	Asserted/Negated—Configure the operation of the PLL and the internal processor clock frequency. Settings are based on the desired bus frequency 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 the PC[0–4] bits in the HID1 register.

8.4.5.3 PLL_CFG[5] (MPC7448-Specific)

The MPC7448 has a sixth PLL configuration signal, PLL_CFG[5], formerly a factory test signal. The setting of the pin is reflected in the PC5 bit in HID1. Refer to Section 2.2.5.2, “Hardware Implementation-Dependent Register 1 (HID1),” for more information on the PC5 bit.

Following are the state meaning and timing comments for the PLL_CFG[5] signal.

State Meaning	Asserted/Negated—Configure the operation of the PLL and the internal processor clock frequency. Setting is based on the desired bus frequency 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. This bit may be read through the PC[5] bit in the HID1 register.

8.4.5.4 Extension Qualifier (EXT_QUAL)—Input

The extension qualifier (EXT_QUAL) signal is an input signal on the MPC7450. Following are the state meaning and timing comments for the EXT_QUAL signal.

State Meaning	Asserted/Negated—Provides a PLL bypass mode for the system clock. Refer to the hardware specifications for more information.
Timing Comments	Assertion/Negation—Must be set low by the system during reset and normal operation.

8.4.5.5 Clock Out (CLK_OUT)—Output

The clock out (CLK_OUT) signal is an output signal (output-only) on the MPC7450. Following are the state meaning and timing comments for the CLK_OUT signal.

State Meaning	Asserted/Negated—Provides a PLL clock output for PLL testing and monitoring. The configuration of the HID1[ECLK] and HID1[BCLK] bits determines whether the CLK_OUT signal clocks at the processor clock frequency, the bus clock frequency, or half of the bus clock frequency. See Table 2-8 for the HID1 register configuration of the CLK_OUT signal. The CLK_OUT signal defaults to a high-impedance state following the assertion of $\overline{\text{HRESET}}$. The CLK_OUT signal is provided for testing only.
Timing Comments	Assertion/Negation—During normal operation, CLK_OUT is driven as specified by HID1(BCLK) and HID1(ECLK).

8.4.6 IEEE 1149.1a-1993 (JTAG) Interface Description

The MPC7450 has five dedicated JTAG signals which are described in Table 8-12. The test data input (TDI) and test data output (TDO) scan ports are used to scan instructions as well as data into the various scan registers for JTAG operations. The scan operation is controlled by the test access port (TAP) controller which in turn is controlled by the test mode select (TMS) input sequence. The scan data is latched in at the rising edge of test clock (TCK).

Table 8-12. IEEE Interface Pin Descriptions

Signal Name	Input/Output	Weak Pull-up Provided	IEEE 1149.1a Function
TCK	Input	No	Scan clock
TDI	Input	Yes	Serial scan input signal
TDO	Output	No	Serial scan output signal
TMS	Input	Yes	TAP controller mode signal
$\overline{\text{TRST}}$	Input	Yes	TAP controller reset

Test reset ($\overline{\text{TRST}}$) is an optional JTAG signal which is used in the MPC7450 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 device. It is recommended that $\overline{\text{TRST}}$ be asserted and negated coincident with the assertion of the $\overline{\text{HRESET}}$ signal.

These signals are not used during normal operation. TMS, TDI, and $\overline{\text{TRST}}$ have internal pull-up resistors provided; TCK does not. For normal operation, TMS and TDI may be left unconnected, and TCK must be set high or low. $\overline{\text{TRST}}$ must be asserted sometime during power-up for JTAG logic initialization. Note that if $\overline{\text{TRST}}$ is tied low, unnecessary power is consumed.

8.4.6.1 JTAG Test Clock (TCK)—Input

The JTAG test clock (TCK) signal is an input on the MPC7450. Following is the state meaning for the TCK input signal.

State Meaning Asserted/Negated—This input should be driven by a free-running clock signal. Input signals to the test access port are clocked-in on the rising edge of TCK. Changes to the test access port output signals occur on the falling edge of TCK. The test logic allows TCK to be stopped.

8.4.6.2 JTAG Test Data Input (TDI)—Input

Following is the state meaning for the TDI input signal.

State Meaning Asserted/Negated—The value presented on this signal on the rising edge of TCK is clocked into the selected JTAG test instruction or data register.

Note that this input contains an internal pull-up resistor to ensure that an unterminated input appears as a high signal level to the test logic.

8.4.6.3 JTAG Test Data Output (TDO)—Output

The JTAG test data output signal is an output on the MPC7450. Following is the state meaning for the TDO output signal.

State Meaning Asserted/Negated—The contents of the selected internal instruction or data register are shifted out onto this signal on the falling edge of TCK. The TDO signal remains in a high-impedance state except when scanning of data is in progress.

8.4.6.4 JTAG Test Mode Select (TMS)—Input

The test mode select (TMS) signal is an input on the MPC7450. Following is the state meaning for the TMS input signal.

State Meaning Asserted/Negated—This signal is decoded by the internal JTAG TAP controller to distinguish the primary operation of the test support circuitry.

Note that this input contains an internal pull-up resistor to ensure that an unterminated input appears as a high signal level to the test logic.

8.4.6.5 JTAG Test Reset ($\overline{\text{TRST}}$)—Input

The test reset ($\overline{\text{TRST}}$) signal is an input on the MPC7450. Following is the state meaning for the $\overline{\text{TRST}}$ input signal.

State Meaning Asserted—This input causes asynchronous initialization of the internal JTAG test access port controller. Note that the signal must be asserted during the assertion of $\overline{\text{HRESET}}$ in order to properly initialize the JTAG test access port. The $\overline{\text{TRST}}$ signal must be asserted to properly initialize the boundary scan chain. This may be accomplished by connecting it to $\overline{\text{HRESET}}$, using logic to OR any external JTAG $\overline{\text{TRST}}$ drivers.

Negated—Indicates normal operation.

Note that this input contains an internal pull-up resistor to ensure that an unterminated input appears as a high signal level (negated) to the test logic.

8.4.7 Configuration Signals Sampled at Reset

Table 8-13 contains a description of the signals sampled for configuration at the negation of $\overline{\text{HRESET}}$. Note that throughout this manual, the reset configuration signals are described as being sampled at the negation of reset. However, the reset configuration signals are actually sampled 3 clock cycles before the negation of $\overline{\text{HRESET}}$. For more information about the timing requirements of these configuration signals relative to the negation of the reset signals, refer to the hardware specifications.

The values on these signals during reset are interpreted to be logic one or zero, regardless of whether the signal name is defined as active-low. The BVSEL and L3_VSEL signals have internal pull-up resistors so that if the signals are not driven, the default value is high (a one), as shown in the table. Please refer to the hardware specifications for information about the required drive strength to override these internal resistors. The PLL_CFG[0:4], and $\overline{\text{BMODE}}$[0:1] signals do not have pull-up resistors and must be driven high or low during the reset period.

Table 8-13. MPC7450 Reset Configuration Signals

Signal Name(s)	Default	State Meaning
$\overline{\text{BMODE}}$[0:1]	Must be driven	These two signals select the bus mode, and configure address bus driven mode and the processor identification. See Table 8-11 for $\overline{\text{BMODE}}$ configuration settings.
PLL_CFG[0:4]	Must be driven	These five signals select the clock frequency ratios used by the PLL of the MPC7450. The hardware specifications list the supported settings and provides more detailed information on the clock frequencies.
BVSEL	1	This signal configures the bus voltage.
L3_VSEL[1]	1	This signal configures the L3 cache bus voltage.

[1] Note that the L3 cache is not supported on the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448.

8.4.8 Power and Ground Signals

The MPC7450 provides the following signal connections for power and ground:

- V_{DD}—Supply voltage connection for the processor core
- VDD_SENSE—On the MPC7447A and the MPC7448, these are internally connected to V_{DD} and are intended to allow an external device to detect the processor core voltage level present inside the device package. If unused, they must be connected directly to V_{DD} or left unconnected.
- OV_{DD}—Supply voltage connection for the system interface drivers
- OVDD_SENSE—On the MPC7447A and the MPC7448, these are internally connected to V_{DD} and are intended to allow an external device to detect the processor core voltage level present inside the device package. If unused, they must be connected directly to V_{DD} or left unconnected.
- GV_{DD}—Supply voltage connection for the L3 cache interface drivers. These power supply signals are isolated from the V_{DD} and OV_{DD} power supply signals.

- AV_{DD}—Power signal provides power to the clock generation phase-locked loop. See the hardware specifications for information on how to use this signal.
- GND—Connection for grounding the MPC7450
- GND_SENSE—On the MPC7447A and the MPC7448, these are internally connected to GND and are intended to allow an external device to detect the processor ground voltage level present inside the device package. If unused, they must be connected directly to GND or left unconnected.

See the hardware specifications for detailed electrical and mechanical information for each signal.

Chapter 9
System Interface Operation

This chapter describes the MPC7450 microprocessor bus interface and its operation. It shows how the MPC7450 signals, defined in Chapter 8, “Signal Descriptions,” interact to perform address and data transfers.

9.1 MPC7450 System Interface Overview

The MPC7450 supports two interface protocols—MPX bus protocol and a subset of the 60x bus protocol. Note that although the 60x bus protocol is implemented by the MPC603e, MPC604e, MPC740, and MPC750 processors, it is referred to as the 60x bus interface. The MPX bus protocol is derived from the 60x bus protocol. The MPX bus interface includes several additional features that provide higher memory bandwidth than the 60x bus and more efficient use of the system bus in a multiprocessing environment.

The MPC7450 bus interface includes a 64-bit data bus with 8 bits of data parity, a 36-bit address bus with 5 bits of address parity, and additional control signals to allow for unique system level optimizations.

The bus interface protocol is configured using the $\overline{\text{BMODE0}}$ configuration signal at reset. If $\overline{\text{BMODE0}}$ is asserted at the negation of $\overline{\text{HRESET}}$, the MPC7450 uses the MPX bus protocol; if $\overline{\text{BMODE0}}$ is negated during the negation of $\overline{\text{HRESET}}$, the MPC7450 uses a limited subset of the 60x bus protocol. Note that the inverse state of $\overline{\text{BMODE}}$[0:1] at the negation of $\overline{\text{HRESET}}$ is saved in MSSCR0[BMODE].

9.1.1 MPC7450 Bus Operation Features

The MPC7450 has a separate address and data bus, each with its own set of arbitration and control signals. This allows for decoupling the data tenure from the address tenure of a transaction and provides for a wide range of system-bus implementations including:

- Nonpipelined bus operation
- Pipelined bus operation
- Split transaction operation

The MPC7450 supports only the normal memory-mapped address segments defined in the PowerPC architecture. Access to direct store segments results in a DSI exception.

9.1.1.1 MPX Bus Features

The MPX bus has the following features:

- Extended 36-bit address bus plus 5 bits of odd parity (41 bits total)
- 64-bit data bus plus 8 bits of odd parity (72 bits total); a 32-bit data bus mode is not supported
- Support for a four-state (MESI) cache coherence protocol
- On-chip snooping to maintain L1 data cache, L2, and L3 cache coherency for multiprocessing applications and DMA environments
- Support for address-only transfers (useful for a variety of broadcast operations in multiprocessor applications)
- Address pipelining
- Support for up to 16 out of order transactions using four data transaction index (DTI[0:3]) signals
- Full data streaming
- Support for data intervention in multiprocessor systems

9.1.1.2 60x Bus Features

The following list summarizes the 60x bus interface features:

- Extended 36-bit address bus plus 5 bits of odd parity (41 bits total)
- 64-bit data bus plus 8 bits of odd parity (72 bits total); a 32-bit data bus mode is not supported
- Support for a four-state (MESI) cache coherence protocol
- On-chip snooping to maintain L1 data cache, L2, and L3 cache coherency for multiprocessing applications and DMA environments
- Support for address-only transfers (useful for a variety of broadcast operations in multiprocessor applications)
- Address pipelining
- Support for up to 16 outstanding transactions. No re-ordering is supported.

9.1.2 Overview of System Interface Accesses

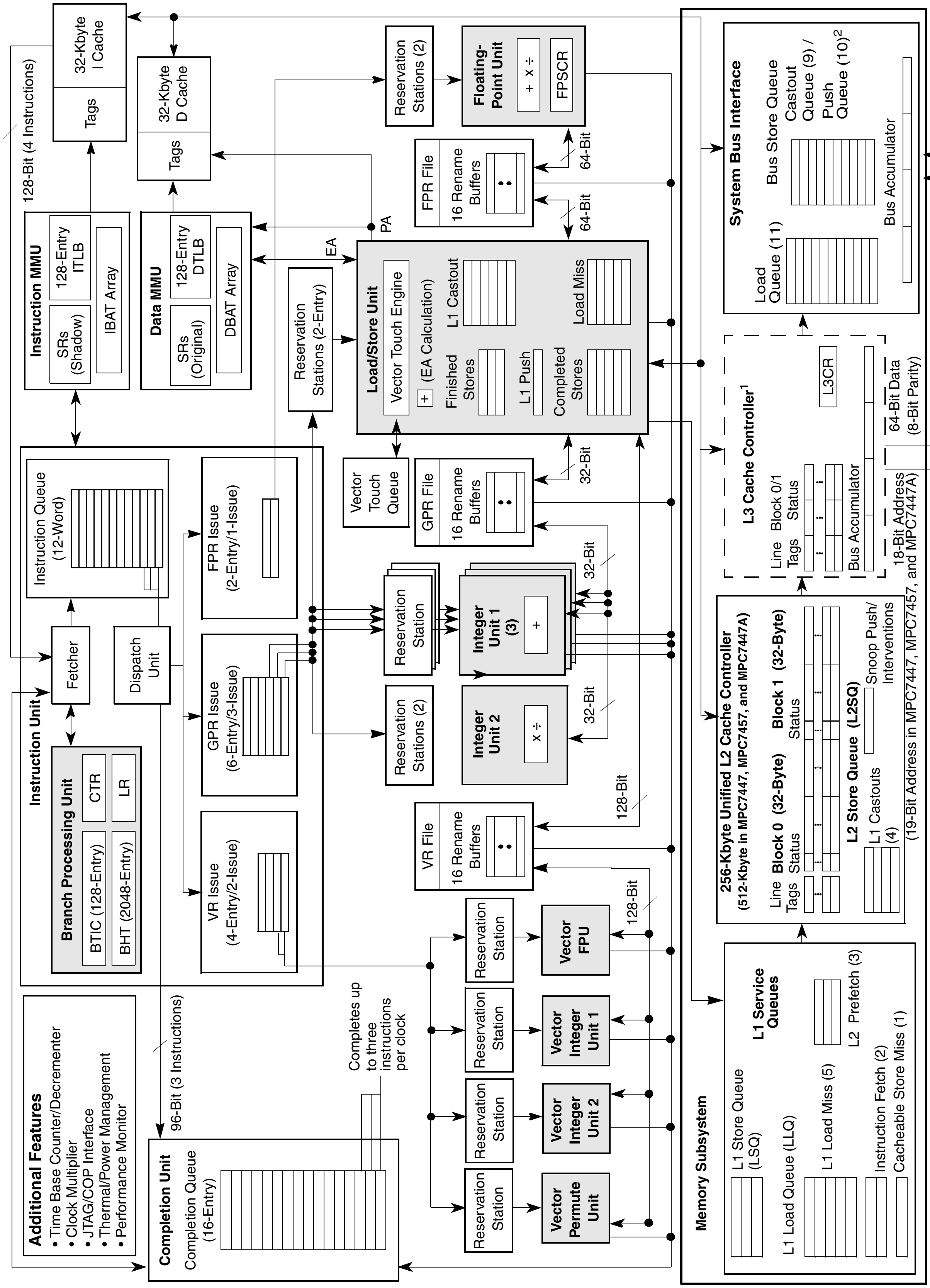
The system interface includes address register queues, prioritization logic, and a bus control unit. The system interface latches snoop addresses for snooping in the L1 data, L2, and L3 caches, the memory hierarchy address register queues, and the reservation controlled by the Load Word and Reserve Indexed (**lwarx**) and Store Word Conditional Indexed (**stwcx.**) instructions. Accesses are prioritized with load operations preceding store operations. Note that the L3 cache is not supported on the MPC7441, MPC7445, MPC7447, MPC7447A, or MPC7448.

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, floating-point, and AltiVec register files and the memory system.

When the MPC7450 encounters an instruction or data access, it calculates the effective address and uses the lower-order address bits to check for a hit in the on-chip, 32-Kbyte L1 instruction and data caches. During L1 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). The physical address bits are then compared with the corresponding cache tag bits to determine if a cache hit occurred in the L1 instruction or data cache. If the access misses in the corresponding cache, the transaction is sent to L1 load miss queue or the L1 store miss queue. L1 load miss queue transactions are sent to the internal 256-Kbyte L2 cache (512-Kbyte L2 cache for MPC7457) and L3 cache controller simultaneously. Store miss queue transactions are queued up in the L2 cache controller and sent to the L3 cache if necessary. If no match is found in the L2 or L3 cache tags, the physical address is used to access system memory.

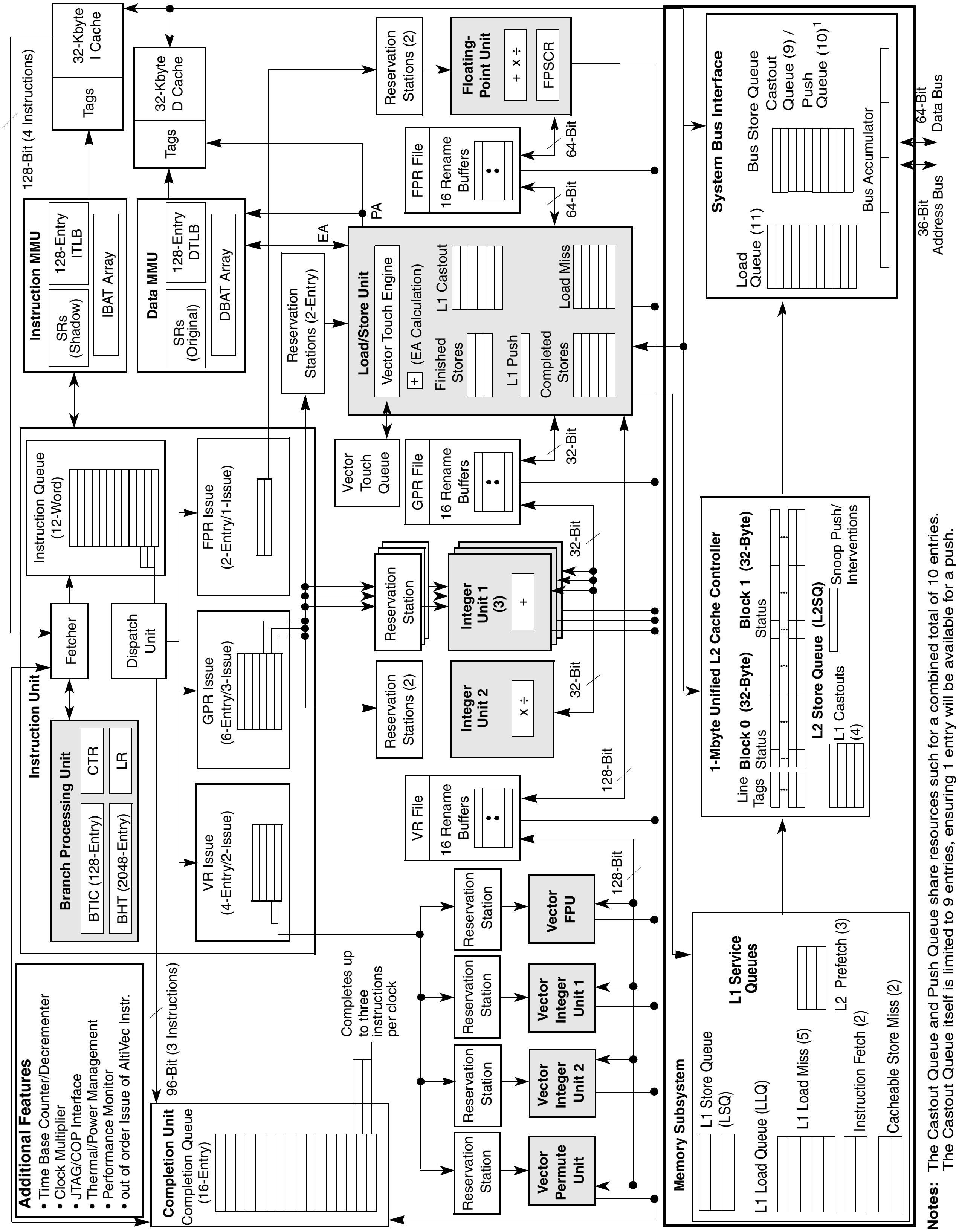
In addition to the loads, stores, and instruction fetches, the MPC7450 performs hardware table search operations following TLB misses, L1, L2, and L3 cache castout operations, and cache-line snoop push operations when a modified cache line detects a snoop hit from another bus master.

Figure 9-1 shows a block diagram of the MPC7450, including the address path from the execution units and instruction fetcher through the translation logic to the caches and system interface logic. Figure 9-2 shows the organization of the MPC7448 execution units.



Notes: 1. The L3 cache interface is not implemented on the MPC7441, MPC7445, MPC7447, or MPC7447A.
2. The Castout Queue and Push Queue share resources such for a combined total of 10 entries.
The Castout Queue itself is limited to 9 entries, ensuring 1 entry will be available for a push.

Figure 9-1. MPC7450 Microprocessor Block Diagram



Notes: The Castout Queue and Push Queue share resources such for a combined total of 10 entries. The Castout Queue itself is limited to 9 entries, ensuring 1 entry will be available for a push.

Figure 9-2. MPC7448 Microprocessor Block Diagram

The MPC7450 uses separate address and data buses and a variety of control and status signals for performing external reads and writes. The address bus is 36 bits wide and the data bus is 64 bits wide. The interface is synchronous—all MPC7450 inputs are sampled at and all outputs are driven from the rising edge of the bus clock. The processor runs at a multiple of the bus clock speed.

9.1.3 Summary of L1 Instruction and Data Cache Operation

The MPC7450 provides independent L1 instruction and data caches. Each cache is a physically-addressed, 32-Kbyte cache with 8-way set associativity. Both caches consist of 128 sets of 8 cache lines, with 32 consecutive bytes in each cache line. The MPC7450 data cache tags are dual-ported and non-blocking, allowing efficient load/store and snoop operations. The MPC7450 supports a four-state cache coherency protocol that includes Modified (M), Exclusive (E), Shared (S), and Invalid (I) data cache states.

The cache control instructions, **dcbt**, **dcbtst**, **dcbz**, **dcbst**, **dcbf**, **dcba**, **dcbi**, and **icbi**, are intended for the management of the local L1, L2, and L3 caches. The MPC7450 interprets the cache control instructions as if they pertain only to its own caches. These instructions are not intended for managing other caches in the system (except to the extent necessary to maintain coherency). The MPC7450 snoops all global ($\overline{\text{GBL}}$ asserted) cache control instruction broadcasts. Execution of the **dcbz** and **dcba** instructions cause a broadcast on the system bus (when M = 1) unless the address cache block is found in the exclusive state in the L1, L2, or L3 cache. The **dcbst**, **dcbf**, **dcbi**, and **icbi** instructions cause a broadcast on the system bus (when M = 1) only if HID1[ABE] is set. Note that the L3 cache is not supported on the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448.

Because the data cache on the MPC7450 is an on-chip, write-back primary cache, the predominant type of transaction for most applications is burst-read memory operations, followed by burst-write memory operations and single-beat (caching-inhibited 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).

On a cache miss, cache blocks are filled in one beat of 32 bytes. The burst fill is performed as a critical-double-word-first operation. For the instruction cache, the critical double word is forwarded to the instruction queue as soon as it is available, thus minimizing stalls due to cache fill latency. The instruction cache is not blocked to internal accesses while a load completes, providing for hits under misses. For the data cache, an entire cache block is collected in an accumulator latch before being loaded into the cache. The critical double word is forwarded to the execution units as soon as it is available.

Cache lines are selected for replacement based on a pseudo least-recently-used (PLRU) algorithm. Each time a cache block is accessed, it is tagged as the most recently used way of the set (unless accessed by the AltiVec LRU instructions, see Section 7.1.2.1, “LRU Instructions”). For every hit in the cache or when a new block is reloaded, the PLRU bits for the set are updated. Data cache replacement selection is performed at reload time, not when a miss occurs. However, instruction cache replacement selection occurs when an instruction cache miss is first recognized—that is, the instruction cache replacement target is selected upon a miss and not at reload.

A data cache block fill is caused by a load miss or write-back store miss in the cache. The cache block that corresponds to the missed address is updated by a burst transfer of the data from the L2 cache, L3 cache, or system memory after any necessary coherency actions have completed.

For more information about the interactions of the instruction and data caches and the system interface, see Section 3.8, “System Bus Interface.”

9.1.4 L2 Cache Overview

The MPC7450 features an integrated L2 cache that is a unified (containing instruction and data) 256 Kbyte on-chip cache. The MPC7457 has a 512 Kbyte on-chip L2 cache. It is 8-way set associative and organized with 32-byte blocks and two blocks per line. Thus each block shares the same tag, but the valid, modified, and shared bits are independently maintained for each block.

9.1.5 L3 Cache Overview

Similar in architecture to the L2 cache, the MPC7450 provides an on-chip, eight-way set associative tag memory, and a dedicated L3 cache port with support for up to 2 Mbyte of external SRAM. The L3 cache is organized with 2 or 4 blocks (sectors) per line, usually operates in write-back mode, and supports system cache coherency through snooping. Note that the L3 cache is not supported on the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448.

The L3 cache receives memory access requests from both the L1 and L2 caches. The L3 accesses are compared to the L3 cache tags and the data or instructions are forwarded from the L3 to the L1 and L2 caches if there is an L3 cache hit, or are forwarded on to the bus interface unit if there is an L3 cache miss or if the address being accessed is from a page marked as caching-inhibited. Burst read accesses that miss in the L3 cache initiate a load operation from the bus interface.

An L1 load or store operation can cause an L3 cache block allocation resulting in the castout of an L3 cache block marked modified to the bus interface. For additional information about the operation of the L3 cache, refer to Section 3.7, “L3 Cache Interface.”

9.1.6 Operation of the System Interface

Memory accesses can occur in single-beat (1, 2, 3, 4, and 8 bytes), double-beat (16 bytes), and four-beat (32 bytes) burst data transfers. For memory accesses, the address and data buses are independent to support pipelining and split transactions. The bus interface can pipeline as many as 16 transactions and, in MPX bus mode, supports full out of order split-bus transactions. The MPC7450 bursts out of reset in MPX bus mode, fetching eight instructions on the MPX bus at a time.

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 MPC7450 to be integrated into systems that implement various fairness and bus-parking procedures to avoid arbitration overhead.

Typically, memory accesses are weakly ordered to maximize the efficiency of the bus without sacrificing coherency of the data. The MPC7450 allows load operations to bypass store operations (except when a dependency exists). Because the processor can dynamically optimize run-time ordering of load/store traffic, overall performance is improved.

Note that the synchronize (**sync**) and enforce in-order execution of I/O (**eieio**) instructions can be used to enforce strong ordering.

This is a synchronous interface—all MPC7450 input signals are sampled and output signals are driven on the rising edge of the bus clock cycle (see the hardware specifications for exact timing information).

9.1.7 Memory Subsystem Control Register (MSSCR0)

The MSSCR0 control register is used to configure many aspects of the memory subsystem and bus protocols for the MPC7450. At power on reset, functions are set to a default; thus the MSSCR0 should be changed if non-default functionality is required. It is a supervisor-only read/write, implementation-specific register accessed as SPR 1014.

MSSCR0 includes parameters that set the maximum number of transactions that a MPC7450 can carry in its data transaction queue, alter how the MPC7450 responds to snoop requests, enable or disable data intervention, indicate when the MPC7450 is operating in the address bus drive mode, and indicate the state of the $\overline{\text{BMODE}}$[0:1] signals during power-on reset. There are other parameters in MSSCR0 that control L2 cache prefetching and the external L3 cache interface behavior. See Section 2.2.5.3, “Memory Subsystem Control Register (MSSCR0),” for more detailed information about the bits of MSSCR0.

9.1.8 Memory Subsystem Status Register (MSSSR0)

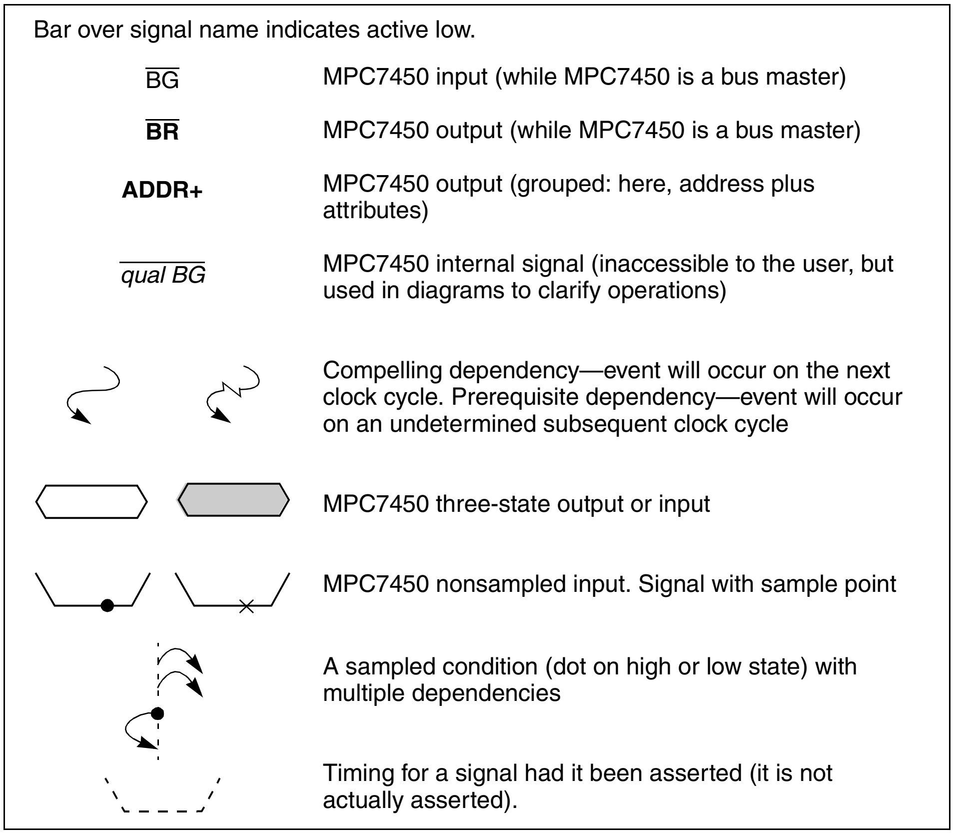
The memory subsystem status register (MSSSR0) acknowledges bus transfer errors and indicates when parity errors are detected in the L2 and L3 caches and the data or address buses. See Section 2.2.5.4, “Memory Subsystem Status Register (MSSSR0),” for more detailed information about the bits of MSSSR0.

9.1.9 Direct-Store Accesses Not Supported

The MPC7450 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. Any attempt to access instructions or data in a direct-store segment causes the MPC7450 to take an ISI or DSI exception.

9.1.10 Common Timing Diagram Symbols

The following sections describe how the MPC7450 interfaces operate, providing detailed timing diagrams that illustrate how the signals interact. In the following sections, timing diagrams are used to illustrate the bus protocols. Figure 9-3 provides a legend for symbols and typographic conventions used in the timing diagrams throughout this chapter.



Bar over signal name indicates active low.

$\overline{\text{BG}}$	MPC7450 input (while MPC7450 is a bus master)
$\overline{\textbf{BR}}$	MPC7450 output (while MPC7450 is a bus master)
ADDR+	MPC7450 output (grouped: here, address plus attributes)
$\overline{\textit{qual BG}}$	MPC7450 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
	MPC7450 three-state output or input
	MPC7450 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 9-3. Timing Diagram Legend

9.2 MPX Bus Protocol

The MPX bus protocol is based on the 60x bus protocol. It also includes several additional features that allow it to provide higher memory bandwidth than the 60x bus and more efficient utilization of the system bus in a multiprocessing environment.

Memory accesses that use the MPX bus protocol are divided into address and data tenures. Each tenure has three phases—bus arbitration, transfer, and termination. The MPX bus protocol also supports address-only transactions. Note that address and data tenures can overlap, as shown in Figure 9-4.

Figure 9-4 shows that the address and data tenures in the MPX bus protocol are distinct from one another and each tenure consists of three phases—arbitration, transfer, and termination. The separation of the address and data tenures allows advanced bus techniques—such as split-bus transactions, enveloped transactions, and pipelining—to be implemented at the system level in multiprocessor systems (see Section 9.2.1, “MPX Bus Pipelining”). Figure 9-4 shows a data transfer that consists of 1-, 2-, or 4-beat transfers. Two- and 4-beat burst transfers of 32-byte cache lines require data transfer termination signals for each beat of data.

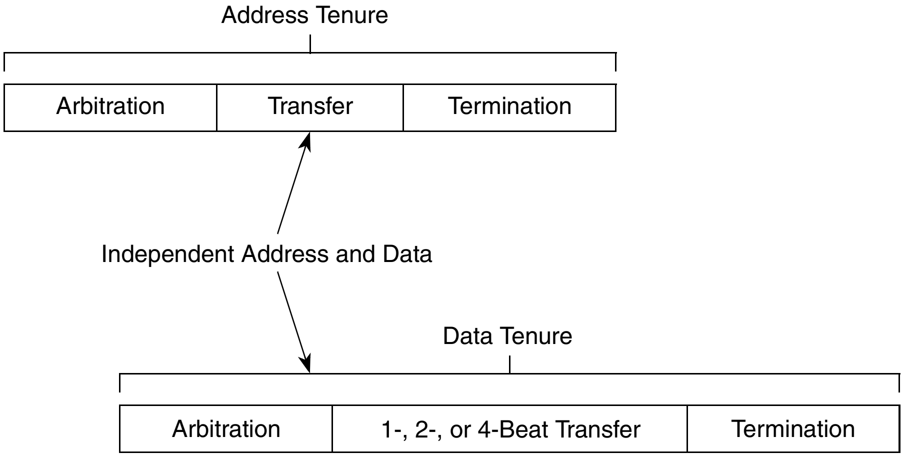


Figure 9-4. Overlapping Tenures on the MPC7450 Bus for Transfers

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 MPC7450 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 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 MPC7450 arbitrates for mastership of the data bus.
  - Transfer: After the MPC7450 is the data bus master, it samples the data bus for read operations or drives the data bus for write operations. The data parity 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.

Note that most transactions require both an address tenure followed by a data tenure. However, the MPC7450 also supports some address-only and, in the case of data intervention, data-only transactions.

Arbitration for both address and data bus mastership is performed by an external arbiter using the address arbitration signals $\overline{\text{BR}}$, and $\overline{\text{BG}}$, and the data arbitration signal $\overline{\text{DBG}}$. Most arbiter implementations require additional signals to coordinate bus master, slave, and snooping activities.

For more detailed information on the arbitration signals, refer to Section 8.2.5, “Address Bus Arbitration Signals,” and Section 8.3.6, “Data Bus Arbitration Signals.”

9.2.1 MPX Bus Pipelining

The MPX bus protocol allows the separation of address and data tenures and provides the following types of bus pipelining:

- Address pipelining
  - Enveloped transactions
  - Split-bus transactions
- Address-only transactions
- Data-only transfers

Address pipelining allows the address tenure of a new bus transaction to begin before the data tenure of the current transaction has finished. An enveloped transaction occurs when both a new address tenure and its data tenure are allowed to begin before the data tenure of a previous transaction has completed.

Split-bus transaction capability allows the data tenure for a transaction to be arbitrated for and granted independently from the address tenure. The data tenure may be granted during the address tenure or after the address tenure has completed. Additionally, the bus activity in a split transaction can be either from the same master or from different masters.

MPX also provides for address-only transactions, or transactions that utilize the address bus only (address tenure), with no data transfer involved (no data tenure). This capability is generally useful in systems where the ability to issue or receive synchronization, cache control, or TLB control commands between devices may be desirable.

The MPX protocol supports a data-only transfer in order to support cache-to-cache transfers (data intervention) and local bus slave. The $\overline{\text{HIT}}$ and $\overline{\text{DRDY}}$ signals exist in MPX to support this type of transfer.

While these capabilities do not inherently reduce memory latency, supporting them can greatly improve effective bus and memory throughput. For this reason, these techniques are most effective in shared-memory multimaster 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 external arbiter must control the pipeline depth and synchronization between masters and slaves.

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. The MPC7450 can pipeline up to 16 address tenures before starting a data tenure.

9.3 MPX Bus Address Tenure

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

An address retry capability, similar to 60x bus mode, is provided to support the snoop coherency protocol. In MPX bus mode, the MPC7450 additionally supports data intervention for more efficient coherency management. Address retry is used only when the MPC7450 cannot service the snoop, $\overline{\text{HIT}}$-style data intervention is not enabled, or in cases where $\overline{\text{HIT}}$-style data intervention is not allowed. An address retry response is issued by a snooping master in order to interrupt another master's transaction on the bus, usually to write back modified data to memory that is in its cache. The address retry causes the original master to abort the current transaction and rerun the transaction at a later time.

9.3.1 MPX Bus Address Bus Arbitration

MPX address bus arbitration differs from arbitration in the 60x bus protocol in two regards. First, the MPC7450 does not use any address bus busy (external $\overline{\text{ABB}}$ or internal *abb*) signal in generating a qualified bus grant. Second, the MPC7450 can drive consecutive address tenures without a dead cycle on the address bus if those address tenures are from the same processor. This means that to end an address tenure from a master, the system must not assert $\overline{\text{BG}}$ to a subsequent master until a cycle after the $\overline{\text{AACK}}$ for the first tenure is asserted. The transfer start ($\overline{\text{TS}}$) signal is still used to qualify bus grant in order to optimize speculative bus parking.

The elimination of $\overline{\text{ABB}}$ from the interface removes logic from critical timing paths in the processor interface, allowing higher frequency bus operation. Removal of $\overline{\text{ABB}}$ does, however, put more responsibility on the system arbiter; now it is the arbiter that must track whether the address bus is busy and not issue an address bus grant ($\overline{\text{BG}}$) to a processor when it would cause a collision on the address bus with an address tenure from another bus master.

Arbiter designs must ensure that no more than one address bus master can be granted the bus at one time (that is, bus grants must be mutually exclusive). In single-master applications, $\overline{\text{BG}}$ can effectively be tied asserted, always granting the bus to the only potential master (see Section 9.3.1.2, "MPX Address Bus Parking"). However, as explained above, $\overline{\text{BG}}$ must be negated in every cycle that the arbiter delays $\overline{\text{AACK}}$.

Note that the MPC7450 may assert $\overline{\text{BR}}$ but not use the bus even if it receives the qualified bus grant. Or it may negate $\overline{\text{BR}}$ (that is, cancel the request) before accepting a qualified bus grant. For example, if an internal data transaction queue fills up due to a $\overline{\text{HIT}}$ and there is no room for a new $\overline{\text{TS}}$, the MPC7450 will withdraw the bus request.

9.3.1.1 Qualified Bus Grant in MPX Bus Mode

When the MPC7450 needs access to the external bus and it is not parked ($\overline{\text{BG}}$ is negated), (see Section 9.3.1.2, "MPX Address Bus Parking"), the following occurs:

- The processor asserts bus request ($\overline{\text{BR}}$) to the arbiter and holds it asserted until
- A qualified bus grant is detected. The equation for a qualified MPX bus grant is as follows:
  Qualified Bus Grant = $\overline{\text{BG}}$ & ¬$\overline{\text{ARTRY}}$ & ¬$\overline{\text{TS}}$ & ¬ (latched state variables)

where latched state variables include latched $\overline{\text{ARTRY}}$ that is not asserted in the current or in the preceding cycle, and regardless of whether the master or any other processor is currently driving $\overline{\text{TS}}$. Figure 9-5 shows the non-parked case; the X's mark the values of the signals that allow a qualified bus grant. Note that $\overline{\text{ABB}}$ (as well as the internal $\overline{abb}$) is no longer in this equation and neither is address acknowledge ($\overline{\text{AACK}}$).

- The processor is granted mastership of the bus and the bus is available.

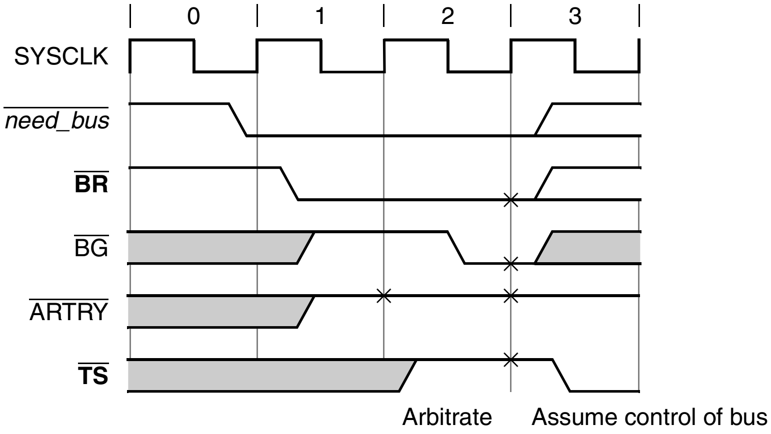


Figure 9-5. MPX Address Bus Arbitration—Non-Parked Case

9.3.1.2 MPX Address Bus Parking

External arbiters must allow only one device at a time to be the address bus master. In systems with only a single master, $\overline{\text{BG}}$ can be grounded (always asserted) to continually grant mastership of the address bus to the MPC7450. This continual granting of mastership is called bus parking.

If the MPC7450 asserts $\overline{\text{BR}}$ before the external arbiter asserts $\overline{\text{BG}}$, the MPC7450 is considered unparked, as shown in Figure 9-5. Figure 9-6 shows the parked case, where a qualified bus grant exists on the clock edge following a $\overline{need\_bus}$ condition.

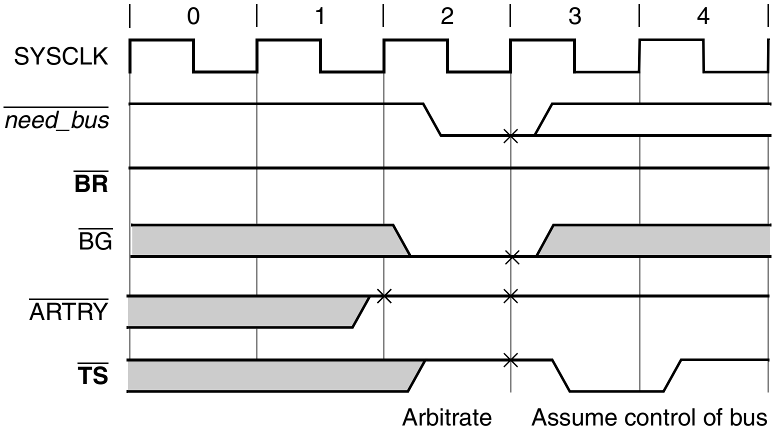


Figure 9-6. MPX Address Bus Arbitration—Parked Case

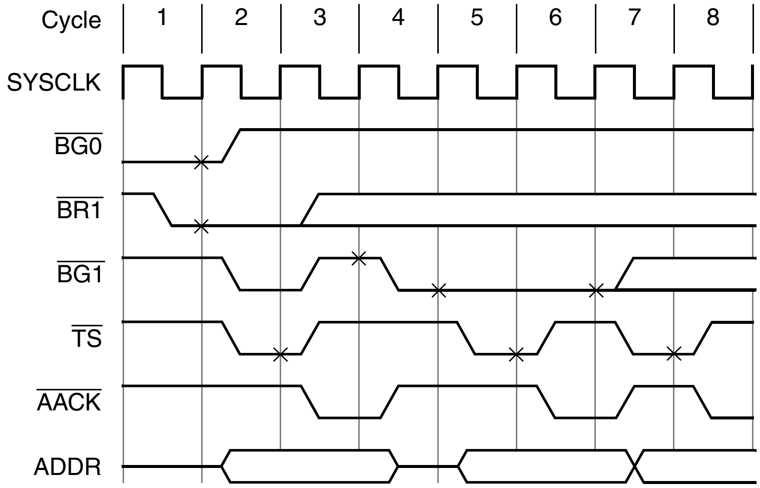
Whereas a non-parked processor must continually reassert $\overline{\text{BR}}$ to the arbiter in order to receive a bus grant, bus parking allows the arbiter to hold $\overline{\text{BG}}$ asserted. Parking permits the processor to skip the bus request (note its inactivity in Figure 9-6) and on the next cycle assume address bus ownership. Address bus tenures can be driven every other cycle by the same master.

The overall access latency for the memory transaction is shortened by one cycle: the system gains back not only the arbitration latency, but also the dead cycle that is between each tenure in the MPX bus interface.

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 predict correctly the next device requesting bus mastership.

From the system arbiter's perspective, address bus parking must be implemented more carefully in MPX bus systems than in 60x bus systems because the qualified bus grant equation no longer includes the $\overline{\text{ABB}}$ signal.

As shown in Figure 9-7, optimal address parking can be implemented in a multimaster system because $\overline{\text{TS}}$ is still in the qualified bus grant equation for MPX bus masters.



Cycle 1: Master 0 has a parked address bus grant. Master 0 has an address tenure ready and a qualified bus grant, so it queues an address tenure for the next cycle. Also in cycle 1, the arbiter samples a bus request from master 1, so the arbiter queues a switch of the bus grants from master 0 to master 1. The arbiter can safely do this because the qualified bus grant equation for MPX bus masters includes ¬$\overline{\mathrm{TS}}$.

Cycle 2: Master 0 begins an address tenure, and master 1 does NOT get a qualified bus grant.

Cycle 3: The arbiter MUST negate the bus grant to master 1 because, without an address bus busy indication or $\overline{\mathrm{AACK}}$ in the qualified bus grant equation, nothing would prevent master 1 from beginning an address tenure in cycle 4 and colliding with the end of master 0's address tenure. Because it would introduce a difficult timing path to require the arbiter to sample $\overline{\mathrm{TS}}$ in cycle 2 and negate $\overline{\mathrm{BG1}}$ in cycle 3, it is suggested that arbiters always pulse $\overline{\mathrm{BG}x}$ high a cycle after swapping $\overline{\mathrm{BG}x}$ and $\overline{\mathrm{BG}y}$, only reasserting $\overline{\mathrm{BG}x}$ after $\overline{\mathrm{AACK}}$ has been driven.

Cycle 4: The arbiter reasserts $\overline{\mathrm{BG1}}$ because it asserted $\overline{\mathrm{AACK}}$ in the previous cycle. (If the arbiter does not know in advance when $\overline{\mathrm{AACK}}$ is to be asserted, this timing might be difficult, and the reassertion of bus grant may have to be delayed a cycle, but most systems should be able to do this.)

Cycle 5: Master 1 gets to start its address tenure. Note that this is the optimal timing for a new master to drive the address bus.

Cycles 5 and 6: The arbiter speculatively parks $\overline{\mathrm{BG1}}$ enabling master 1 to begin another address tenure immediately in cycle 7.

Figure 9-7. Address Parking in MPX Bus Multiprocessor Systems

9.3.2 MPX Bus 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 the description of bus snooping in Section 9.3.3, "MPX Bus Address Tenure Termination." The signals used in this phase are transfer start ($\overline{\mathrm{TS}}$), and the address and attributes signals listed in the signal tables.

The MPC7450 supports a little-endian mode in which the low-order address bits are operated on (or *munged*) based on the program-requested data transfer size. This munging is performed internally before the address reaches the internal caches and bus units. When little-endian mode is selected, the MPX bus interface still operates in big-endian mode. That is, byte address 0 of a double word—as selected by A[33:35] on the bus—still selects the most significant (left-most) byte of the double word on data bus byte

D[0:7]. Byte lane swapping or other operations may have to be performed externally by the system if the MPC7450 is interfaced to a true little-endian environment.

Note that the MPC7450 does not work with the MPC106 bridge device in little-endian mode if misaligned data is accessed.

The signals used in the address transfer include the following signal groups:

- Address transfer start signal—transfer start (TS)
- Address transfer signals—address bus (A[0:35]) and address parity (AP[0:4])
- Address transfer attribute signals—transfer type (TT[0:4]), transfer size (TSIZ[0:2]), transfer burst (TBST), cache inhibit (CI), write-through (WT), and global (GBL)

The MPC7450 can be configured to use an extended, 36-bit address bus using HID0[XAEN]. When extended physical addressing is disabled and when functioning as an output, the four most significant bits (A[0:3]) are zeroes. Note that unused address pins cannot be left floating. If any address pins are unused by the system, they should be driven by the system during the address tenure of the transaction to be snooped or tied low with a pulldown resistor. When extended physical addressing is enabled, the address bus contains a 36 bit physical address.

Figure 9-8 shows that the timing for all of these signals, except TS, is identical. All of the address transfer and address transfer attribute signals are combined into the ADDR+ grouping in Figure 9-8. The TS signal indicates that the MPC7450 has begun an address transfer and the address and transfer attributes are valid (within the context of a synchronous bus).

In Figure 9-8, the address transfer occurs during bus clock cycles 1 and 2 (arbitration occurs in bus clock cycle 0, and the address transfer is terminated in bus clock 3). In this diagram, the address bus termination input, AACK, is asserted to the MPC7450 on the bus clock following assertion of TS (as shown by the dependency line). This is the minimum duration of the address transfer for the MPC7450; the duration can be extended by delaying the assertion of AACK for one or more bus clock cycles.

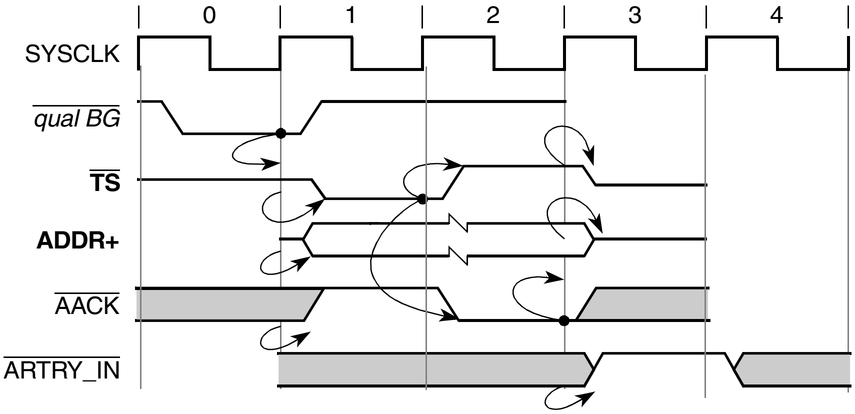


Figure 9-8. Address Bus Transfer

9.3.2.1 Address Bus Driven Mode

In addition to selecting MPX bus mode at the negation of $\overline{\text{HRESET}}$, the $\overline{\text{BMODE0}}$ signal is used to select address bus driven mode after $\overline{\text{HRESET}}$ is negated. In this mode, the address bus is actively driven by the MPC7450 on every cycle after a qualified bus grant is sampled, independent of whether the MPC7450 has a bus transaction to run or not. When the MPC7450 is driving the address bus but not running an address transaction, the address bus is not driven to any specific value. This mode provides for improved electrical characteristics on the address and attributes signals by reducing the time that these signals are not actively driven.

If $\overline{\text{BMODE0}}$ is asserted after $\overline{\text{HRESET}}$ is negated, address bus driven mode is selected; if $\overline{\text{BMODE0}}$ is negated after $\overline{\text{HRESET}}$ is negated, normal address bus driving mode (address bus not always driven) is selected. The read-only ABD bit in MSSCR0 indicates whether the MPC7450 is in address bus driven mode.

9.3.2.2 Address Bus Streaming

In MPX bus mode, the MPC7450 can perform address bus streaming, driving consecutive address tenures from the same master without a dead cycle in between. Two-cycle address tenures are possible if $\overline{\text{AACK}}$ is not delayed and the same master receives a qualified bus grant to drive another address tenure.

9.3.2.3 Address Bus Parity

When the MPC7450 is the address bus master and a valid address is on the bus, the MPC7450 always generates one bit of correct odd-byte parity for each of the four bytes of address plus one additional bit of parity for A[0:3]. AP[0] contains odd parity for A[0:3]; AP[1] contains odd parity for A[4:11]; AP[2] contains odd parity for A[12:19]; AP[3] contains odd parity for A[20:27]; and AP[4] contains odd parity for A[28:35]. For pull-up/pull-down requirements, see the hardware specifications.

If the MPC7450 is not the master and $\overline{\text{TS}}$ is asserted, the MPC7450 calculates parity values for the address bus and the calculated values are compared with the AP[0:4] inputs even if extended addressing is disabled. If there is an error and address parity checking is enabled (HID1[EBA] = 1), a machine check exception is generated. An address bus parity error causes a checkstop condition if MSR[ME] is cleared to 0. For more information about checkstop conditions, see Chapter 4, “Exceptions.”

The MPC7450 does not implement an address parity error (APE) signal as found on previous microprocessors that implement the PowerPC architecture.

Note that unused address parity pins cannot be left floating. If the address bits corresponding to an address parity bit are always driven to a 0, the parity bit must be driven to a 1. If the address bits are attached to pull-down resistors, the corresponding address parity bit should be attached to a pull-up resistor.

9.3.2.4 Address Transfer Attributes

The transfer attributes include the transfer type (TT[0:4]), transfer burst ($\overline{\text{TBST}}$), transfer size (TSIZ[0:2]), write-through ($\overline{\text{WT}}$), cache inhibit ($\overline{\text{CI}}$), and global ($\overline{\text{GBL}}$) signals. Attribute differences from the 60x bus are as follows:

- The definition of the TSIZ signals is expanded. See Section 9.3.2.4.2, “Transfer Size (TSIZ[0:2]) and Transfer Burst TBST Signals.”
- The read claim (RCLAIM) transfer type is added to the MPX bus mode for accesses generated by touch-for-store instructions.

9.3.2.4.1 Transfer Type (TT[0:4]) Signals

The transfer type signals (TT[0:4]) indicate the type of transaction to be performed. 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. Table 9-1 describes the MPX bus specification transfer encodings and the behavior of the MPC7450 as a bus master. The MPX bus transfer type encodings are the same as those for the 60x bus with the addition of the RCLAIM command. RCLAIM is used to identify touch-for-store instructions on the MPX bus. The effect of the RCLAIM transaction is to establish exclusive ownership of a cache line without marking that cache line as modified in the requesting processor’s cache.

Table 9-1. Transfer Type Encodings for MPX Bus Mode

Generated by MPC7450 as Bus Master		TT0	TT1[1]	TT2	TT3	TT4	MPX Bus Specification	
Type	**Source**						**Command**	**Type**
Address only	**dcbst**	0	0	0	0	0	Clean block	Address only
Address only	**dcbf**	0	0	1	0	0	Flush block	Address only
Address only	sync	0	1	0	0	0	sync	Address only
Address only	**dcba**, **dcbz** [2]	0	1	1	0	0	Kill block	Address only
Address only	**eieio**	1	0	0	0	0	**eieio**	Address only
Single-beat write (non$\overline{\text{GBL}}$)	**ecowx**	1	0	1	0	0	External control word write	Single-beat write
Address only	**tlbie**	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	—
Address only	**tlbsync**	0	1	0	0	1	**tlbsync**	Address only
Address only	**icbi**	0	1	1	0	1	**icbi**	Address only
N/A	N/A	1	X	X	0	1	Reserved	—

Table 9-1. Transfer Type Encodings for MPX Bus Mode (continued)

Generated by MPC7450 as Bus Master		TT0	TT1[1]	TT2	TT3	TT4	MPX Bus Specification	
Type	**Source**						**Command**	**Type**
Single-beat write or burst	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, **dcbf**, **dcbi, dcbst** push, or snoop copyback	0	0	1	1	0	Write-with-kill	Burst
Single-beat read or burst	Data load or instruction fetch	0[3]	1	0	1	0	Read	Single-beat read or burst
N/A	N/A	1	0	0	1	0	Write-with-flush-atomic[4, 5]	Single-beat write
N/A	N/A	1	0	1	1	0	Reserved	N/A
Burst	**lwarx** [5]	1[1]	1	0	1	0	Read-atomic	Single-beat read or burst
Burst	**stwcx.** [5]	1	1	1	1	0	Read-with-intent-to-modify-atomic	Burst
Burst	Store miss	0	1	1	1	0	Read-with-intent-to-modify	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
Burst	**dstst**, **dststt**, or **dcbtst**	0	1	1	1	1	Read claim (RCLAIM)	Burst
N/A	N/A	1	X	X	1	1	Reserved	—

[1] TT1 can generally be interpreted as a read/write indicator for the bus.

[2] The processor can also issue a kill block if a series of stores results in an entire cache line being modified, see Section 3.1.2.3, “Store Gathering/Merging,” for details.

[3] TT0 differentiates between a Read-atomic (**lwarx**) operation—TT0 high, and a Read (cache-inhibiting load or instruction fetch) operation—TT0 low.

[4] Not generated by the MPC7450, but snooped.

[5] Caching-inhibited **lwarx** and caching-inhibited or write-through **stwcx.** cause DSI exceptions on the MPC7450.

9.3.2.4.2 Transfer Size (TSIZ[0:2]) and Transfer Burst $\overline{\text{TBST}}$ Signals

The TSIZ[0:2] signals indicate the size of the requested data transfer and may be used along with $\overline{\text{TBST}}$ and A[32:35] 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.

The MPC7450 allows for two burst sizes in order to support both cache block transfers (of 32 bytes) and caching-inhibited or write-through AltiVec loads and stores (of 16 bytes). Thus the definition of the

TSIZ[0:2] bits when $\overline{\text{TBST}}$ is asserted is expanded from that in 60x bus mode. Table 9-2 defines the $\overline{\text{TBST}}$ and TSIZ[0:2] encodings used by the MPC7450 in MPX bus mode.

Table 9-2. $\overline{\text{TBST}}$ and TSIZ[0:2] Encodings in MPX Bus Mode

$\overline{\text{TBST}}$	TSIZ0	TSIZ1	TSIZ2	Transfer Size[1]
Asserted	0	0	0	Not supported [2]
Asserted	0	0	1	Quad-word (16-byte) burst
Asserted	0	1	0	32-byte burst
Asserted	0	1	1	Undefined
Asserted	1	0	0	Not supported [2]
Asserted	1	0	1	undefined
Asserted	1	1	0	Undefined
Asserted	1	1	1	Undefined
Negated	0	0	0	8 bytes
Negated	0	0	1	1 byte
Negated	0	1	0	2 bytes
Negated	0	1	1	3 bytes
Negated	1	0	0	4 bytes
Negated	1	0	1	5 bytes [3]
Negated	1	1	0	6 bytes [3]
Negated	1	1	1	7 bytes [3]

[1] 3-byte transfers may be requested by the MPC7450 starting at any byte address within the double word from byte address 0 to byte address 5.

4-byte transfers may be requested by MPC7450 starting at any byte address within the double word from byte address 0 to byte address 4.

[2] Not supported means the transfer size is defined by the MPX bus, but is not supported by the MPC7450.

[3] Although defined, the MPC7450 never generates a transaction of this size.

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 MPC7450. The MPC7450 never generates a bus transaction with a transfer size of 5 bytes, 6 bytes, or 7 bytes.

For **eciwx/ecowx** operations, a transfer size of 4 bytes is implied, and the $\overline{\text{TBST}}$ and TSIZ(0:2) signals are redefined to specify the resource identifier (RID). The RID is copied from the three low-order bits of the external address register (EAR[28-31]). For these operations, the $\overline{\text{TBST}}$ signal indicates the state of EAR[28] without inversion (active high).

9.3.2.4.3 Write-Through ($\overline{\text{WT}}$), Cache Inhibit ($\overline{\text{CI}}$), and Global ($\overline{\text{GBL}}$) Signals

In general, the MPC7450 provides the $\overline{\text{WT}}$, $\overline{\text{CI}}$, and $\overline{\text{GBL}}$ signals to indicate the status of a transaction target as determined by the WIM bit settings during address translation by the MMU. There are exceptions, as described in Section 3.8.3, “Transfer Attributes.”

During write operations, the $\overline{\text{WT}}$ signal reflects the write-through status for the transaction as determined by the MMU address translation. It is also asserted for burst writes due to **dcbf** (flush) and **dcbst** (clean) instructions and for snoop pushes. The $\overline{\text{WT}}$ signal is negated for accesses caused by the execution of **ecowx** instructions. Because write-through status is not meaningful for read operations, the MPC7450 uses the $\overline{\text{WT}}$ signal during read transactions to indicate that the transaction is an instruction fetch ($\overline{\text{WT}}$ = 1) or a data load ($\overline{\text{WT}}$ = 0) operation.

The MPC7450 also provides the $\overline{\text{CI}}$ transfer attribute. This signal reflects the caching-inhibited status of the transaction as determined by the MMU address translation (WIM bits). Note that if the L1 data cache is disabled, all data accesses are treated as caching-inhibited and $\overline{\text{CI}}$ is asserted regardless of the WIM bit settings. $\overline{\text{CI}}$ is always asserted for **eciwx/ecowx** bus transactions independent of the address translation.

The MPC7450 provides the additional transfer attribute $\overline{\text{GBL}}$. This signal indicates the coherency requirements (global or non-global) for the transaction as determined by the MMU address translation.

9.3.2.5 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. However, 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. Non-intervention burst write transfers are always performed zero double word first. Intervention burst writes (see Section 9.4.2.4, “MPX Bus Data Intervention,”) are performed critical double word first.

Table 9-3 describes the order of the double words (DW) transferred during burst operations.

Table 9-3. Burst Ordering

Data Transfer	For Starting Address:			
	A[31:32] = 00	A[31:32] = 01	A[31:32] = 10	A[31:32] = 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

Notes: A[31:32] specifies the 1st double word of the 32-byte block being transferred; the remaining double words to transfer must wrap around the block.

A[33:35] are always 0b000 for burst transfers performed by the MPC7450.

DW*n* represents the double word addressed by A[31:32]=*n* if a nonburst transfer were performed.

Each data beat is terminated with one valid assertion of $\overline{\text{TA}}$.

9.3.2.6 Effect of Alignment in Data Transfers

Table 9-4 lists the aligned transfers that can occur on the MPX bus. These are transfers in which the data is aligned to an address that is an integral multiple of the size of the data. For example, Table 9-4 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 9-4. Aligned Data Transfers

Transfer Size	TSIZ[0:2]	A[33:35]	Data Bus Byte Lane(s)[1]							
			0	1	2	3	4	5	6	7
Byte	0 0 1	0 0 0	A	—	—	—	—	—	—	—
	0 0 1	0 0 1	—	A	—	—	—	—	—	—
	0 0 1	0 1 0	—	—	A	—	—	—	—	—
	0 0 1	0 1 1	—	—	—	A	—	—	—	—
	0 0 1	1 0 0	—	—	—	—	A	—	—	—
	0 0 1	1 0 1	—	—	—	—	—	A	—	—
	0 0 1	1 1 0	—	—	—	—	—	—	A	—
	0 0 1	1 1 1	—	—	—	—	—	—	—	A
Half word	0 1 0	0 0 0	A	A	—	—	—	—	—	—
	0 1 0	0 1 0	—	—	A	A	—	—	—	—
	0 1 0	1 0 0	—	—	—	—	A	A	—	—
	0 1 0	1 1 0	—	—	—	—	—	—	A	A
Word	1 0 0	0 0 0	A	A	A	A	—	—	—	—
	1 0 0	1 0 0	—	—	—	—	A	A	A	A
Double word	0 0 0	0 0 0	A	A	A	A	A	A	A	A

[1] These indicate the byte portions of the requested operand that are read or written during that bus transaction. Entries marked with '—' are not required and are ignored during read transactions; they are driven with undefined data during all write transactions. Entries marked within 'A' means that byte lane is used.

Because the processor has an on-chip, copyback cache, most bus transactions issued to the MPX bus are burst reads and burst writes that are aligned to double-word boundaries. The only transfers that may have alignment considerations on the bus are single beat transactions that bypass the cache (cache-disabled, caching-inhibited, and write-through store transactions). The MPC7450 allows the transfer of any block of contiguous bytes within a doubleword. No single beat bus transactions may cross a double-word boundary. Transfers of strings or data that are aligned in such a way that they cross a double-word boundary are broken down into multiple bus transactions. Two-beat transactions generated by caching-inhibited or write-through AltiVec loads and stores or by store-gathered, non-burst writes must be aligned to a natural quad-word boundary.

9.3.2.6.1 Misalignment Example

Although most operations hit in the primary cache (or generate burst memory operations if they miss), the MPC7450 interface supports misaligned transfers within a double-word (64-bit aligned) boundary, as shown in Table 9-5. Note that the 4-byte transfer in Table 9-5 is only one example of misalignment. As long as the attempted transfer does not cross a double-word boundary, the MPC7450 can transfer the data on the misaligned address (for example, a half-word read from an odd byte-aligned address). Any attempt to address data that crosses a double-word boundary requires two bus transfers to access the data.

Due to the performance degradations, misaligned memory operations should be avoided. In addition to the double-word straddle boundary condition, the address translation logic can generate substantial exception overhead when the load/store multiple and load/store string instructions access misaligned data. It is strongly recommended that software attempt to align data where possible.

Table 9-5. Misaligned Data Transfers (4-Byte Examples)

Transfer Size (Four Bytes)	TSIZ[0:2]	A[33:35]	Data Bus Byte Lanes[1]							
			0	1	2	3	4	5	6	7
Aligned	1 0 0	0 0 0	A	A	A	A	—	—	—	—
Misaligned	1 0 0	0 0 1	—	A	A	A	A	—	—	—
Misaligned	1 0 0	0 1 0	—	—	A	A	A	A	—	—
Misaligned	1 0 0	0 1 1	—	—	—	A	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	—	—	—	—	—

[1] A = Byte lane used
— = Byte lane not used

9.3.2.6.2 Alignment of External Control Instructions

The size of the data transfer associated with **eciwx** and **ecowx** instructions is always four bytes. If an operand for an **eciwx** or **ecowx** instruction is misaligned and crosses any word boundary, the MPC7450 generates an alignment exception.

9.3.3 MPX Bus Address Tenure Termination

The address tenure of a bus operation is terminated when completed with the assertion of $\overline{\text{AACK}}$ (address acknowledge).

The assertion of $\overline{\text{AACK}}$ can be as early as one bus clock cycle following $\overline{\text{TS}}$ (see Figure 9-10), which allows a minimum address tenure of two bus cycles. Thus, using address bus streaming, two-cycle address tenures are possible if $\overline{\text{AACK}}$ is not delayed and the same master receives a qualified bus grant to drive another address tenure. Note that $\overline{\text{AACK}}$ must be asserted for only one bus clock cycle.

Because the MPC7450 does not terminate an address transfer until the $\overline{\text{AACK}}$ input is asserted, the system can extend or pace the address transfer phase by delaying the assertion of $\overline{\text{AACK}}$ to the MPC7450. The MPC7450 requires a minimum of 5 core cycles to process a snoop and generate a response after latching $\overline{\text{TS}}$ and associated transfer attributes. As a result, if the processor core frequency is less than five times the system bus frequency, the system must extend the address tenure of all transactions that are snooped by the processor by delaying assertion of $\overline{\text{AACK}}$. For core to bus frequency ratios of 2:1 and 2.5:1, $\overline{\text{AACK}}$ must be delayed a minimum of two bus cycles; for core:bus frequency ratios of 3:1, 3.5:1, 4:1, and 4.5:1, $\overline{\text{AACK}}$ must be delayed a minimum of one bus cycle. Table 9-6 summarizes the required system $\overline{\text{AACK}}$ delay for ratios less than 5:1.

Table 9-6. Required System $\overline{\text{AACK}}$ Delay for Ratios < 5:1

Dynamic PLL Ratio	Required $\overline{\text{AACK}}$ Delay
2:1	2 cycles
2.5:1	
3:1	1 cycle
3.5:1	
4:1	
4.5:1	

Since the processor requires five internal clock cycles to provide the correct response to a snoop on the external bus, delaying address acknowledge assures that the correct snoop response will be asserted by the processor.

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 of $\overline{\text{ARTRY}}$ causes the entire transaction (address and data tenure) to be rerun. $\overline{\text{ARTRY}}$ is asserted during the address tenure for the following reasons:

- No device on the bus has the required buffer space to handle the transaction.
- A device has a transient pipeline collision.
- A snooping device must push modified data to maintain coherency.

As a snooping device, the MPC7450 asserts $\overline{\text{ARTRY}}$ if a snooped transaction hits modified data in the data cache and data intervention is disabled (MSSCR0[EIDIS] = 0b1) or data intervention is enabled but intervention is not possible. The MPC7450 also asserts $\overline{\text{ARTRY}}$ if the cache line is in an internal castout queue or if the snooped transaction could not be serviced.

As a bus master, the MPC7450 responds to the assertion of $\overline{\text{ARTRY}}$ by aborting the bus transaction and re-requesting the bus. Note that after recognizing the assertion of $\overline{\text{ARTRY}}$ and aborting the transaction in progress, the MPC7450 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. Internally, the address queue that was retried is continually re-arbitrated with the other internal queues until the next qualified bus grant is recognized.

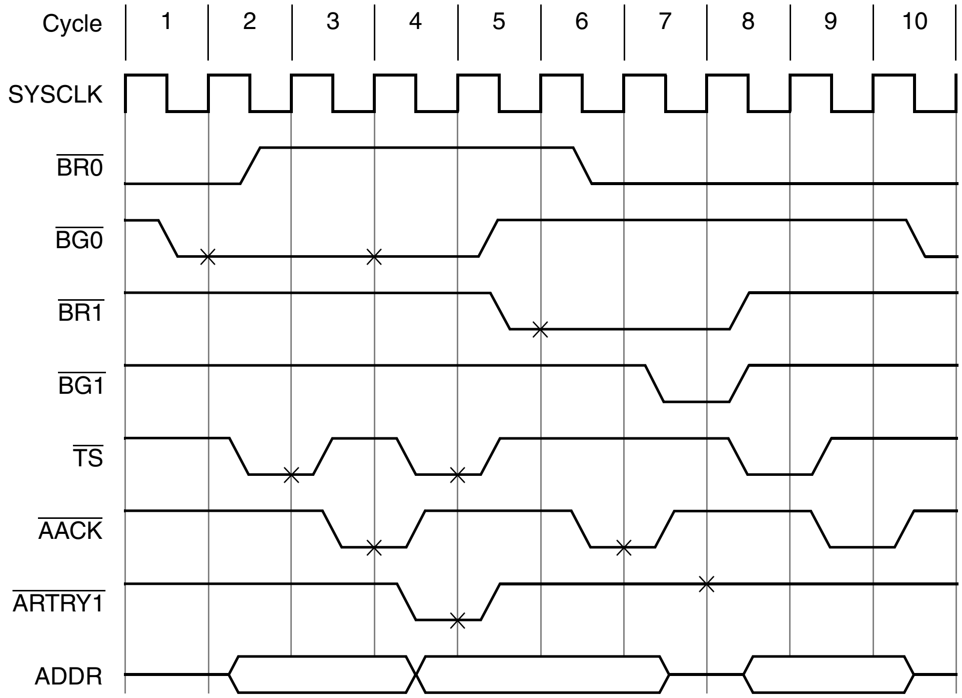
9.3.3.1 Address Retry Window and Qualified $\overline{\text{ARTRY}}$

If an address retry is required, $\overline{\text{ARTRY}}$ is asserted by a bus snooping device as early as the second cycle after the assertion of $\overline{\text{TS}}$. Once asserted, $\overline{\text{ARTRY}}$ must remain asserted through the cycle after the assertion of $\overline{\text{AACK}}$; the bus clock cycle starting two clock cycles after $\overline{\text{TS}}$ and ending with the cycle after the assertion of $\overline{\text{AACK}}$ is referred to as the address retry window.

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 MPC7450 does not recognize an early $\overline{\text{ARTRY}}$ but only recognizes $\overline{\text{ARTRY}}$ the cycle after $\overline{\text{AACK}}$ (the qualified $\overline{\text{ARTRY}}$). If the assertion of $\overline{\text{ARTRY}}$ is received up to or on the bus cycle of the first (or only) assertion of $\overline{\text{TA}}$ for the data tenure, the MPC7450 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.

$\overline{\text{ARTRY}}$ assertions can have additional implications because the MPC7450 allows new address tenures to begin without a dead cycle in between. The MPC7450 allows a new address tenure from the same master to begin the cycle after $\overline{\text{AACK}}$, which overlaps the address retry window of the previous address tenure. If this happens, the system and all bus devices must recognize that the second $\overline{\text{TS}}$ is implicitly retried as well. As with the 60x interface, however, $\overline{\text{ARTRY}}$ does not affect the termination of an address tenure—address tenures are only terminated by $\overline{\text{AACK}}$. Therefore, even if an address tenure is to be retried by $\overline{\text{ARTRY}}$ for the previous address tenure, $\overline{\text{AACK}}$ is asserted to terminate the second address tenure (shown in Figure 9-9).



Cycle 1: The master has requested the bus and receives a qualified bus grant.

Cycle 2: The master begins the address tenure by driving $\overline{\text{TS}}$ and the address.

Cycle 3: The system responds with $\overline{\text{AACK}}$, ending the address tenure. The master receives another (parked) address bus grant.

Cycle 4: The master begins a new address tenure by driving $\overline{\text{TS}}$ and a new address. Some snooping device, however, asserts $\overline{\text{ARTRY}}$ for the first transaction. Bus grant remains parked to processor 0.

Cycle 5: The system delays $\overline{\text{AACK}}$ for the second transaction for some reason. $\overline{\text{BG0}}$ is negated to allow the retrying processor to request the bus. Processor 1 takes advantage of this window of opportunity and requests the bus to perform a push of the data that caused the retry.

Cycle 6: The system asserts $\overline{\text{AACK}}$ to terminate the second address tenure. Because the window of opportunity has passed, processor 0 requests the address bus again to retry its transaction. But the arbiter may NOT rearbitrate and grant the address bus to processor 0 before the push requested in the window of opportunity.

Cycle 7: Even though this cycle would be the address retry window for the second address tenure, no processor may assert $\overline{\text{ARTRY}}$ because that transaction was implicitly canceled by the $\overline{\text{ARTRY}}$. (If $\overline{\text{AACK}}$ had not been delayed, an assertion of $\overline{\text{ARTRY}}$ here could cause contention with the snooper that would be driving $\overline{\text{ARTRY}}$ negated from the address retry window of the first address tenure.) A bus grant is given to processor 1 to perform its push.

Cycle 8: Processor 1 begins its snoop push.

Cycle 9: The snoop push address tenure is acknowledged and terminated.

Cycle 10: The arbiter now grants processor 0 the address bus to retry its transaction.

Figure 9-9. Overlapped $\overline{\text{ARTRY}}$ and $\overline{\text{TS}}$ (with a Delayed $\overline{\text{AACK}}$) in MPX Bus Mode

Note that in Figure 9-9, the $\overline{\text{AACK}}$ for the second address tenure is delayed by a clock cycle. This is to demonstrate how a system must handle a delayed $\overline{\text{AACK}}$ overlapping the window of opportunity after a retry. In this case the address bus grant for the snoop push requested in the window of opportunity must be delayed until at least the cycle after $\overline{\text{AACK}}$. This may be required to avoid critical timing conflicts between $\overline{\text{ARTRY}}$ and $\overline{\text{AACK}}$. Note that the grant could be delayed further to avoid any critical timing conflicts between $\overline{\text{AACK}}$ and the bus grant, if necessary. In any case, if the address tenure of the second transaction extends past the window of opportunity after the assertion of $\overline{\text{ARTRY}}$, the arbiter must not rearbitrate and grant the address bus to any device that may have requested the bus after the window of opportunity but before the address tenure for the snoop push.

9.3.3.2 Snoop Copybacks and the Window-of-Opportunity

During the clock cycle of a qualified $\overline{\text{ARTRY}}$, the MPC7450 also determines if it should negate $\overline{\text{BR}}$ and ignore $\overline{\text{BG}}$ on the following cycle. On the following cycle, all other bus devices negate address bus requests, and they do not qualify address bus grants. This cycle is the window of opportunity for the snooping master that asserted $\overline{\text{ARTRY}}$ and needs to perform a snoop copyback operation. Thus, the snooping master that asserted $\overline{\text{ARTRY}}$ is the only device allowed to assert $\overline{\text{BR}}$. 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 9-10.

When the MPC7450 asserts $\overline{\text{ARTRY}}$ due to a snoop operation and is ready to perform the snoop push, it always asserts $\overline{\text{BR}}$ in the window of opportunity to obtain bus mastership for the copyback cycle. (A copyback operation due to a snoop hit to a modified block is sometimes referred to as a snoop push.) Note that the copyback is a non-global ($\overline{\text{GBL}}$ negated) transaction. External devices on the MPX and 60x bus must not assert $\overline{\text{ARTRY}}$ for non-global transactions.

Note that even if the MPC7450 asserts $\overline{\text{BR}}$ in the window of opportunity for a snoop push, it may be several bus cycles later before the MPC7450 is able to perform the necessary transaction. The timing of $\overline{\text{TS}}$ may be dependent on resource constraints. The bus arbiter should keep $\overline{\text{BG}}$ asserted until it detects that $\overline{\text{BR}}$ is negated or $\overline{\text{TS}}$ is asserted from the MPC7450 indicating that the snoop copyback has begun. The system should ensure that no other address tenures occur until the current snoop push from the MPC7450 is completed.

It may occur in some systems that the MPC7450 was unable to perform a pending snoop copyback when a new snoop operation is performed. In this case, the MPC7450 requests the bus in the window of opportunity if it hits on the new snooped address, and it performs the snoop copyback operation for the earlier snooped address rather than for the current snooped address in order to clear its internal snoop queue. The MPC7450 may require up to three pending snoop copybacks to service a current snoop. In such cases, system arbiters may need to ensure that the MPC7450 does not respond with $\overline{\text{ARTRY}}$ before proceeding with another transaction (loop snooping).

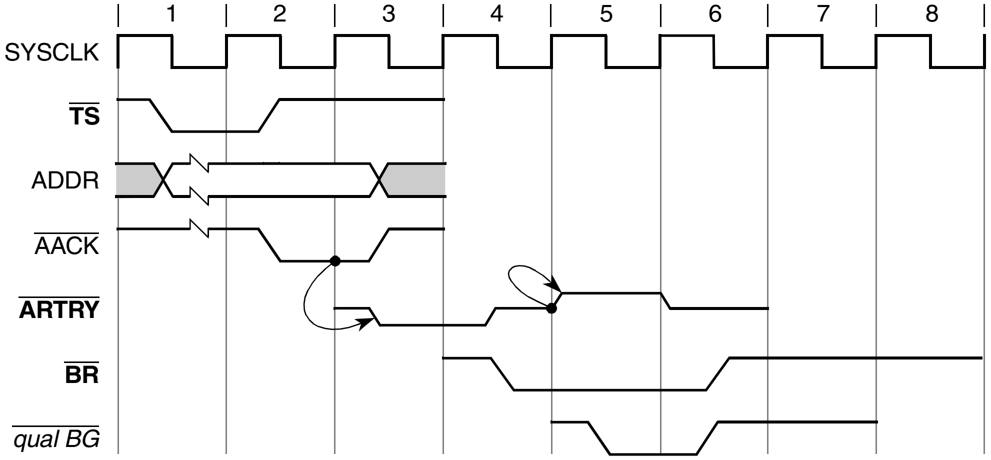


Figure 9-10. Snooped Address Cycle with $\overline{\text{ARTRY}}$

Both $\overline{\text{ARTRY}}$ and $\overline{\text{SHD}}$[0:1] go through a precharge cycle unless this is disabled by setting the precharge disable bit (HID1[PAR]). As shown in Figure 9-10, these signals are asserted for one bus clock cycle, released to high-impedance for half the next clock cycle, driven high for one clock cycle, and finally released to high-impedance for the remaining half clock cycle.

9.3.3.3 Shared ($\overline{\text{SHD0}}$, $\overline{\text{SHD1}}$) Signals in MPX Bus Mode

The shared response for the MPC7450 in MPX bus mode is divided into two separate output signals, $\overline{\text{SHD0}}$ and $\overline{\text{SHD1}}$. The MPX bus mode requires two signals because address tenures may occur every other cycle. Because the shared response must be driven negated between assertions and because multiple devices may drive a shared response in a given snoop response window, it is necessary to release the shared signal to high-impedance after asserting it, drive it negated, and release it to high-impedance again. Timing requirements make this very difficult for a single signal that may need to be asserted on the second cycle after a previous assertion.

If the MPC7450 needs to assert a shared snoop response and $\overline{\text{SHD0}}$ was not asserted in the cycle before or the cycle of $\overline{\text{TS}}$, it should be asserted in the snoop response window to indicate a shared response. If $\overline{\text{SHD0}}$ was asserted the cycle before or the cycle of $\overline{\text{TS}}$, $\overline{\text{SHD1}}$ must be used to indicate a shared response. A master observing the snoop response must consider the shared response asserted if either $\overline{\text{SHD0}}$ or $\overline{\text{SHD1}}$ is asserted.

The timing may vary for the release to high-impedance, negating, and re-release to high-impedance of $\overline{\text{SHD0}}$ and $\overline{\text{SHD1}}$. To ensure compatibility with the standard 60x interface in which $\overline{\text{SHDx}}$ might need to be asserted up to every three bus cycles, the MPC7450 implements the 60x-style timing for both $\overline{\text{SHD0}}$ and $\overline{\text{SHD1}}$; that is, $\overline{\text{SHD0}}$ and $\overline{\text{SHD1}}$ have the same timing as $\overline{\text{ARTRY}}$, in which the signal is released to high-impedance for a fraction of a cycle, negated for up to an entire cycle (crossing a bus cycle boundary) before being released to high-impedance again. Note that future implementations with the MPX bus protocol may define this timing differently. The MPC7450 does not assert either $\overline{\text{SHD0}}$ or $\overline{\text{SHD1}}$ any more often than every fourth bus clock cycle (see Figure 9-11).

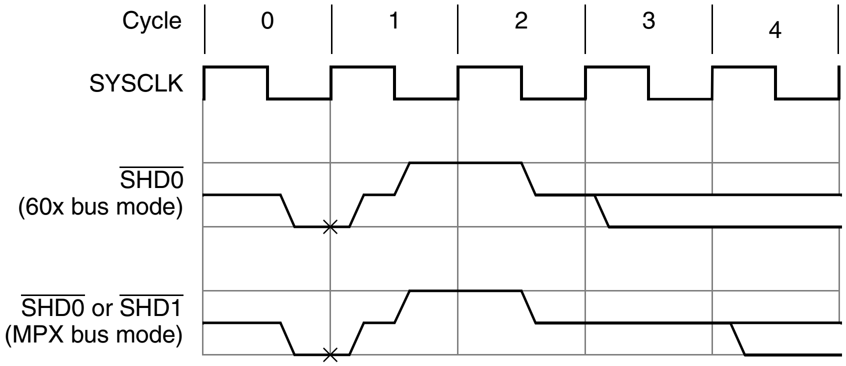


Figure 9-11. SHD0 and SHD1 Negation Timing

To ease timing requirements, devices detecting SHD0 or SHD1 should not sample these signals beyond the second cycle after AACK, because the signals can either be released to high-impedance or fractionally precharged in those cycles.

9.3.3.4 Hit (HIT) Signal and Data Intervention

The HIT signal allows the MPX bus to support cache-to-cache transfers and local bus slaves. A device that asserts HIT in the address retry window indicates to the system that it owns the data for the requested load, and that it can supply the data by performing a data-only transaction; this is known as data intervention. Note that the HIT signal is a point-to-point signal between the MPC7450 and the central arbiter/memory controller. The master that requested the load does not sample the HIT signal and does not know that the data is coming from a master or local bus slave rather than from memory.

When data intervention is enabled (MSSCR0[EIDIS] = 0b0), the MPC7450 attempts to intervene to service a load when it detects a snoop hit to modified data (see Section 3.3.2, “Coherency Support”). Unless ARTRY is asserted simultaneously by another bus device, the MPC7450 requests a data-only transaction by using the DRDY protocol described in Section 9.4.2.4, “MPX Bus Data Intervention.”

Normally, if a snooping processor asserts ARTRY and has modified data for that request, it asserts BR during the window-of-opportunity to perform a snoop push. If the MPC7450 asserts HIT and another device asserts ARTRY, the MPC7450 does not assert BR in the window-of-opportunity to perform the push. The system must perform fair arbitration to ensure that the push is allowed to make progress, or let the retried master attempt the transaction again at which point the MPC7450 will assert ARTRY and assert BR in the window-of-opportunity to perform the snoop push. Note that in some instances, a single master may assert both ARTRY and HIT in response to a snooped transaction. In this case, the ARTRY has precedence and the HIT signal is ignored.

When the MPC7450 performs data intervention, it always provides 32-bytes of intervention data in critical-double-word-first ordering, regardless of the TSIZ[0:2] encoding or the state of the TBST signal for the snooped transaction. Thus, if a system implementation provides global snoop traffic to the MPC7450 that could result in data intervention from the MPC7450 and does not consist of 32-byte transfers, the system must either sink the entire 32-bytes of intervention data from the MPC7450 or else disable all types data intervention in the MPC7450 by setting MSSCR0[EIDIS].

9.4 MPX Bus Data Tenure

This section describes the three phases of the data tenure used in the MPX bus protocol—data bus arbitration, data transfer, and data transfer termination. In addition, the MPX bus implements several features to improve both bandwidth and utilization of the data bus compared with the 60x bus mode. These include the following:

- Support for data-only transactions used for cache-to-cache data transfers (data intervention) and data transfers from local bus slaves
- Support for data tenure reordering
- Simplification of signals and modes to optimize the most timing-critical logic paths
- Support for data streaming which eliminates the requirement for a dead cycle between burst data tenures from a single source

9.4.1 MPX Bus Data Bus Arbitration

The $\overline{\text{TS}}$ signal is an implied data bus request from the MPC7450; the arbiter must qualify $\overline{\text{TS}}$ with the transfer type (TT[0:4]) signals to determine if the current address transfer is an address-only operation, which does not require a data bus transfer. If the data bus is needed, the arbiter grants data bus mastership by asserting the $\overline{\text{DBG}}$ input to the MPC7450. Additionally, TSIZ[0:2], and $\overline{\text{TBST}}$ signals provide information about the requested data transfer to external logic.

9.4.1.1 Qualified Data Bus Grant in MPX Bus Mode

As with the address bus arbitration phase, the MPC7450 must qualify the $\overline{\text{DBG}}$ input with a number of input signals before assuming bus mastership. In MPX bus mode, the qualified data bus grant equation is as follows:

Qualified Data Bus Grant = $\overline{\text{DBG}}$ & ¬($\overline{\text{ARTRY}}$& retriable) & ¬(latched state variables)

In MPX bus mode, a qualified data bus grant occurs when the following conditions are satisfied:

- $\overline{\text{DBG}}$ is asserted
- $\overline{\text{ARTRY}}$ is not asserted in the address retry window for the address phase of this transaction; in other words, if the address tenure is currently in progress, then the ARTRY signal has not been asserted anytime from the beginning of the retry window up to this point; or if the address tenure has already completed, then the ARTRY was not asserted anytime during the retry window.
- The MPC7450 is ready to begin a transaction, meaning the following:
  - — The processor is not already using the data bus, or
  - — The processor has a burst access in progress, the processor has already received the next-to-last $\overline{\text{TA}}$ for the current burst access, and the source of the next access is the same as the source of the current access.

These conditions mean that the system arbiter must never assert $\overline{\text{DBG}}$ to a processor when the data bus is busy with a transaction for another processor. The system arbiter must synthesize its own data bus busy state. However, note that if a transaction for device A is currently receiving its last $\overline{\text{TA}}$ for a multi-beat transaction and the same device is ready to source data to processor B, $\overline{\text{DBG}}$ to processor B may be asserted, and the data source can continue to stream another data tenure.

The negation of $\overline{\text{ARTRY}}$ for that address tenure indicates that the address tenure associated with the data tenure about to be granted was not retried by a snooping device. Because of address pipelining, an assertion of $\overline{\text{ARTRY}}$ may not be for the data tenure about to be granted; therefore, it may not affect its data bus grant qualification.

One bus clock cycle after accepting a qualified data bus grant, the MPC7450 begins driving or sampling the data bus and sampling the transfer acknowledge signals.

9.4.2 MPX Bus Data Transfer

The data transfer signals of the MPC7450 include D[0:63] and DP[0:7]. For memory accesses, the D[0:63] signals form a 64-bit data path for read and write operations. The MPC7450 does not directly support dynamic interfacing to subsystems with less than a 64-bit wide data path, and it does not support a static 32-bit data bus mode. See Section 9.3.2, “MPX Bus Address Transfer,” for more information.

The MPC7450 transfers data in either four-beat burst, two-beat burst, or single-beat transfers. The type of transaction initiated by the MPC7450 depends on whether the code or data is caching-inhibited or caching-allowed and, for store operations, whether the cache is in write-back or write-through mode which is controlled by software on either a page or block basis. Memory structures must be marked as caching-allowed (and write-back for data store operations) in the respective page or block descriptor to take advantage of burst transfers as they can achieve significantly higher bus throughput than single-beat operations.

Four-beat burst transfers are used to transfer 32-byte cache blocks into or out of the internal caches. Two-beat burst transfers are used for caching-inhibited or write-through 128-bit AltiVec loads or stores. Four-beat or two-beat stores can be created by gathering multiple smaller stores. Single-beat transfers (one to eight bytes) are performed for non-AltiVec caching-inhibited or write-through operations.

Single-beat transactions may be either aligned or misaligned depending on the load or store instruction addressing (see Section 9.3.2.6, “Effect of Alignment in Data Transfers”). Two-beat no-cacheable or write-through AltiVec loads or stores are always aligned on quad word boundaries. Four-beat burst operations are always aligned on cache line address boundaries. A burst transfer has an assumed address order. For load or store operations that miss in the cache (and are marked as caching-allowed and, for stores, write-back in the MMU), the MPC7450 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 eight-word-aligned data.

9.4.2.1 Data Bus Parity

The MPC7450 generates one bit of odd parity for each byte of the data bus when it is driving data for a write transaction. The MPC7450 checks for correct odd parity across the entire data bus when it is receiving data for a read transaction if data parity checking is enabled in HID1[EBD].

The data bus parity (DP[0:7]) signals are assigned to the corresponding data signals as shown in Table 9-7.

Table 9-7. Correspondence of Data Parity Signals with Data Signals

Data Parity Signals	Data Signals
DP[0]	D[0:7]
DP[1]	D[8:15]
DP[2]	D[16:23]
DP[3]	D[24:31]
DP[4]	D[32:39]
DP[5]	D[40:47]
DP[6]	D[48:55]
DP[7]	D[56:63]

For pull-up/pull-down requirements, see the hardware specifications.

If a data parity error is detected, the processor may conditionally take a machine check interrupt or enter the checkstop state as directed by MSR[ME]. Note that the MPC7450 does not implement a data parity error (DPE) signal as found on previous microprocessors that implement the PowerPC architecture.

9.4.2.2 Earliest Transfer of Data

Because the MPC7450 has no means to retry a data tenure, data must never be transferred before the last cycle of the snoop response window. Do not issue a $\overline{\text{TA}}$ before an $\overline{\text{AACK}}$.

An additional requirement in systems that support data intervention or local bus slaves that use the $\overline{\text{HIT}}$/$\overline{\text{DRDY}}$ protocol is that data must only be driven or sampled after the completion of the snoop response window, so that the $\overline{\text{HIT}}$ signal can be sampled.

9.4.2.2.1 Data Streaming in MPX Bus Mode

The MPX bus mode supports a streaming mode of data transfer that allows burst data tenures from a single source to be driven back-to-back without requiring a dead cycle between the tenures.

Data streaming in MPX bus mode is supported only if the following conditions are satisfied:

- Data source is the same.
- First transfer is a multiple-beat transfer.

The first condition for data streaming is to prevent contention on the data bus. Streaming must not be allowed if data is driven by two different devices.A dead cycle must be placed between the two adjacent data tenures to allow the first device to stop driving the bus before the second device starts driving the bus. For example, if the first data transfer is a processor read from memory and the second data transfer is a processor write to memory, data streaming is not supported; a minimum of one dead cycle must be inserted between the final $\overline{\text{TA}}$ of the read and the first $\overline{\text{TA}}$ of the write. In this example, the earliest cycle that the processor accepts $\overline{\text{DBG}}$ as a qualified $\overline{\text{DBG}}$ is the cycle after the final $\overline{\text{TA}}$ of the read.

The second condition for data to stream from one data transfer to a second is that the first transfer be a multiple-beat transfer (requiring at least two $\overline{\text{TA}}$s). If the first data transfer is a single-beat transfer (only one $\overline{\text{TA}}$), the earliest that the processor will accept a $\overline{\text{DBG}}$ as a qualified $\overline{\text{DBG}}$ is the cycle after the single $\overline{\text{TA}}$ assertion.

Note that with these two constraints, it is possible to stream from a multiple-beat transaction into a single-beat transaction. While this may not affect overall system performance, it may simplify the design of the data bus arbiter and memory controller.

9.4.2.3 Data Tenure Reordering

The MPC7450 allows data tenures to be executed out of order with respect to their corresponding address tenures in MPX bus mode. The system must supply an index with each data bus grant to indicate which of the master's outstanding data tenures is being serviced. The data transfer index inputs, DTI[0:3], provide this function. The DTI[0:3] signals act as a pointer into the MPC7450's queue of outstanding transactions, and it indicates which transaction is to be serviced by the subsequent data tenure. The MPC7450 supports up to sixteen outstanding transactions. The depth of the data transaction queue is programmable in MSSCR0[DTQ].

The DTI[0:3] signals are connected to the system arbiter and to other potential masters (either bussed or point-to-point, depending on the system implementation). DTI[0:3] for a given data bus tenure is driven by the system arbiter on the cycle prior to the associated $\overline{\text{DBG}}$. The MPC7450 continually samples DTI[0:3] and qualifies the pointer on the subsequent cycle if the data bus grant is qualified.

A DTI[0:3] value of 0b0000 indicates that the data tenure for the oldest remaining transaction is to be serviced; a value of 0b0001 indicates that the second oldest remaining transaction is to be serviced, and so forth. A value of 0b1111 selects the sixteenth and oldest MPC7450 transaction. The system tracks the status of the MPC7450 queues by monitoring $\overline{\text{TS}}$ and $\overline{\text{DBG}}$. The MPC7450 adds a new transaction to the tail of its queue with each assertion of $\overline{\text{TS}}$ for an address and data transaction and each assertion of $\overline{\text{HIT}}$ for data-only transactions. It removes an entry if the transaction is retried with $\overline{\text{ARTRY}}$ or when it receives a qualified $\overline{\text{DBG}}$. When a transaction is removed from the queue, all transactions newer than the transaction removed shift forward.

The system must not provide an illegal DTI[0:3] value in the cycle before any qualified data bus grant when the processor has one or more valid data transactions queued. An illegal DTI[0:3] value is one that does not have any valid corresponding data transaction unless the DTI[0:3] value is 0b0000. For instance, if only two data transactions are queued, a DTI[0:3] value of 0b0010 or greater is illegal. The processor's behavior is boundedly undefined if an illegal DTI[0:3] value is detected at this time.

Data tenure reordering can be disabled by pulling all DTI[0:3] signals down. This causes the MPC7450 to always select the oldest transaction in the outstanding transaction queue.

9.4.2.4 MPX Bus Data Intervention

If the MPC7450 performs a read, a RWITM, or an RCLAIM of data that exists modified in another processor's cache, the 60x bus requires that the transaction be retried and data pushed to memory before the transaction is begun a second time. A more efficient approach, used by the MPC7450 in MPX bus mode, is to allow the data to be forwarded directly to the requesting master from the processor that has it cached. This is called data intervention. The MPC7450 performs this function through the $\overline{\text{HIT}}$/$\overline{\text{DRDY}}$ protocol and data-only transactions. Data intervention may be disabled by setting MSSCR0[EIDIS].

Note that data intervention is only allowed for full cache-line transfers; however the MPC7450 does not sample $\overline{\text{WT}}$ or $\overline{\text{CI}}$ and can assert $\overline{\text{HIT}}$ for any read type snoop. A snooping MPC7450 does not assert $\overline{\text{HIT}}$ for write-with-flush transactions.

An important implication of data intervention is that data must always be pushed with the critical data first, and the full double-word address must always be placed on the address bus. Otherwise, data could be received in the wrong order by the requesting master.

Intervention allows the latency for data that exists in another processor’s cache to be reduced from over 20 bus cycles to as low as 5 or 6 cycles, as shown in Figure 9-12. Figure 9-12 shows transfer involving different DTI indices occurring in 6 cycles. Note that this could be reduced in some systems to 5 cycles if the DTI index provided to both processors is the same for this transfer and the DTI is able to be presented in clock cycle 3.

Figure 9-12 shows a timing diagram of the best case scenario for the activation of $\overline{\text{HIT}}$ and $\overline{\text{DRDY}}$ for the MPX bus protocol. However, note that the MPC7450 is not able to assert $\overline{\text{DRDY}}$ in the same cycle as $\overline{\text{HIT}}$; it asserts $\overline{\text{DRDY}}$ sometime after $\overline{\text{HIT}}$. See Section 9.4.2.4.2, “DRDY Timing,” for more information.

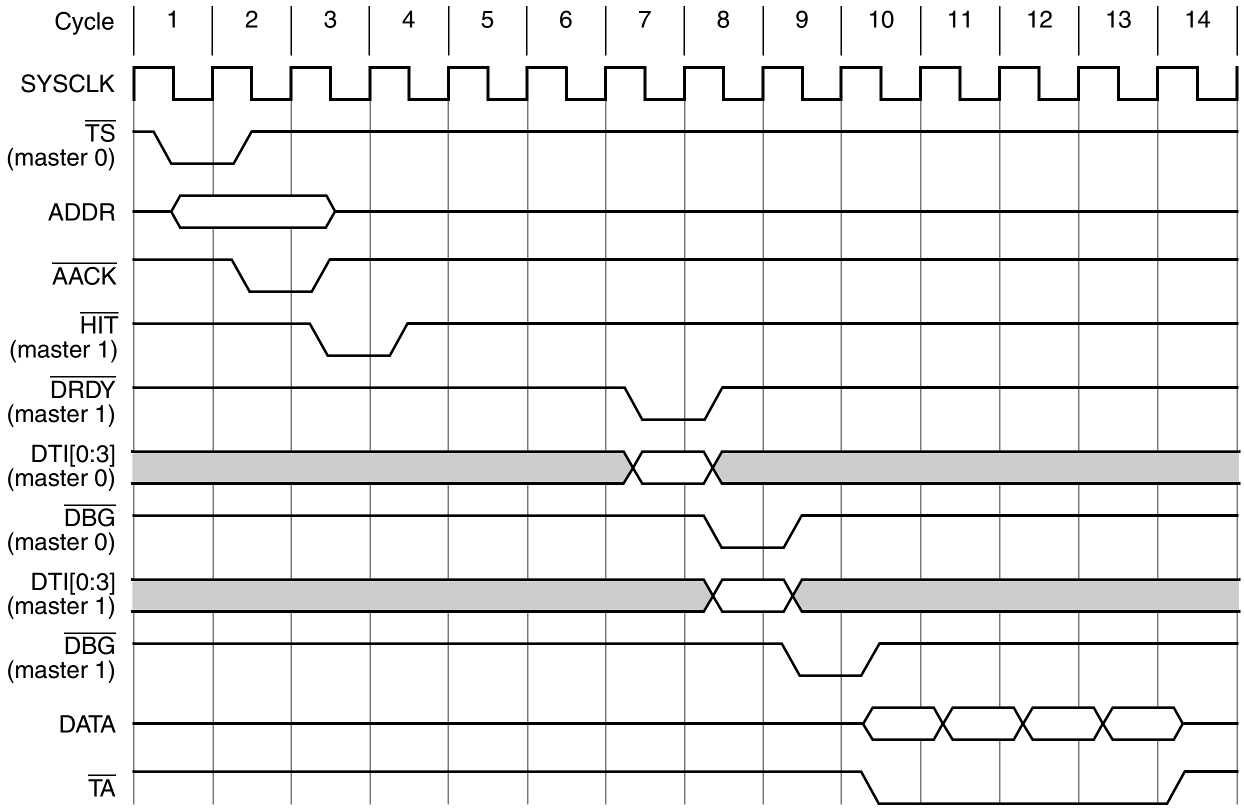


Figure 9-12. Data Intervention for Read (Atomic) and RWITM (Atomic) Using Data-Only Transfer Protocol

9.4.2.4.1 Data-Only Transaction Protocol

A data-only transaction in MPX bus mode is differentiated from a typical address and data transaction in the 60x protocol in that the data bus is explicitly requested by the snooper or slave involved in the transaction, and no new address is specified by the snooper or slave. Data-only transactions are responses by snoopers or slaves to a transaction already initiated by a master with an address tenure, so the address is already known to the system.

Data-only transactions can be issued in response to address-only transactions (for example, flush or clean), or address and data transactions (for example, read or RWITM) to which the MPC7450 asserted the $\overline{\text{HIT}}$ response in the snoop response window.

Figure 9-13 shows an example of a data-only transaction for a flush.

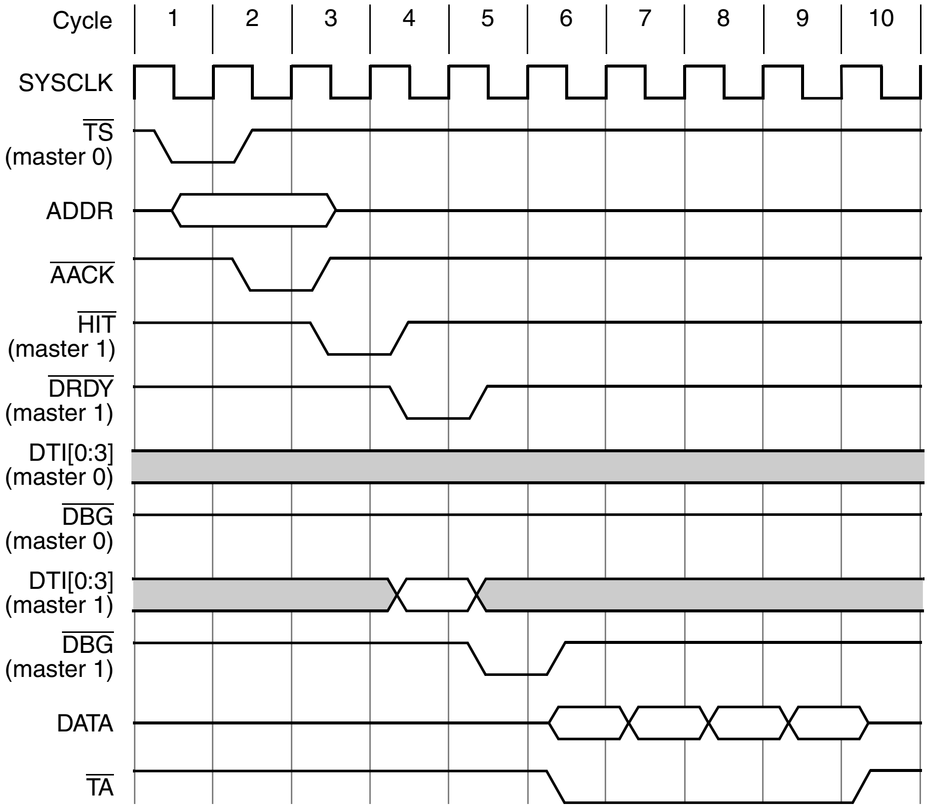


Figure 9-13. Data-Only Transaction for a Flush Operation

Address and data transactions do not need to explicitly request the data bus because the request is implicit in the transaction's address tenure. Processors that perform data-only transactions, however, request the data bus by asserting $\overline{\text{DRDY}}$ when they have the data available. The $\overline{\text{DRDY}}$ causes the arbiter to assert $\overline{\text{DBG}}$ to all devices participating in the transaction. Whichever device in the system is responsible for asserting $\overline{\text{TA}}$ then does so to complete the transaction.

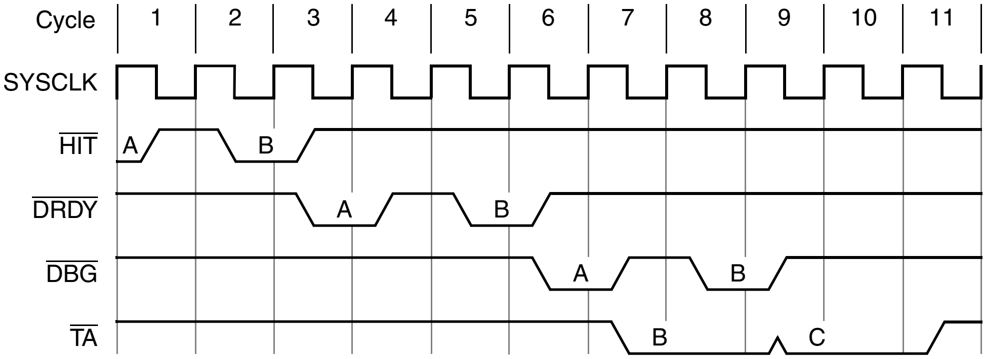
9.4.2.4.2 DRDY Timing

Although MPX bus protocol allows for HIT and DRDY to be asserted simultaneously, the MPC7450 does not assert DRDY simultaneously with HIT. The MPC7450 asserts DRDY sometime after HIT because the MPC7450 requires more cycles to get the data into the intervention queue. DRDY is a pulsed signal and must be negated on the cycle after it is asserted unless pipelining of data-only intervention requests naturally requires consecutive assertions of DRDY.

9.4.2.4.3 Pipelining of Data-Only Transactions

The MPC7450 may pipeline multiple data-only transactions. That is, it may assert HIT for additional transactions, even if it has not been able to complete pending data-only transactions for prior snoop hits. The MPC7450 may do this until its internal buffers fill up. Similarly, a device may assert DRDY for additional data-only transactions before previous data-only transactions have completed their data tenures. There is no restriction on the number of outstanding DRDY assertions. The MPC7450 can assert DRDY the same number of times as the number of outstanding HIT assertions. The MPC7450 supports 16 outstanding transactions if no DBGs have been issued for any of the queued data transactions.

Because DRDY is a pulsed signal, if DRDY is held low for multiple cycles, the system interprets this as multiple assertions of DRDY. If a device asserts DRDY when the system is not expecting a DRDY (no pending data-only transaction has been indicated), the system ignores the DRDY signal. An important example of this would be the cycle after an ARTRY. See Section 9.4.2.4.4, “Retrying Data-Only Transactions.” An example of pipelined data-only transactions is shown in Figure 9-14. (Note that Figure 9-14 assumes that data is all driven from the same source, so that data streaming is possible.)



Cycles 0, 2: The device asserts HIT for transactions, A and B.
Cycles 3 and 5: The device asserts DRDY for the first A and B respectively.
Cycle 6: DBG for transaction A issued.
Cycle 7: Data for transaction A is driven.
Cycle 8: DBG for transaction B issued.
Cycle 9: Data for transaction C is driven.

Figure 9-14. Pipelined Data-Only Transactions

9.4.2.4.4 Retrying Data-Only Transactions

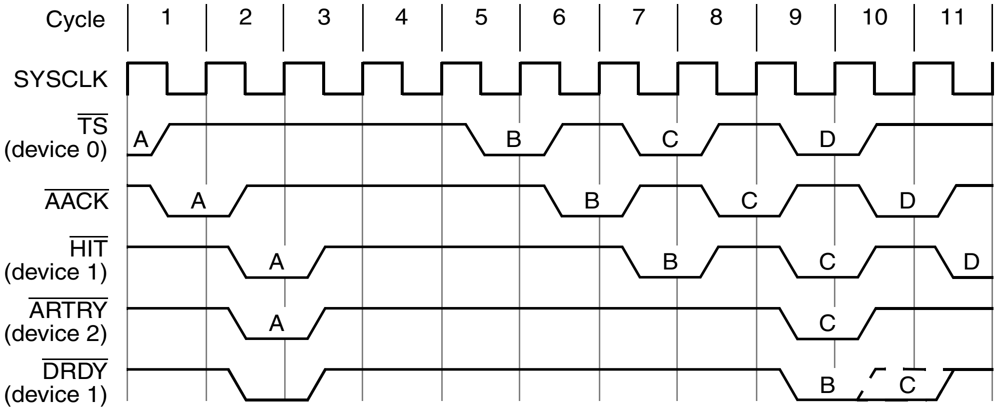
As described in Section 9.3.3.4, “Hit (HIT) Signal and Data Intervention,” it is possible for the MPC7450 to signal a data-only transaction with the HIT signal while another device asserts ARTRY. In this case, the data-only transaction must be considered retried, and the system does not expect a corresponding DRDY.

In general, if DRDY is asserted by an intervening master when the system is not expecting a DRDY, it is considered spurious and is ignored. However, if the system is expecting a DRDY for another transaction from that device, the MPC7450 does not assert DRDY after the transaction has been retried. That is, there is no ambiguity between a spurious DRDY from a retried transaction and a valid DRDY for a later transaction.

DRDY would never be asserted the cycle after HIT to avoid the spurious DRDY problem if HIT is ARTRYed. Neither the MPC7400/7410 nor the MPC7450 will ever assert a spurious DRDY because they will never assert DRDY the cycle after HIT.

Note that if data-only tenures are being pipelined, and the DRDY for a previous HIT is asserted at the same time as a new HIT and ARTRY, the DRDY must not be ignored.

Examples of retrying data-only transactions, including pipelined HITs and DRDYs, are shown in Figure 9-15.



Cycle 2: Transaction A receives an ARTRY so the HIT signal is ignored by the system.

Cycle 5: Transaction B starts. This is the earliest possible TS after an ARTRY.

Cycle 7: Transaction B receives a HIT response. DRDY is delayed.

Cycle 9: Transaction C receives a HIT and an ARTRY. The system understands that the DRDY is for transaction B and considers it valid.

Cycle 10: The MPC7450 will never assert DRDY for transaction C the cycle after HIT, regardless of ARTRY.

Cycle 11: HIT for transaction D is asserted. The bus master must not assert DRDY for transaction C at this point since the system would interpret it as the DRDY for transaction D.

Figure 9-15. Retry Examples of Data-Only Transactions

9.4.2.4.5 Ordering of Data-Only Transactions

All data-only transactions for a given MPC7450 processor must be handled in order. This does not mean that other data tenures must be handled in order with respect to data-only transactions, or that data-only transactions from different devices must maintain order relative to one another. See Section 9.4.2.3, “Data Tenure Reordering.” However, if the MPC7450 has asserted HIT for more than one transaction, the corresponding data-only transactions must be serviced in the same order because there is no defined way for the arbiter to distinguish between them except to expect them in order.

9.4.2.4.6 Snarfing

Snarfing is a term used to describe a situation where one device provides data to another target device and a third device (typically a memory or cache controller) samples the data for its own purposes. The third device is said to have snarfed the data transaction.

Snarfing is necessary when a processor uses data intervention to forward modified data to another processor because the processor performing the read operation will mark the data shared or exclusive even though the data is modified with respect to memory. The MPC7450 does not perform exclusive or shared intervention, so any data intervention will occur with modified data.

The system must snarf for interventions due to READ, RCLAIM, CLEAN, or FLUSH transactions and update memory and caches accordingly. Snarfing is not necessary for RWITM transactions serviced with data intervention because the data will be marked modified in the requesting processor’s cache.

9.4.3 MPX Bus Data Tenure Termination

Three signals are used to terminate the individual data beats of the data tenure and the data tenure itself—$\overline{\text{TA}}$ (transfer acknowledge), $\overline{\text{TEA}}$ (transfer error acknowledge), and $\overline{\text{ARTRY}}$ (address retry).

$\overline{\text{TA}}$ is used to signal normal termination of a data beat or transaction (last data beat of burst). It must always be asserted on the bus cycle coincident with the data that it is qualifying. It may be withheld by a slave for any number of clocks until valid data is ready to be supplied or accepted.

$\overline{\text{TEA}}$ is used to signal a non-recoverable error during the data transaction. It may be asserted as early as the last cycle of the snoop response window or as late as the cycle of the last or only $\overline{\text{TA}}$. The assertion of $\overline{\text{TEA}}$ terminates the data tenure immediately even if in the middle of a burst; however, it will not prevent incorrect data from being written into the MPC7450’s caches or registers. The assertion of $\overline{\text{TEA}}$ initiates either a machine check exception or a checkstop condition based on the setting of MSR[ME].

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. (It may not be due to address pipelining.) If $\overline{\text{ARTRY}}$ is used, the earliest allowable assertion of $\overline{\text{TA}}$ to the processor is directly dependent on the latest possible assertion of $\overline{\text{ARTRY}}$ to MPC7450. See Section 9.3.3, “MPX Bus Address Tenure Termination,” for more information.

If the $\overline{\text{ARTRY}}$ and $\overline{\text{TEA}}$ signals are asserted in the same clock, the $\overline{\text{ARTRY}}$ signal takes precedence and the $\overline{\text{TEA}}$ signal is ignored. This means that the transaction is repeated until the $\overline{\text{ARTRY}}$ condition is resolved.

9.4.3.1 Normal Single-Beat Transfer Termination

Normal termination of a single-beat data read operation occurs when $\overline{\text{TA}}$ is asserted by a responding slave. The $\overline{\text{TEA}}$ signal must remain negated during the transfer (see Figure 9-16).

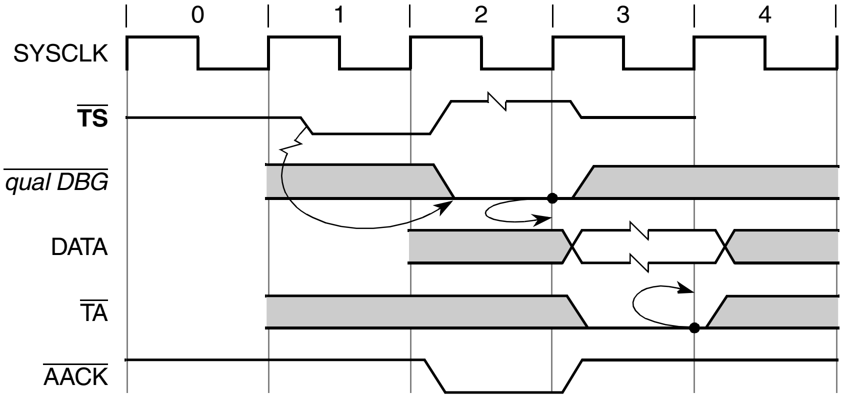


Figure 9-16. Normal Single-Beat Read Termination

Figure 9-17 shows a single-beat write operation with a normal termination.

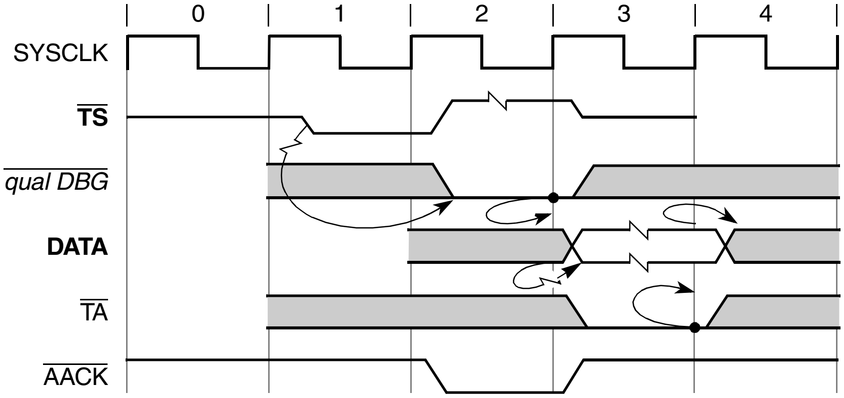


Figure 9-17. Normal Single-Beat Write Termination

9.4.3.2 Normal Burst Transfer Termination

Normal termination of a burst transfer occurs when $\overline{\text{TA}}$ is asserted for four bus clock cycles, as shown in Figure 9-18. 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, $\overline{\text{TEA}}$ must remain negated during the transfer. For write bursts, $\overline{\text{TEA}}$ must remain negated for a successful transfer.

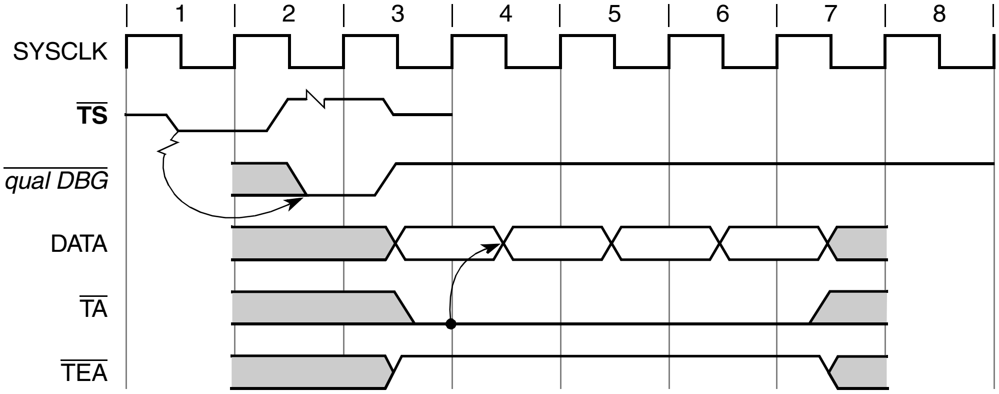


Figure 9-18. Normal Burst Transaction

Figure 9-19 shows the effect of using $\overline{\text{TA}}$ to pace the data transfer rate. Notice that in bus clock cycle 5 of Figure 9-19, $\overline{\text{TA}}$ is negated for the second data beat. The MPC7450 data pipeline does not proceed until bus clock cycle 6 when $\overline{\text{TA}}$ is reasserted.

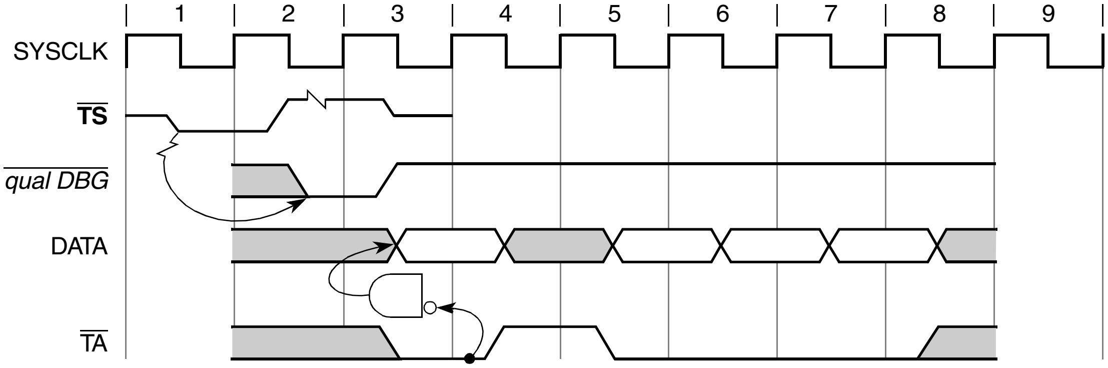


Figure 9-19. Read Burst with $\overline{\text{TA}}$ Wait States

9.4.3.3 Data Transfer Termination Due to a Bus Error

The $\overline{\text{TEA}}$ signal indicates that a bus error has occurred during a data tenure. Asserting $\overline{\text{TEA}}$ to the MPC7450 terminates the transaction (that is, subsequent assertions of $\overline{\text{TA}}$ are ignored).

Assertion of the $\overline{\text{TEA}}$ signal causes a machine check exception (or a checkstop condition within the MPC7450). For more information, see Section 4.6.2, “Machine Check Exception (0x00200).” Note that the MPC7450 does not implement a synchronous error capability for memory accesses. This means that the exception instruction pointer saved into the SRR0 register 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). Also note that assertion of $\overline{\text{TEA}}$ does not invalidate data entering the registers or the cache. Also, the address corresponding to the access that caused $\overline{\text{TEA}}$ to be asserted is not latched by the MPC7450. To recover, the exception handler must remedy the cause of the $\overline{\text{TEA}}$, or the MPC7450 must be reset; therefore, this function should only be used to indicate fatal system conditions to the processor (such as parity or uncorrectable ECC errors).

After the MPC7450 has committed to run a transaction, that transaction must eventually complete. Address retry causes the transaction to be restarted; TA wait states delay termination of individual data beats. Eventually, however, the system must either terminate the transaction or assert the TEA signal. For this reason, care must be taken to check for the end of physical memory and the location of certain system facilities to avoid memory accesses that result in the assertion of TEA.

9.5 60x Bus Protocol

The MPC7450 implements a subset of the 60x bus protocol. Note that although this protocol is implemented by the MPC603-, MPC604-, MPC750-, and MPC7400-families of processors, it is referenced as the 60x bus interface.

As described in Section 9.1, “MPC7450 System Interface Overview,” the MPC7450 samples the BMODE0 signal at the negation of HRESET to determine which bus protocol to use. If BMODE0 is negated at the negation of HRESET, then the MPC7450 uses the 60x bus protocol subset.

The MPC7450 will burst out of reset in the 60x bus protocol.

9.5.1 60x Bus Pipelining

The 60x bus 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. In a pipelined implementation, data bus tenures are kept in strict order with respect to address tenures.

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 multimaster 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 external arbiter must control the pipeline depth and synchronization between masters and slaves. The design of the external arbiter affects pipelining by regulating address bus grant (BG), data bus grant (DBG), and address acknowledge (AACK) signals. For example, a one-level pipeline is enabled by asserting 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. The MPC7450 can pipeline up to 16 address tenures before starting a data tenure.

In a pipelined implementation of the 60x bus protocol, data bus tenures are kept in strict order with respect to address tenures; there is no provision for data tenure reordering as in MPX bus mode.

9.6 60x Bus Address Tenure

As with the MPX bus protocol, the 60x bus protocol uses a three phase address tenure, consisting of address bus arbitration, address transfer, and address transfer termination. However, the 60x bus protocol requires a turn-around cycle between each address tenure. Therefore, in 60x bus mode, the MPC7450 does not support MPX bus style address streaming and requires a dead cycle between address tenures (even if those address tenures are from the same processor).

9.6.1 60x Bus Address Bus Arbitration

The elimination of $\overline{\text{ABB}}$ from the interface puts more responsibility on the system arbiter. Arbiter designs must ensure that no more than one address bus master can be granted the bus at one time (that is, bus grants must be mutually exclusive).

In 60x bus mode, when the MPC7450 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 9-20. The external arbiter must grant master-elect status to the potential master by asserting the bus grant ($\overline{\text{BG}}$) signal when the bus is idle.

9.6.1.1 Qualified Bus Grant in 60x Bus Mode

The qualified bus grant equation for 60x bus mode is as follows:

```
Qualified Bus Grant = BG & ¬ARTRY & ¬TS & ¬(latched state variables)
```

where “latched state variables” include latched $\overline{\text{ARTRY}}$. Thus, a qualified bus grant occurs when $\overline{\text{BG}}$ is asserted, $\overline{\text{ARTRY}}$ is not asserted in the current or in the preceding cycle, and $\overline{\text{TS}}$ is not asserted by this or any other processor.

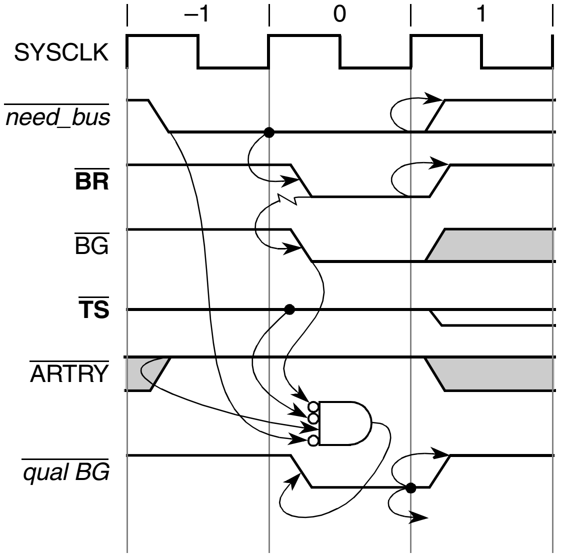


Figure 9-20. 60x Address Bus Arbitration–Non-Parked Case

9.6.1.2 60x Address Bus Parking

From the bus master's perspective, bus parking for the 60x interface is the same as for the MPX interface. If a master needs the bus and receives a bus grant and all qualifying conditions are met, the master may immediately assume control of the address bus.

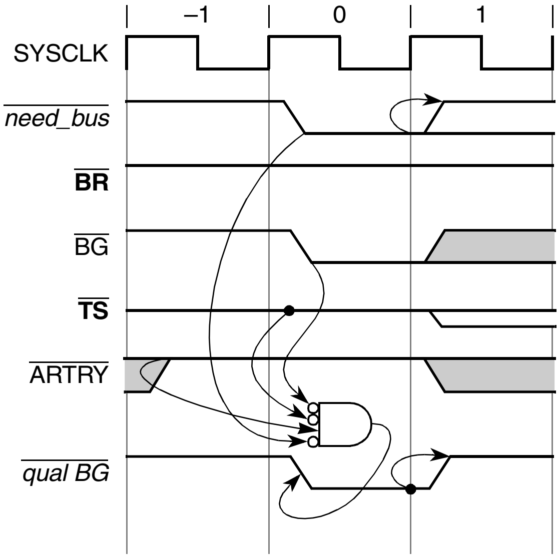


Figure 9-21. 60x Address Bus Arbitration–Parked Case

9.6.2 60x Bus Address Transfer

The 60x bus protocol requires a turn-around cycle between each address tenure. This implies that 60x address tenures must take at least three bus clock cycles (because the system can provide $\overline{\text{AACK}}$ no earlier than the cycle following the assertions of $\overline{\text{TS}}$).

9.6.2.1 60x Address Bus Driven Mode

Address bus driven mode provides for improved electrical characteristics on the address and attributes signals by reducing the time that these signals are not actively driven. The $\overline{\text{BMODE0}}$ signal is used to select address bus driven mode after $\overline{\text{HRESET}}$ is negated. If $\overline{\text{BMODE0}}$ is asserted after $\overline{\text{HRESET}}$ is negated, address bus driven mode is selected; if $\overline{\text{BMODE0}}$ is negated after $\overline{\text{HRESET}}$ is negated, normal address bus driving mode (address bus not always driven) is selected. The read-only ABD bit in MSSCR0 indicates whether the MPC7450 is in address bus driven mode.

9.6.2.2 60x Address Bus Parity

In 60x bus mode, the MPC7450 supports 36-bit address bus and five parity signals. Address parity generation and reporting in 60x bus mode operates identically to that in MPX bus mode.

9.6.2.3 60x Address Transfer Attributes

Transfer attribute signals such as TT[0:4], TSIZ[0:2], ($\overline{\text{TBST}}$), ($\overline{\text{WT}}$), ($\overline{\text{CI}}$), and ($\overline{\text{GBL}}$) used by the 60x bus are the same as are used by the MPX bus with the following exceptions:

- The MPX bus read claim (RCLAIM) transfer type (TT[0:4] = 0b01111) is not supported in 60x bus mode. Therefore, there is no distinct transaction type to identify the touch-for-store instructions (**dcbtst**, **dstst**, and **dststt**). In 60x bus mode, the touch-for-store instructions use the read transfer type (TT[0:4] = 0b01010), the same as a load miss.
- There are fewer defined encodings for the $\overline{\text{TBST}}$ and TSIZ[0:2] signals in 60x bus mode.

9.6.2.3.1 60x Transfer Size (TSIZ[0:2]) and Transfer Burst ($\overline{\text{TBST}}$) Signals

In 60x bus mode, the MPC7450 only supports bursting for cache block transfers (4 double words). Unlike the 60x bus mode of the MPC7400 and MPC7410, the MPC7450 does not automatically break caching-inhibited or write-though AltiVec loads and stores of 16-bytes into two 8-byte transfers. For the MPC7450, caching-inhibited AltiVec loads or stores and write-through AltiVec stores cause alignment exceptions in 60x bus mode.

The 60x bus protocol does not support a 16-byte transaction, but caching-inhibited instruction fetches are internally 16-bytes. In 60x bus mode, all caching-inhibited instruction fetches are performed on the bus as caching-allowed, 32-byte burst transactions. No forwarding of the critical 16 bytes is done, and the data is not reloaded into any MPC7450 cache.

Table 9-8 defines the $\overline{\text{TBST}}$ and TSIZ[0:2] encodings used by the MPC7450 in 60x bus mode.

Table 9-8. $\overline{\text{TBST}}$ and TSIZ[0:2] Encodings in 60x Bus Mode

$\overline{\text{TBST}}$	TSIZ0	TSIZ1	TSIZ2	Transfer Size[1]
Asserted	0	0	0	Undefined
Asserted	0	0	1	Undefined
Asserted	0	1	0	4 double-word burst
Asserted	0	1	1	Undefined
Asserted	1	0	0	Undefined
Asserted	1	0	1	Undefined
Asserted	1	1	0	Undefined
Asserted	1	1	1	Undefined
Negated	0	0	0	8 bytes
Negated	0	0	1	1 byte
Negated	0	1	0	2 bytes
Negated	0	1	1	3 bytes
Negated	1	0	0	4 bytes

Table 9-8. $\overline{\text{TBST}}$ and TSIZ[0:2] Encodings in 60x Bus Mode

$\overline{\text{TBST}}$	TSIZ0	TSIZ1	TSIZ2	Transfer Size[1]
Negated	1	0	1	5 bytes [2]
Negated	1	1	0	6 bytes [2]
Negated	1	1	1	7 bytes [2]

[1] 3-byte transfers may be requested by the MPC7450 starting at any byte address within the double word from byte address 0 to byte address 5.
4-byte transfers may be requested by MPC7450 starting at any byte address within the double word from byte address 0 to byte address 4.

[2] Although defined, the MPC7450 never generates a transaction of this size.

9.6.2.4 Aligned and Misaligned Transfers

Performance on misaligned transfers may be substantially less than on aligned transfers, and it is recommended that software attempt to align code and data if possible. See Section 9.3.2.6, “Effect of Alignment in Data Transfers,” for a detailed description of alignment considerations for transactions in 60x and MPX bus modes. Note that the MPC7450 is not compatible with the MPC107 bridge device in little-endian mode if misaligned data is accessed.

9.6.3 60x Bus Address Transfer Termination

In 60x bus mode, addresses are terminated as they are in MPX bus mode, except that the 60x bus protocol requires a turn-around cycle on the bus before a new address tenure may begin. This implies that 60x address tenures must take at least three bus clock cycles (because the system can provide $\overline{\text{AACK}}$ no earlier than the cycle following the assertions of $\overline{\text{TS}}$).

9.6.3.1 Snoop Response and $\overline{\text{SHD}}$ Signal

The MPC7450 asserts the $\overline{\text{SHD0}}$ signal as an output coincident with the $\overline{\text{ARTRY}}$ output signal if the cache block that caused a snoop hit is pushed as the processor’s next address transaction.

The shared $\overline{\text{SHD0}}$ signal functions similarly to $\overline{\text{SHD0}}$ and $\overline{\text{SHD1}}$ in MPX bus mode. Because the 60x bus protocol does not allow a master to drive a new address tenure every other cycle as does the MPX protocol, only one snoop response signal, $\overline{\text{SHD0}}$, is necessary.

9.7 60x Bus Data Tenure

This section describes the data bus arbitration, transfer, and termination phases defined by the 60x bus protocol used by the MPC7450.

9.7.1 60x Bus Data Bus Arbitration

The MPC7450 data bus uses the $\overline{\text{DBG}}$ signal and arbiter logic in data arbitration. Additionally, the combination of $\overline{\text{TS}}$ and TT[0:4] provides information about the data bus request to external logic.

As a result of address pipelining, the MPC7450 may have up to 16 data tenures queued to be performed when it receives a qualified $\overline{\text{DBG}}$. The data tenures must be performed in the same order in which their address tenures were performed. The $\overline{\text{DBWO}}$ input signal is not implemented on the MPC7450, so the limited out of order capability available in the 60x bus protocol is not supported.

If the MPC7450 has any data tenures to perform, it always accepts data bus mastership to perform a data tenure when it recognizes a qualified $\overline{\text{DBG}}$.

9.7.1.1 Qualified Data Bus Grant in 60x Bus Mode

When the MPC7450 is operating in 60x bus mode, a qualified data bus grant occurs when the following conditions are satisfied:

- $\overline{\text{DBG}}$ is asserted.
- $\overline{\text{ARTRY}}$ is not asserted in the address retry window for the address phase of this transaction.
- The processor is ready to begin a data transaction.
- The processor is not already using the data bus.

Note that in some systems, it may be possible to have a qualified data bus grant as early as the clock cycle when $\overline{\text{TS}}$ is asserted.

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. Because the MPC7450 can pipeline transactions, there may be an outstanding data bus transaction when a new address transaction is retried. In this case, the MPC7450 becomes the data bus master to complete the previous transaction and the $\overline{\text{ARTRY}}$ does not affect the data tenure.

The $\overline{\text{DBB}}$ signal is not implemented on the MPC7450. As is the case with $\overline{\text{ABB}}$, the elimination of $\overline{\text{DBB}}$ from the MPC7450 interface puts more responsibility on the data bus arbiter as it must synthesize an internal version of *dbb* to assist it in tracking the start and end of the data tenure. Because $\overline{\text{DBB}}$ 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.

9.7.2 60x Bus Data Transfers

Data and data parity signals operate the same in 60x bus mode as they do in MPX bus mode except for the following exceptions:

- The MPC7450 does not support data tenure reordering in 60x bus mode; therefore, DTI[0:3] must be pulled down in 60x bus mode.
- The MPC7450 does not support data streaming from one data tenure to the next in 60x bus mode.
- The MPC7450 does not support data intervention in 60x bus mode. Non-intervention burst write transfers are the only burst writes performed in 60x bus mode and are always performed zero double word first.

As with MPX bus mode, data must never be transferred before the last cycle of the address retry window (that is, valid data must never precede a possible $\overline{\text{ARTRY}}$ for that transaction).

9.7.3 60x Bus Data Tenure Termination

The 60x bus protocol defines four signals to terminate data bus transactions—$\overline{\text{TA}}$, $\overline{\text{DRTRY}}$ (data retry), $\overline{\text{TEA}}$ (transfer error acknowledge), and $\overline{\text{ARTRY}}$. The MPC7450 does not implement the $\overline{\text{DRTRY}}$ signal; therefore, the 60x interface on the MPC7450 is always operating in the higher-performance, no-$\overline{\text{DRTRY}}$ mode. $\overline{\text{TA}}$, $\overline{\text{TEA}}$, and $\overline{\text{ARTRY}}$ function the same in 60x bus mode as they do in MPX bus mode.

If the $\overline{\text{ARTRY}}$ and $\overline{\text{TEA}}$ signals are asserted in the same clock, the $\overline{\text{ARTRY}}$ signal takes precedence and the $\overline{\text{TEA}}$ signal is ignored. This means that the transaction is repeated until the $\overline{\text{ARTRY}}$ condition is resolved.

Because the MPC7450 has no $\overline{\text{DRTRY}}$, the assertion of $\overline{\text{ARTRY}}$ by a snooping device must occur prior to or coincident with the first assertion of $\overline{\text{TA}}$ to the MPC7450; assertion of $\overline{\text{ARTRY}}$ must never occur on the cycle after the first assertion of $\overline{\text{TA}}$.

9.8 60x Bus Timing Examples

This section shows timing diagrams for various scenarios using the 60x bus interface. Figure 9-22 illustrates the fastest single-beat reads possible for the MPC7450 and 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 delays the third address tenure.

Note that all bidirectional signals are released to high-impedance between bus tenures.

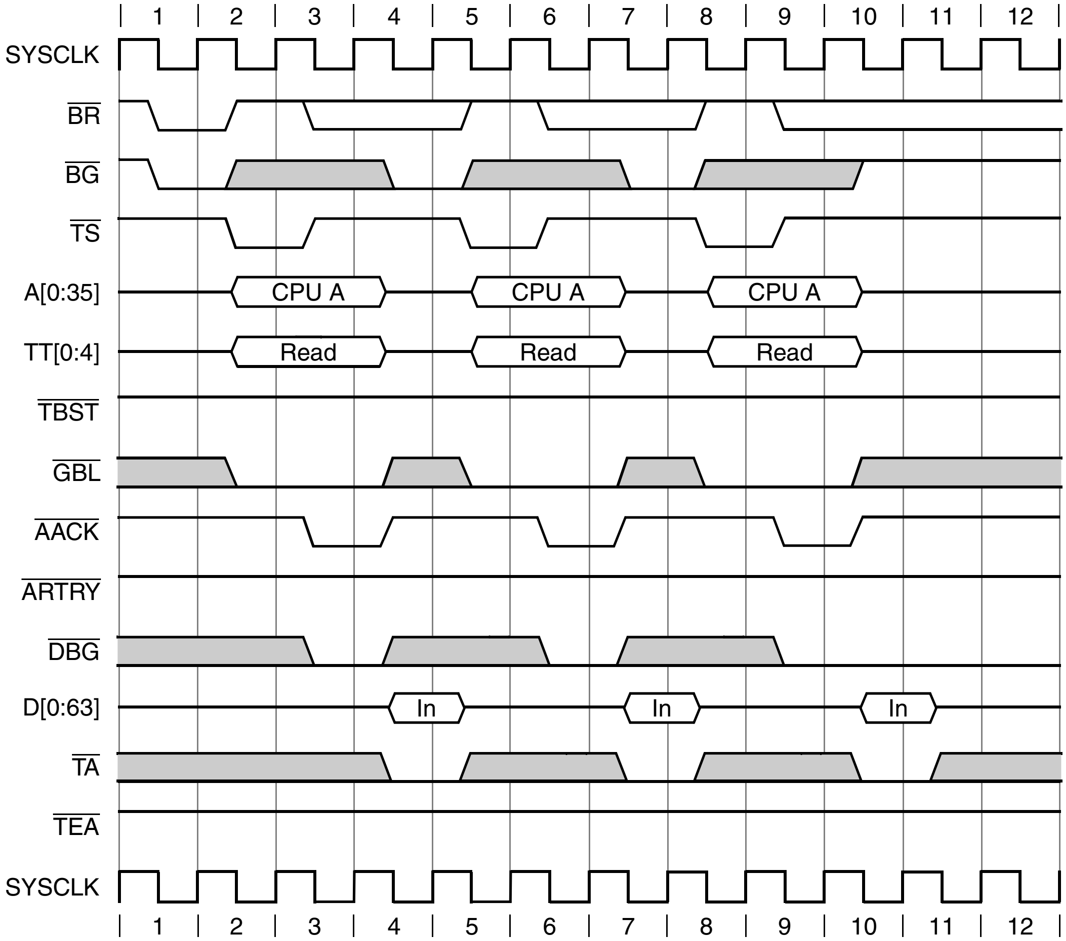


Figure 9-22. Fastest Single-Beat Reads

Figure 9-23 illustrates the fastest single-beat writes supported by the MPC7450. All bidirectional signals are released to high-impedance between bus tenures.

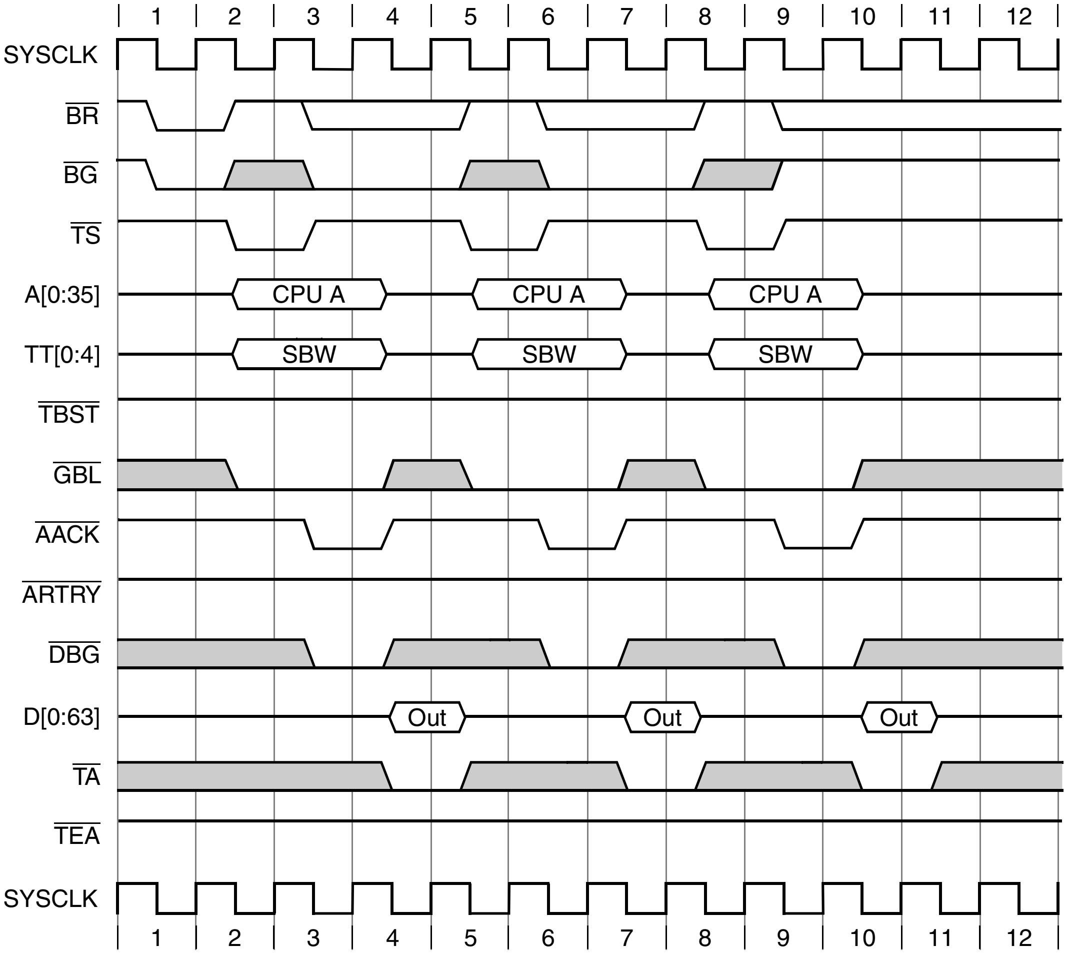


Figure 9-23. Fastest Single-Beat Writes

Figure 9-24 shows two ways that single-beat reads are delayed:

- The $\overline{\text{TA}}$ signal is negated to insert wait states in clock cycles 4 and 5.
- For the second access, $\overline{\text{DBG}}$ is delayed until clock cycle 8 (could have been asserted in clock cycle 7).

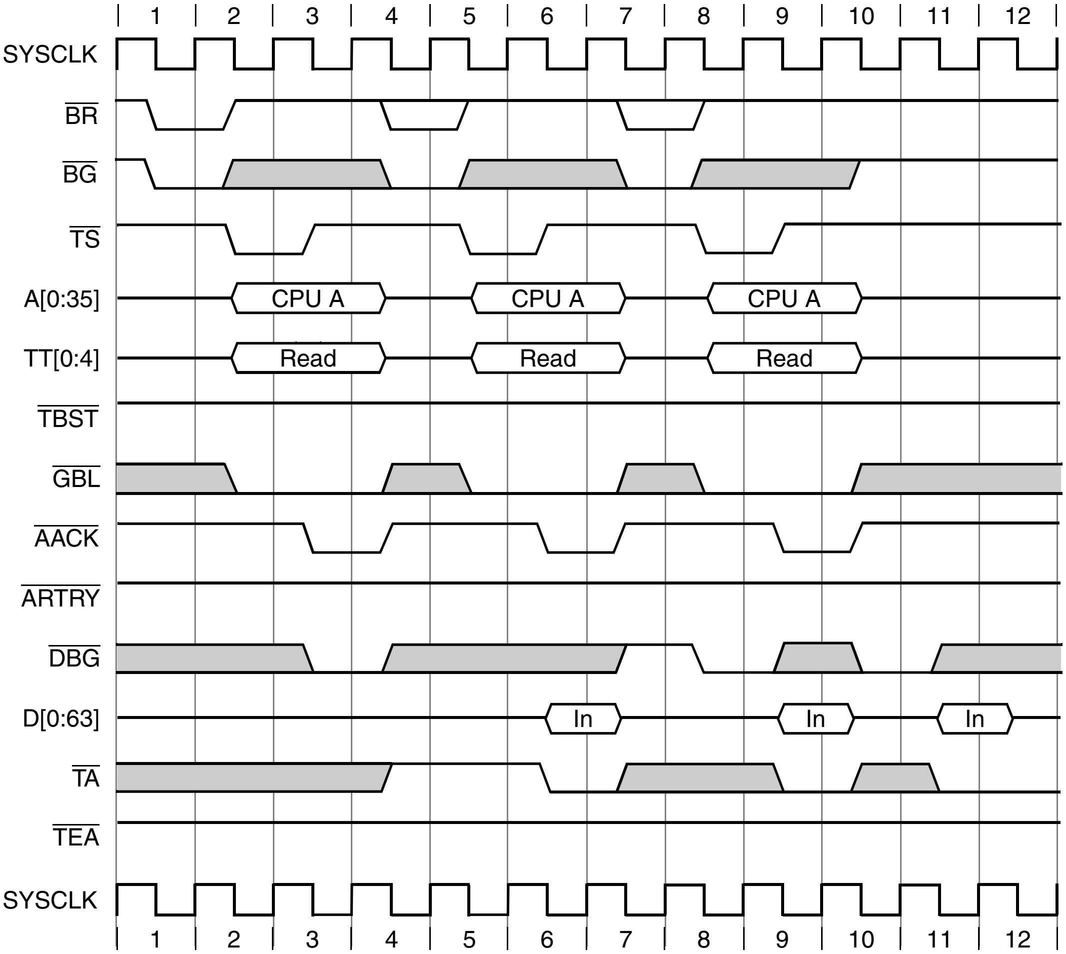


Figure 9-24. Single-Beat Reads Showing Data-Delay Controls

Figure 9-25 shows data-delay controls in a single-beat write operation. Note that all bidirectional signals are released to high-impedance 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 4 and 5.
- In clock 7, $\overline{\text{DBG}}$ is held negated, which delays the start of the data tenure.

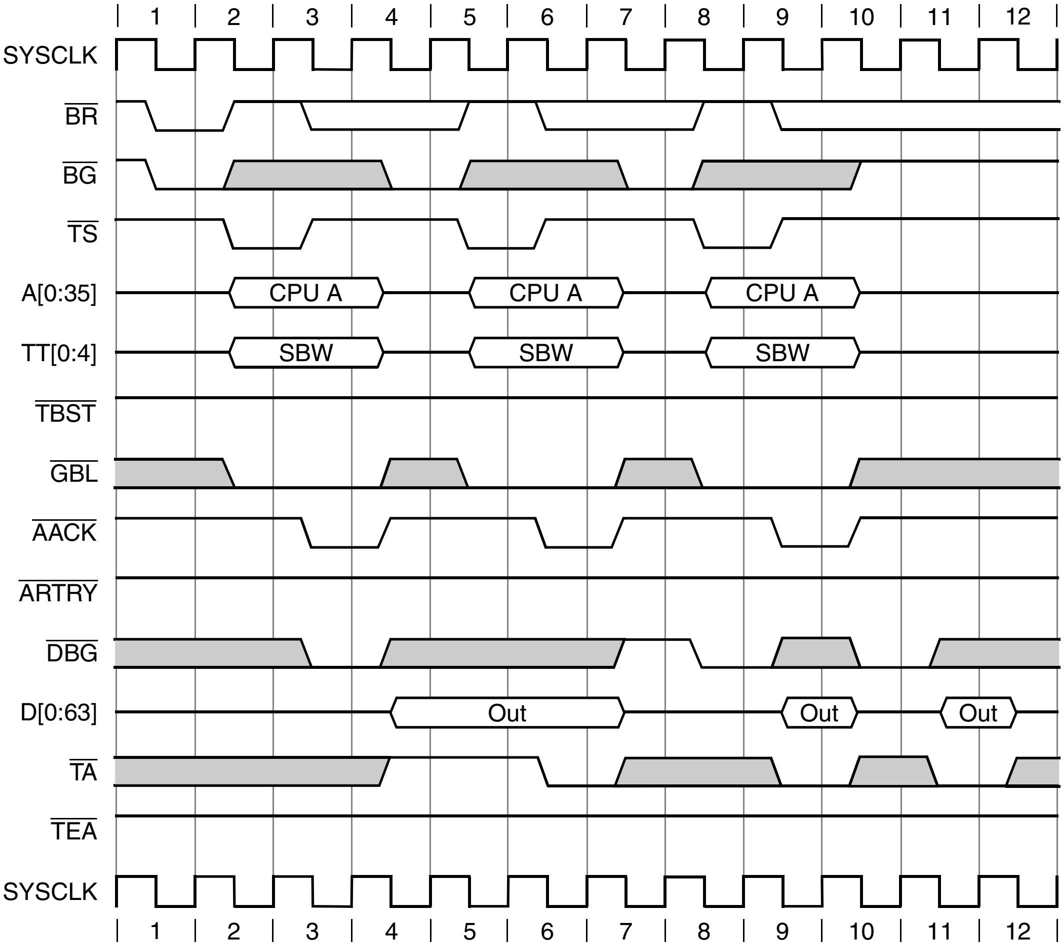


Figure 9-25. Single-Beat Writes Showing Data Delay Controls

Figure 9-26 shows the use of data-delay controls with burst transfers. Note that all bidirectional signals are released to high-impedance between bus tenures. Burst transfers are delayed in the following ways:

- The first data beat of bursted read data (clock 4) is the critical double word.
- The write burst shows the use of $\overline{\text{TA}}$ signal negation to delay the third data beat.

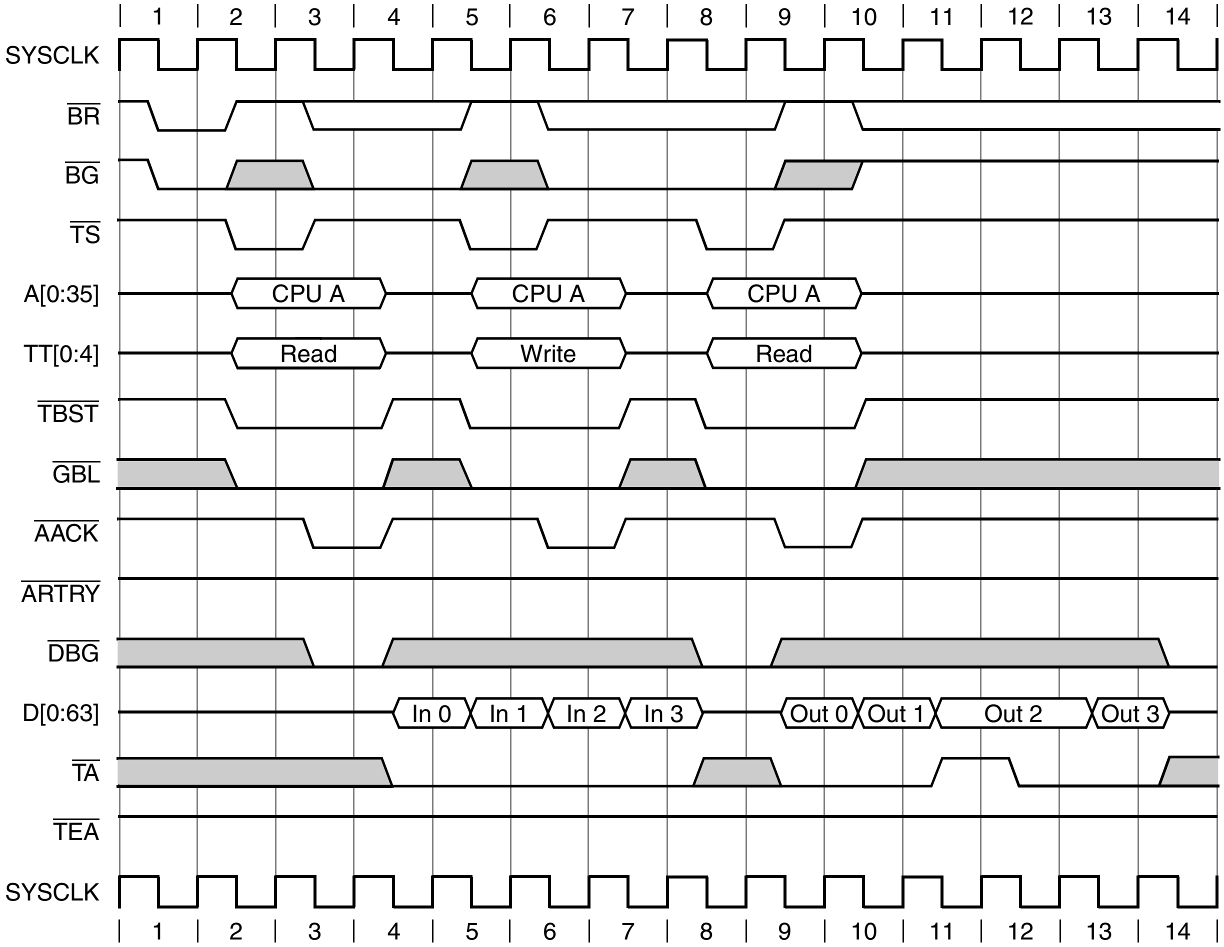


Figure 9-26. Burst Transfers with Data Delay Controls

Figure 9-27 shows the use of the $\overline{\text{TEA}}$ signal. Note that all bidirectional signals are released to high-impedance between bus tenures. Note the following:

- The first data beat of the read burst (in clock 4) is the critical quad word.
- The $\overline{\text{TEA}}$ signal truncates the burst write transfer on the third data beat.
- The $\overline{\text{TEA}}$ eventually causes the MPC7450 to take an exception.

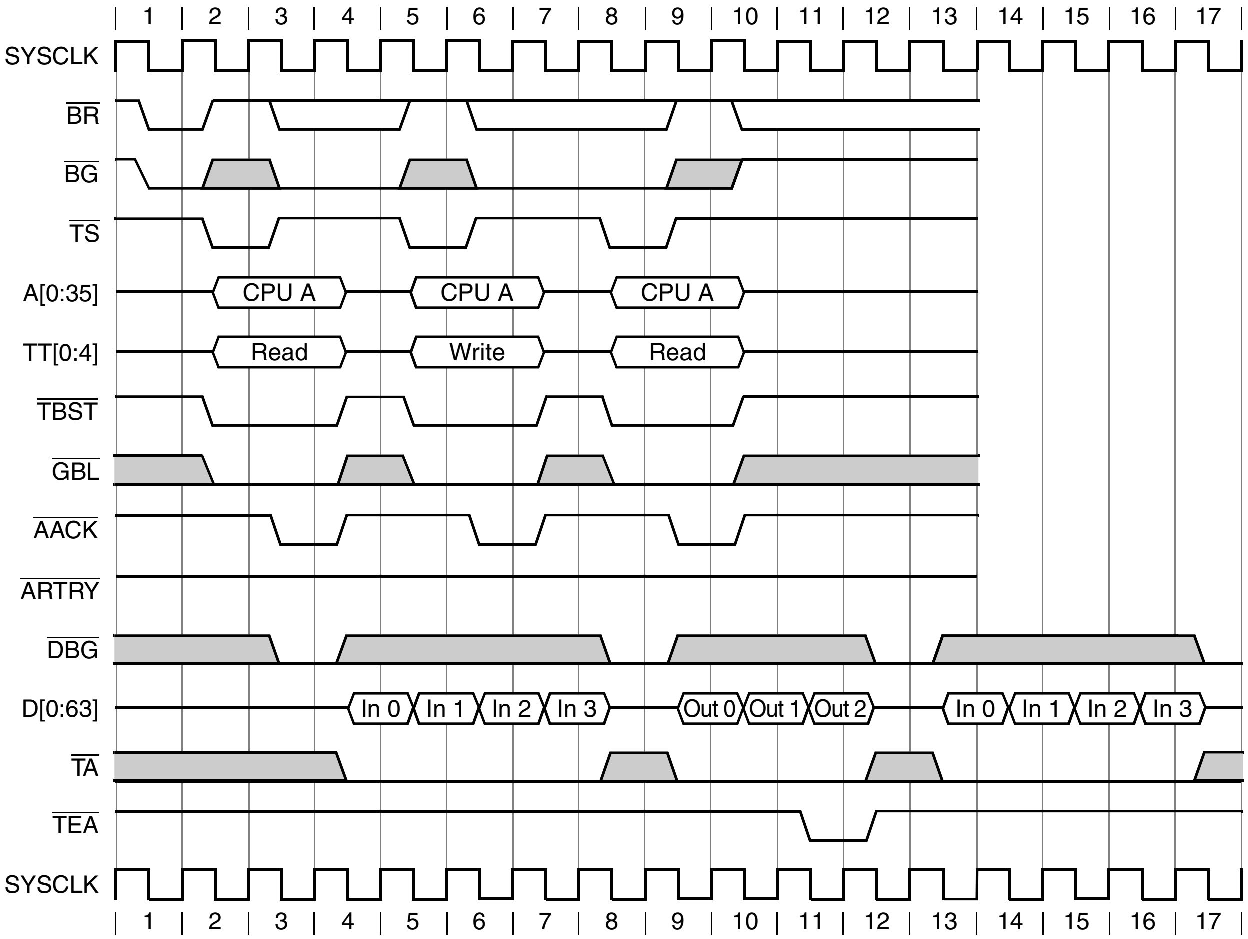


Figure 9-27. Use of Transfer Error Acknowledge ($\overline{\text{TEA}}$)

9.9 Reset, Interrupt, Checkstop, and Power Management Signal Interactions

This section describes the hard and soft reset input signals, external interrupts, checkstop operations, and power management signal interactions. See Chapter 4, “Exceptions,” and Chapter 8, “Signal Descriptions,” for more information on the exceptions caused by these signals and for signal descriptions, respectively.

9.9.1 Reset Inputs

The MPC7450 has two reset inputs, described as follows:

- $\overline{\text{HRESET}}$ (hard reset)—The $\overline{\text{HRESET}}$ signal is used for power-on reset sequences, or for situations in which the MPC7450 must go through the entire cold start sequence of internal hardware initialization. The MPC7450 will initiate burst transactions after power-on reset in 60x bus mode.
- $\overline{\text{SRESET}}$ (soft reset)—The soft reset input provides warm reset capability. This input can be used to avoid forcing the MPC7450 to complete the cold start sequence.

When either reset input negates, the processor attempts to fetch code from the system reset exception vector. The vector is located at offset 0x00100 from the exception prefix (MSR[IP]). The MSR[IP] bit is set when $\overline{\text{HRESET}}$ negates.

9.9.2 External Interrupts

The external interrupt input signals ($\overline{\text{INT}}$, $\overline{\text{SMI}}$ and $\overline{\text{MCP}}$) of the MPC7450 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] and the HID1[EMCP] bits are set.

9.9.3 Checkstops

The MPC7450 has two checkstop input signals—$\overline{\text{CKSTP\_IN}}$ (nonmaskable) and $\overline{\text{MCP}}$ (enabled when MSR[ME] is cleared and HID1[EMCP] is set)—and a checkstop output ($\overline{\text{CKSTP\_OUT}}$) signal. If $\overline{\text{CKSTP\_IN}}$ or $\overline{\text{MCP}}$ is asserted, the MPC7450 halts operations by gating off all internal clocks. The MPC7450 asserts $\overline{\text{CKSTP\_OUT}}$ if $\overline{\text{CKSTP\_IN}}$ is asserted.

If $\overline{\text{CKSTP\_OUT}}$ is asserted by the MPC7450, it has entered the checkstop state, and processing has halted internally. The $\overline{\text{CKSTP\_OUT}}$ signal can be asserted for various reasons including receiving a $\overline{\text{TEA}}$ signal and detection of external parity errors. For more information about checkstop state, see Section 4.6.2.2, “Checkstop State (MSR[ME] = 0).”

All non-test output signals are disabled during a checkstop.

9.9.4 Power Management Signals

This section describes the MPC7450's support for power management. The system quiescence control signals ($\overline{\text{QREQ}}$ and $\overline{\text{QACK}}$) allow the processor to enter the nap or sleep low-power states and bring bus activity to a quiescent state in an orderly fashion.

Prior to entering the nap or sleep-power state, the MPC7450 asserts the $\overline{\text{QREQ}}$ 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 $\overline{\text{QACK}}$ signal. At this time the MPC7450 may enter the nap or sleep power-saving state. When the MPC7450 is in the quiescent state, it stops snooping bus activity.

While the MPC7450 is in the nap state, the system power controller can enable snooping by the MPC7450 by negating the $\overline{\text{QACK}}$ signal for at least eight bus clock cycles, after which the MPC7450 is capable of snooping bus transactions. The reassertion of $\overline{\text{QACK}}$ following the snoop transactions causes the MPC7450 to reenter the nap power state. See Chapter 10, “Power and Thermal Management,” for more information on the power-saving modes of the MPC7450.

Once the MPC7450 has made a request to enter the nap power-saving state, the $\overline{\text{QREQ}}$ signal may be negated on any clock cycle to service an internal interrupt (such as a decrementer or time base exception). The $\overline{\text{TS}}$ for the exception vector fetch can occur as early as the clock cycle that $\overline{\text{QREQ}}$ is negated.

9.10 IEEE 1149.1a-1993 Compliant Interface

The MPC7450 boundary-scan interface is a fully-compliant implementation of the IEEE 1149.1a-1993 standard. This section briefly describes the MPC7450 IEEE 1149.1a-1993 (JTAG) interface. See Section 8.4.6, “IEEE 1149.1a-1993 (JTAG) Interface Description,” for more information on the function of the JTAG signals.

9.10.1 JTAG/COP Interface

The MPC7450 has extensive on-chip test capability including the following:

- Debug control/observation (COP)
- Boundary scan (standard IEEE 1149.1a-1993 (JTAG) compliant interface)
- Support for manufacturing test
- Support for the following standard JTAG instructions:
  - BYPASS
  - EXTEST
  - SAMPLE/PRELOAD
  - CLAMP
  - HIGHZ

The COP and boundary-scan logic are not used under typical operating conditions. Detailed discussion of the MPC7450 test functions is beyond the scope of this document; however, sufficient information has been provided to allow the system designer to disable the test functions that would impede normal operation.

The JTAG/COP interface is shown in Figure 9-28. For more information, refer to *IEEE Standard Test Access Port and Boundary Scan Architecture IEEE STD 1149-1a-1993*.

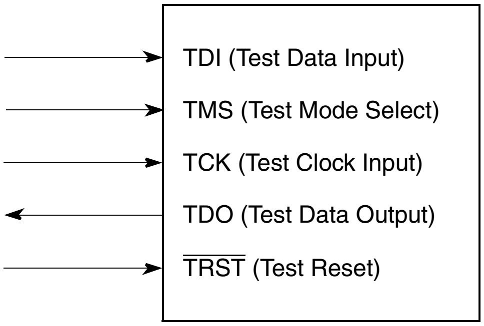


Figure 9-28. IEEE 1149.1a-1993 Compliant Boundary-Scan Interface

Chapter 10

Power and Thermal Management

The MPC7450 is designed for low-power operation. It provides both automatic and program-controlled power reduction modes. The instruction cache throttling mechanism allows on-chip thermal measurement by reducing the instruction dispatch rate. When used with dynamic power management, instruction cache throttling provides the system designer with a flexible way to control device temperature while allowing the processor to continue operating.

10.1 Dynamic Power Management

If an MPC7450 functional unit is idle, it automatically goes into a low-power mode. This mode does not affect operational performance. Dynamic power management automatically supplies or withholds power to execution units individually, based upon the contents of the instruction stream.

CMOS circuits consume negligible power when they are not switching, so stopping the clock to an execution unit effectively eliminates its power consumption. Each MPC7450 execution unit has an independent clock input that is controlled automatically on a clock-by-clock basis; for example, clocking is withheld from the floating-point unit if no floating-point instructions are being executed. The operation of dynamic power management is transparent to software or any external hardware.

Dynamic power management is enabled by setting HID0[DPM], as described in Section 2.2.5.1, “Hardware Implementation-Dependent Register 0 (HID0).”

10.2 Programmable Power Mode

The following three power saving modes are available to the system:

- Nap—Instruction fetching is halted. Only those clocks for time base, decrementer, and JTAG logic remain running. The MPC7450 goes into the doze state to snoop memory operations on the bus and then back to nap using a $\overline{\text{QREQ}}$/$\overline{\text{QACK}}$ processor-system handshake protocol.
- Sleep—Power consumption is further reduced by disabling bus snooping, leaving only the PLL in a locked and running state. All internal functional units are disabled.
- Deep sleep—When the MPC7450 is in sleep mode, the system can disable the PLL. The system can then disable the SYSCLK source for greater system power savings. Power-on reset procedures for restarting and relocking the PLL must be followed upon exiting deep sleep.

Figure 10-1 shows the four power management stages and the seven state transitions (T1–T7).

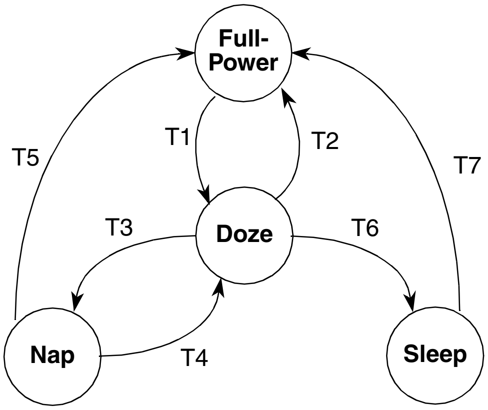


Figure 10-1. Power Management State Diagram

The states are described as follows:

- Full-power—Normal operation. All clocks are on, instructions are fetched and executed.
- Doze—All clocks are on, instruction dispatch is halted. Snoops are serviced by all caches. $\overline{\text{QREQ}}$ is asserted. Note that this is not a user-definable state; it is an intermediate state between full-power and either nap or sleep.
- Nap—The timebase and COP clocks run, all other clocks are off. Instruction dispatch is halted and $\overline{\text{QREQ}}$ remains asserted.
- Sleep—All clocks are off except COP/JTAG clocks. Instruction dispatch is halted. $\overline{\text{QREQ}}$ remains asserted.

The transitions are described in Table 10-1.

Table 10-1. Power Management State Transitions

Transition	Cause
T1	HID0[NAP] or HID0[SLEEP] and MSR[POW] set and core is idle.
T2	External interrupt, $\overline{\text{SMI}}$ interrupt, $\overline{\text{SRESET}}$, $\overline{\text{HRESET}}$, Machine check, Decrementer interrupt
T3	HID0[NAP] and MSR[POW] are set, the system asserts $\overline{\text{QACK}}$, bus and SRAM are idle.
T4	The system negates $\overline{\text{QACK}}$, signaling a pending snoop operation.
T5	External interrupt, $\overline{\text{SMI}}$ interrupt, $\overline{\text{SRESET}}$, $\overline{\text{HRESET}}$, Machine check, Decrementer interrupt
T6	HID0[SLEEP] and MSR[POW] are set, the system asserts $\overline{\text{QACK}}$, the bus and SRAM are idle.
T7	External interrupt, $\overline{\text{SMI}}$ interrupt, $\overline{\text{SRESET}}$, $\overline{\text{HRESET}}$, or Machine check

The definition of the five external interrupts are explained in Table 4.2 in Chapter 4.

The MPC7450 has two software-controllable power-saving modes, nap and sleep, which progressively reduce power dissipation. Nap mode also implements a doze state in which the processor can snoop transactions on the MPX/60x bus and then return to nap.

Software controls the HID0[NAP,SLEEP] and MSR[POW] bits and can move the MPC7450 into a power savings mode at any time. To wake from a power-saving mode by using an external interrupt, the user should take care that MSR[EE] is set.

10.2.1 Full-Power Mode

The full-power mode is the default power state. As the MPC7450 is fully powered, the internal functional units are operating at the full processor clock speed. If dynamic power management mode (DPM) is enabled, any idle functional units automatically enter a low-power state without affecting performance, software execution, or external hardware.

10.2.2 Nap Mode

Nap mode disables clocking in the MPC7450 core but maintains the lock on the PLL. The time base/decrementer and the COP/JTAG logic remain operational.

Nap mode reduces power consumption by halting instruction fetch and dispatch, disabling bus snooping. Only the decrementer/time base registers, the PLL, and JTAG logic remain in a powered state. The MPC7450 returns to the full-power state after receiving an external asynchronous interrupt, a system management interrupt, a hard or soft reset, or a machine check input ($\overline{\text{MCP}}$), or decrementer exception. Return to full-power state from a nap state takes only a few processor clock cycles. Because the processor is in nap mode, the system causes the processor to enter doze state by negating $\overline{\text{QACK}}$. The processor snoops bus operations and performs appropriate memory coherency operations while in the doze mode. When the system reasserts $\overline{\text{QACK}}$, the processor returns to the nap mode after completing any memory operations.

10.2.2.1 Entering Nap Mode

In full-power state, software can set HID0[NAP] and MSR[POW]. Because the core is idle, the MPC7450 transitions to doze state and asserts $\overline{\text{QREQ}}$. The system should then assert $\overline{\text{QACK}}$ if no snoop operations are pending. Because all bus activity is halted, the MPC7450 enters nap mode.

10.2.2.2 Exiting Nap Mode

Any External interrupt, SMI interrupt, SRESET, HRESET, or Machine check wakes MPC7450 from nap and puts it into full-power state within a few processor cycles. Additionally the Decrementer interrupt wakes the machine from nap.

10.2.2.3 Snooping in Nap Mode (Doze)

If the MPC7450 is in nap mode, it enters the doze state to service a snoop while the system negates $\overline{\text{QACK}}$. The MPC7450 requires 4 bus cycles to transit from nap to doze before the snoop operation appears on the bus. Once the snoop operation has completed its tenure on the bus the system may reassert $\overline{\text{QACK}}$ if no other snoop operations are pending. The MPC7450 continues to service the snoop if necessary, as in the case of a data cache push as a result of the snoop, and then return to nap when it is complete.

10.2.3 Sleep Mode

The sleep mode disables all clocking in the MPC7450. The PLL remains locked for fast wake response. The JTAG logic remains operational. Note that the MPC7450 does not enter doze state to snoop if it is in sleep mode. Caches must be flushed to guarantee cache coherency if other system components access memory while the MPC7450 is sleeping.

Sleep mode minimizes power consumption by disabling all internal functional units. The PLL remains locked and running but no clocks propagate to functional units. Note that the time base is not operational in sleep mode. It must be reset by an external source upon exiting sleep mode. The MPC7450 returns to the full-power state upon receipt of an external asynchronous interrupt, a system management interrupt, a hard or soft reset, or a machine check input ($\overline{\text{MCP}}$). Return to full-power state from a sleep mode takes only a few processor clock cycles.

10.2.3.1 Entering Sleep Mode

In the full-power state, software can set HID0[SLEEP] and MSR[POW]. Because the core is idle, the MPC7450 enters doze state and asserts $\overline{\text{QREQ}}$. The system should then assert $\overline{\text{QACK}}$ if no snoop operations are pending. The MPC7450 enters sleep mode as long as there is no bus activity.

10.2.3.2 Exiting Sleep Mode

Any external interrupt (External interrupt, SMI interrupt, SRESET, HRESET, and Machine check) wakes the MPC7450 from sleep mode and places it into full-power state within a few processor cycles. The Decrementer interrupt does not wake the machine from sleep. Because the time base has been inactive during the sleep mode it is necessary to update from an external reference upon exiting sleep.

10.2.3.3 Deep Sleep Mode

In sleep mode, the MPC7450 can achieve further power savings by disabling the PLL. To put the MPC7450 into deep sleep from the sleep mode, the system can configure the PLL to the off state (PLL_CFG[0:4] = 0x1E, PLL_CFG[0:5] = 0x3C in the MPC7448). See the *MPC7447A RISC Microprocessor Hardware Specifications*. Therefore, the system can achieve further power savings by disabling the SYSCLK source. Waking the MPC7450 from deep sleep can be achieved only with $\overline{\text{HRESET}}$. SYSCLK should be resumed before asserting $\overline{\text{HRESET}}$ and reconfiguring PLL_CFG to an active state. Refer to the power-on reset instructions for time required for PLL to achieve lock.

Power-on reset procedures for restarting and relocking the PLL must be followed upon exiting this deep sleep state. Returning the MPC7450 to 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.

10.2.4 Power Management Software Considerations

Because the MPC7450 is a three-issue processor with out of order execution capability, care must be taken in how the power management mode is entered. Furthermore, nap and sleep modes require all outstanding bus operations to be completed before these modes are entered. Normally, during system configuration time, one of these power management modes would be selected by setting the appropriate HID0 bit. Later on, the mode is invoked by setting MSR[POW]. To ensure a clean transition into and out of a power-management mode, the **mtmsr** which sets the POW bit must be preceded by the proper cache flushes, instruction cache disable followed by **isync**, TLB invalidates, and then **dssall** and **sync** instructions. The sleep mode entry sequence is as follows:

```
         mtHID0 (NAP | SLEEP)
         ...
         ...
         dssall
         ... cache flushes... (1)
         ... instruction cache disable ... (2)
         isync (3)
         ... TLB invalidates ... (4)
loop     sync
         mtmsr[POW = 1]
         isync
         b loop
```

The **dssall** instruction is needed to kill any outstanding stream touch instructions not killed by a **sync**.

10.2.5 Dynamic Frequency Switching (DFS)

The new dynamic frequency switching (DFS) feature adds the ability to divide the processor-to-system bus ratio by two or four during normal functional operation by setting the HID1[DFS2, DFS4] bits. The frequency change occurs in one clock cycle and no idle waiting period is required to switch between modes. This feature may or may not be used in combination with voltage derating by a system to reduce the power consumption of the processor.

For example, an MPC7447A microprocessor operating at 1-GHz internal frequency, configured to an 8:1 processor-to-system ratio at power-on reset, can dynamically change the processor-to-system ratio to 4:1 by setting the DFS2 bit. The processor frequency would be reduced to 500 MHz without ever cycling the processor through hard reset. The applied system clock frequency also does not need to change for this power reduction step. See the *MPC7447A RISC Microprocessor Hardware Specifications* for acceptable frequency/voltage combinations.

NOTE

DFS is not available for all processor-to-system ratios configurable during hard reset and the MPC7447A and MPC7448 have some differences in DFS implementations. See Section 10.2.5.2, “DFS in the MPC7447A” and Section 10.2.5.3, “DFS in the MPC7448” for more information about the differences in DFS implementation. Additional details are also documented in the hardware specifications for the individual device.

10.2.5.1 Snooping Restrictions

During the processor cycles in which dynamic frequency switching occurs, the address tenure of an external snoop must not be in progress on the external bus if the selected DFS ratio is less than 5:1.

If a snoop address tenure is in progress during the transition of the HID1[DFS2, DFS4] bits, then the processor may:

- Respond with either the incorrect snoop response for the snooped transaction,
- Respond with the incorrect snoop response for the next transaction, or
- Cause contention on the $\overline{\text{ARTRY}}$ or $\overline{\text{SHD}}$ pins.

The processor requires a minimum of 5 core cycles to process a snoop and generate a response after latching $\overline{\text{TS}}$ and associated transfer attributes. As a result, if the processor core frequency is less than five times the system bus frequency, the system must extend the address tenure of all transactions that are snooped by the processor by delaying assertion of $\overline{\text{AACK}}$. For core to bus frequency ratios of 2:1 and 2.5:1, $\overline{\text{AACK}}$ must be delayed a minimum of two bus cycles; for core:bus frequency ratios of 3:1, 3.5:1, 4:1, and 4.5:1, $\overline{\text{AACK}}$ must be delayed a minimum of one bus cycle. Table 10-2 summarizes the required system $\overline{\text{AACK}}$ delay for ratios less than 5:1.

Table 10-2. Required System $\overline{\text{AACK}}$ Delay for Ratios < 5:1

Dynamic PLL Ratio	Required $\overline{\text{AACK}}$ Delay
2:1	2 cycles
2.5:1	
3:1	1 cycle
3.5:1	
4:1	
4.5:1	

Since the processor requires five internal clock cycles to provide the correct response to a snoop on the external bus, delaying address acknowledge assures that the correct snoop response will be asserted by the processor.

The proper sequencing for changing $\overline{\text{AACK}}$ delay is to:

1. Modify the system to delay $\overline{\text{AACK}}$,
2. Alter the HID1[DFS2, DFS4] bits to choose the ratio < 5:1,
3. Operate at the lower frequency,
4. Clear the DFS bits, and
5. Modify the system to remove $\overline{\text{AACK}}$ delay.

10.2.5.2 DFS in the MPC7447A

The MPC7447A supports divide-by-two (DFS2) mode via software control of the HID1[DFS2] bit; divide-by-four (DFS4) mode is not supported on the MPC7447A. Since the MPC7447A does not support quarter clock ratios or the 1:1 ratio, the DFS feature is limited to halving the integer PLL ratios to 2:1 and higher. See the *MPC7447A RISC Microprocessor Hardware Specifications* for specific information regarding PLL ratios and DFS.

DFS can be used in conjunction with the Nap and Sleep modes and it is possible to enter and exit these modes while DFS is enabled. Because the only way to wake from Deep Sleep mode is by assertion of $\overline{\text{HRESET}}$ and DFS is disabled (HID1[DFS2] cleared) by default, it is not possible to wake from Deep Sleep mode and maintain DFS enabled.

10.2.5.3 DFS in the MPC7448

The MPC7448 adds support for additional divide ratios in DFS (divide by two and divide by four). Similar to the 7447A, the minimum PLL ratio implemented by DFS is 2:1. See the *MPC7448 RISC Microprocessor Hardware Specifications* for specific information regarding PLL ratios and DFS.

As with the MPC7447A, DFS can be used in conjuction with the Nap and Sleep modes and it is possible to enter and exit these modes while DFS is enabled. In addition, the MPC7448 allows hardware control of DFS via the $\overline{\text{DFS2}}$ and $\overline{\text{DSF4}}$ signals, making it possible to exit nap or sleep mode and return to full-power mode with DFS enabled. These signals can be used in conjunction with or in lieu of software control, and can also be used to disable DFS mode completely. See Section 8.4.4.6, “DFS Divide-by-Two and Divide-by-Four (DFS2 and DFS4) (MPC7448-Specific) for more information about the DFS2 and DFS4 signals. See also the *MPC7448 RISC Microprocessor Hardware Specifications* for additional details on DFS2 and DFS4 mode.

10.3 Instruction Cache Throttling

The MPC7450 provides an instruction cache throttling mechanism to effectively reduce the instruction execution rate without the complexity and overhead of dynamic clock control. When used with dynamic power management, instruction cache throttling provides the system designer with a flexible way to control device temperature while allowing the processor to continue operating.

The instruction cache throttling mechanism simply reduces the instruction dispatch rate. Normally, as many as three instructions are dispatched each cycle. For thermal management, the MPC7450 provides a supervisor-level instruction cache throttling control register (ICTC). The instruction dispatch rate is reduced by writing a nonzero value into ICTC[INTERVAL] and enabling instruction cache throttling by setting ICTC[E]. When this bit is clear, MPC7450 dispatches instructions normally. The overall junction temperature reduction results from dynamic power management of execution units when the MPC7450 is idle between the instruction dispatches; thus, instruction cache throttling does not provide thermal reduction unless HID0[DPM] = 1. Note that during instruction cache throttling, the PLL configuration remains unchanged. System software can control instruction forwarding by writing a nonzero value to the ICTC register, a supervisor-level register shown in Figure 10-2. Note also when instruction cache throttling is enabled, to reduce overall junction temperature, the performance does degrade.

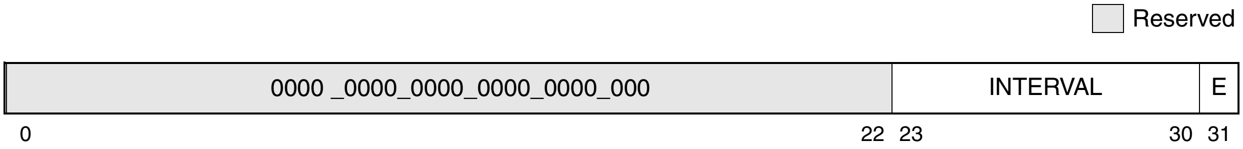


Figure 10-2. Instruction Cache Throttling Control Register (ICTC)

Table 10-3 describes ICTC fields.

Table 10-3. ICTC Field Descriptions

Bits	Name	Description
0–22	—	Reserved, should be cleared.
23–30	INTERVAL	Instruction forwarding interval expressed in processor clocks. When throttling is enabled, the interval field specifies the minimum number of cycles between instructions being dispatched. (The MPC7450 dispatches one instruction every INTERVAL cycle.) The minimum interval for throttling control is 2 cycles. 0x00, 0x01, 0x02 One instruction dispatches every 2 processor clocks 0x03 One instruction dispatches every 3 processor clocks ... 0xFF One instruction dispatches every 255 processor clocks
31	E	Enable instruction throttling. 0 Instructions dispatch normally. 1 Only one instruction dispatches every INTERVAL cycles.

10.4 MPC7447A and MPC7448 Temperature Diode

The processor has a temperature diode that can be used in conjunction with other system temperature monitoring devices. These devices use the negative temperature coefficient of a diode operated at a constant current to determine the temperature of the microprocessor and its environment. For specifications of the processor’s on-board temperature diode see the appropriate hardware specification.

Chapter 11
Performance Monitor

The PowerPC architecture defines an optional performance monitor facility that provides the ability to monitor and count predefined events such as processor clocks, misses in the instruction cache, data cache, or L2 cache, types of instructions dispatched, mispredicted branches, and other occurrences. The count of such events (that may be an approximation) can be used to trigger the performance monitor exception. Note that some earlier processors implemented the performance monitor facility before it was defined by the PowerPC architecture.

The performance monitor can be used for the following:

- To increase system performance with efficient software, especially in a multiprocessing system—Memory hierarchy behavior can be monitored and studied in order to develop algorithms that schedule tasks (and perhaps partition them) and that structure and distribute data optimally.
- To characterize processors—Some environments may not be easily characterized by a benchmark or trace.
- To help system developers bring up and debug their systems

AltiVec Technology and the Performance Monitor

The AltiVec technology features do not affect the basic implementation of the performance monitor. However, the performance monitor provides the ability to count the following AltiVec operations:

- Instructions executed by the individual AltiVec execution units
- Completed AltiVec load instructions
- Cycles during which the VIU1, VIU2, VFPU, and VPU had a valid dispatch but invalid operands
- VFPU traps that can be generated in Java mode
- Completed **mtvscr** and **mtvrsave** instructions
- Times that the VSCR saturate bit is set
- Completed of **dss** and **dssall** instructions
- **dst***x* instruction event counts—dispatches, misses, refreshes, suspensions, premature cancellations, and resumptions
- Cycle counts when the VALU has a valid **mfvscr** dispatch but cannot execute it because it is not at the bottom of the completion queue (CQ)

11.1 Overview

The performance monitor uses the following resources defined by the PowerPC architecture:

- The performance monitor mark bit in the MSR (MSR[PMM]). This bit identifies programs to be monitored.
- The privilege level bit in the MSR (MSR[PR]). This bit identifies the mode the processor is in (supervisor or user).
- Special-purpose registers (SPRs):
  - The performance monitor counter registers (PMC1–PMC6) are 32-bit counters used to count the times a software-selectable event has occurred. PMC5 and PMC6 are used to count events generated by the memory subsystem. UPMC1–UPMC6 provide user-level read access to these registers.
  - The monitor mode control registers (MMCR0–MMCR2). Control fields in the MMCR*n* registers select events to be counted, determine whether performance monitor exceptions are caused by a time-base register transition, maintain counter overflow, and specify the conditions under which counting is enabled. UMMCR0–UMMCR2 provide user-level read access to these registers.
  - The sampled instruction address register (SIAR) contains the effective address of the last instruction that completes before the performance monitor exception is generated. USIAR provides user-level read access to the SIAR.
  - Note that in previous processors the optional SDAR and USDAR registers could be written to by boot code without causing an exception, this is not the case in the MPC7450. An **mtspr** or **mfspr** SDAR or USDAR instruction causes a program exception.
- The performance monitor exception follows the normal PowerPC exception model and has a defined exception vector offset (0x00F00). Its priority is below the trace exception and above the AltiVec unavailable exception.

11.2 Performance Monitor Exception

The performance monitor provides the ability to generate a performance monitor exception triggered by an enabled condition or event. This exception is triggered by an enabled condition or event defined as follows:

- A PMC*n* register overflow condition occurs:
  - MMCR0[PMC1CE] and PMC1[OV] are both set
  - MMCR0[PMC*n*CE] and PMC*n*[OV] are both set ($n > 1$)
- A time-base event—MMCR0[TBEE] = 1 and the TBL bit specified in MMCR0[TBSEL] changes from 0 to 1

MMCR0[PMXE] must be set for any of these conditions to signal a performance monitor exception.

Although the performance monitor exception may occur with MSR[EE] = 0, the exception is not taken until MSR[EE] = 1.

As a result of a performance monitor exception being generated, the performance monitor saves in the SIAR the effective address of the last instruction completed before the exception is taken. Note that SIAR is not updated if performance monitor counting has been disabled by setting MMCR0[0].

The priority of the performance monitor exception is below the trace exception and above the AltiVec unavailable exception. See Section 4.2, “MPC7450 Exception Recognition and Priorities,” for a list of exception priorities.

Exception handling for the performance monitor exception is described in Section 4.6.13, “Performance Monitor Exception (0x00F00).”

11.2.1 Performance Monitor Signals

The $\overline{\text{PMON\_IN}}$ signal is used by the performance monitor event MMCR0[PMC1SEL] = 7 (0b000_0111) to count the number of times the $\overline{\text{PMON\_IN}}$ signal transitions from negated to asserted. Note that this event is enabled in the performance monitor control registers (MMCR0, or MMCR1, PMC*n*) and must be enabled in order for this event to be monitored.

The $\overline{\text{PMON\_OUT}}$ signal is asserted by the performance monitor when a performance monitor threshold or negative counter condition has been reached, whether or not performance monitor exceptions are enabled; that is, the setting of MMSR0[PMXE] does not affect the function of this signal.

11.2.2 Using Timebase Event to Trigger or Freeze a Counter or Generate an Exception

The use of the trigger and freeze counter conditions depends on these enabled conditions and events. When MMCR0[TBEE] (timebase enable event) is 1, a timebase transition is generated to the performance monitor if the TBL bit specified in MMCR0[TBSEL] changes from 0 to 1. Timebase transition events can be used to freeze the counters (MMCR0[FCECE]), trigger the counters (MMCR0[TRIGGER]), or generate an exception (MMCR0[PMXE]).

Changing the bits specified in MMCR0[TBSEL] while MMCR0[TBEE] is set may cause a false 0 to 1 transition that signals the specified action (freeze, trigger, or exception) to occur immediately.

11.3 Performance Monitor Registers

The following sections describe the registers used by the performance monitor. Note that reading and writing to the performance monitor SPRs do not synchronize the processor.

An explicit synchronization instruction, such as **sync**, should be placed before and after an **mfspr** or **mtspr** of one of these registers to guarantee an accurate count.

11.3.1 Performance Monitor Special-Purpose Registers

The performance monitor incorporates the SPRs listed in Table 11-1 and Table 11-2. The supervisor-level registers in Table 11-1 are accessed through the **mtspr** and **mfspr** instructions.

Table 11-1. Performance Monitor SPRs—Supervisor Level

SPR Number	spr[5–9] ǁ spr[0–4]	Register Name
944	0b11101 10000	Monitor mode control register 2—MMCR2
945	0b11101 10001	Performance monitor counter register 5—PMC5
946	0b11101 10010	Performance monitor counter register 6—PMC6
951	0b11101 10111	Breakpoint address mask register—BAMR
952	0b11101 11000	Monitor mode control register 0—MMCR0
953	0b11101 11001	Performance monitor counter register 1—PMC1
954	0b11101 11010	Performance monitor counter register 2—PMC2
955	0b11101 11011	Sampled instruction address register—SIAR
956	0b11101 11100	Monitor mode control register 1—MMCR1
957	0b11101 11101	Performance monitor counter register 3—PMC3
958	0b11101 11110	Performance monitor counter register 4—PMC4

The user-level registers in Table 11-2 are read-only and are accessed with the **mfspr** instruction. Attempting to write to one of these registers in either supervisor or user mode causes a program exception.

Table 11-2. Performance Monitor SPRs—User Level (Read-Only)

SPR Number	spr[5–9] ǁ spr[0–4]	Register Name
928	0b11101 00000	User monitor mode control register 2—UMMCR2
929	0b11101 00001	User performance monitor counter register 5—UPMC5
930	0b11101 00010	User performance monitor counter register 6—UPMC6
936	0b11101 01000	User monitor mode control register 0—UMMCR0
937	0b11101 01001	User performance monitor counter register 1—UPMC1
938	0b11101 01010	User performance monitor counter register 2—UPMC2
939	0b11101 01011	User sampled instruction address register—USIAR
940	0b11101 01100	User monitor mode control register 1—UMMCR1
941	0b11101 01101	User performance monitor counter register 3—UPMC3
942	0b11101 01110	User performance monitor counter register 4—UPMC4

11.3.2 Monitor Mode Control Register 0 (MMCR0)

The monitor mode control register 0 (MMCR0), shown in Figure 11-1 is a 32-bit SPR provided to specify events to be counted and recorded. If the state of MSR[PR] and MSR[PMM] matches a state specified in MMCR0, then counting is enabled; see Section 11.4, “Event Counting,” for further details. The MMCR0 can be accessed only in supervisor mode. User-level software can read the contents of MMCR0 by issuing an **mfspr** instruction to UMMCR0, described in Section 11.3.2.1, “User Monitor Mode Control Register 0 (UMMCR0).”

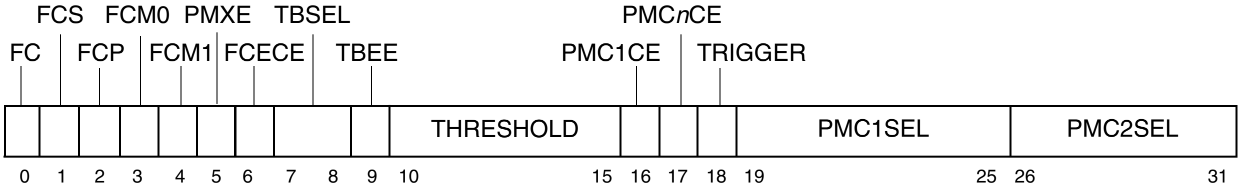


Figure 11-1. Monitor Mode Control Register 0 (MMCR0)

This register is automatically cleared at power-up. Reading this register does not change its contents. Table 11-3 describes the fields of MMCR0.

Table 11-3. MMCR0 Field Descriptions

Bits	Name	Description
0	FC	Freeze counters 0 The PMCs are incremented (if permitted by other MMCR bits). 1 The PMCs are not incremented (performance monitor counting is disabled). The processor automatically sets this bit when an enabled condition or event occurs and MMCR0[FCECE] = 1. Note that SIAR is not updated if performance monitor counting is disabled.
1	FCS	Freeze counters in supervisor mode 0 The PMCs are incremented (if permitted by other MMCR bits). 1 The PMCs are not incremented if MSR[PR] = 0.
2	FCP	Freeze counters in user mode 0 The PMCs are incremented (if permitted by other MMCR bits). 1 The PMCs are not incremented if MSR[PR] = 1.
3	FCM1	Freeze counters while mark = 1 0 The PMCs are incremented (if permitted by other MMCR bits). 1 The PMCs are not incremented if MSR[PMM] = 1.
4	FCM0	Freeze counters while mark = 0 0 The PMCs are incremented (if permitted by other MMCR bits). 1 The PMCs are not incremented if MSR[PMM] = 0.
5	PMXE	Performance monitor exception enable 0 Performance monitor exceptions are disabled. 1 Performance monitor exceptions are enabled until a performance monitor exception occurs; at that time, MMCR0[PMXE] is automatically cleared. Software can clear PMXE to prevent performance monitor exceptions. Software can also set PMXE and then poll it to determine whether an enabled condition or event occurred.

Table 11-3. MMCR0 Field Descriptions (continued)

Bits	Name	Description
6	FCECE	Freeze counters on enabled condition or event 0 The PMCs are incremented (if permitted by other MMCR bits). 1 The PMCs are incremented (if permitted by other MMCR bits) until an enabled condition or event occurs when MMCR0[TRIGGER] = 0, at that time MMCR0[FC] is set. If the enabled condition or event occurs when MMCR0[TRIGGER] = 1, FCECE is treated as if it were 0. The use of the trigger and freeze counter conditions depends on the enabled conditions and events described in Section 11.2, “Performance Monitor Exception.”
7–8	TBSEL	Time-base selector. Selects the time-base bit that can cause a time-base transition event (the event occurs when the selected bit changes from 0 to 1). 00 TBL[31] 01 TBL[23] 10 TBL[19] 11 TBL[15] Time-base transition events can be used to periodically collect information about processor activity. In multiprocessor systems in which the TB registers are synchronized among processors, time-base transition events can be used to correlate the performance monitor data obtained by the several processors. For this use, software must specify the same TBSEL value for all the processors in the system. Because the time-base frequency is implementation-dependent, software should invoke a system service program to obtain the frequency before choosing a value for TBSEL.
9	TBEE	Time-base event enable 0 Time-base transition events are disabled. 1 Time-base transition events are enabled. A time-base transition is generated to the performance monitor if the TB bit specified in MMCR0[TBSEL] changes from 0 to 1. Time-base transition events can be used to freeze the counters (MMCR0[FCECE]), trigger the counters (MMCR0[TRIGGER]), or signal an exception (MMCR0[PMXE]). Changing the bits specified in MMCR0[TBSEL] while MMCR0[TBEE] is enabled may cause a false 0 to 1 transition that signals the specified action (freeze, trigger, or exception) to occur immediately.
10–15	THRESHOLD	Threshold. Contains a threshold value between 0 to 63. Two types of thresholds can be counted. • The first type counts any event that lasts longer than the threshold value and uses MMCR2[THRESHMULT] to scale the threshold value by 2 or 32. • The second type counts only the events that exceed the threshold value. This type does not use MMCR2[THRESHMULT] to scale the threshold value (MMCR2[THRESHMULT] = 0). By varying the threshold value, software can obtain a profile of the characteristics of the events subject to the threshold. For example, if PMC1 counts cache misses that the duration exceeds the threshold value, software can obtain the distribution of cache miss durations for a given program by monitoring the program repeatedly using a different threshold value each time.
16	PMC1CE	PMC1 condition enable. Controls whether counter negative conditions due to a negative value in PMC1 are enabled. 0 Counter negative conditions for PMC1 are disabled. 1 Counter negative conditions for PMC1 are enabled. These events can be used to freeze the counters (MMCR0[FCECE]), trigger the counters (MMCR0[TRIGGER]), or signal an exception (MMCR0[PMXE]).

Table 11-3. MMCR0 Field Descriptions (continued)

Bits	Name	Description
17	PMC*n*CE	PMC*n* condition enable. Controls whether the counter negative conditions due to a negative value in any PMC*n* (that is, in any PMC except PMC1) are enabled. 0 Counter negative conditions for all PMC*n*s are disabled. 1 Counter negative conditions for all PMC*n*s are enabled. These events can be used to freeze the counters (MMCR0[FCECE]), trigger the counters (MMCR0[TRIGGER]), or signal an exception (MMCR0[PMXE]).
18	TRIGGER	Trigger 0 The PMCs are incremented (if permitted by other MMCR bits). 1 PMC1 is incremented (if permitted by other MMCR bits). The PMC*n*s are not incremented until PMC1 is negative or an enabled condition or event occurs, at that time the PMC*n*s resume incrementing (if permitted by other MMCR bits) and MMCR0[TRIGGER] is cleared. The description of FCECE explains the interaction between TRIGGER and FCECE. Uses of TRIGGER include the following: • Resume counting in the PMC*n*s when PMC1 becomes negative without causing a performance monitor exception. Then freeze all PMCs (and optionally cause a performance monitor exception) when a PMC*n* becomes negative. The PMC*n*s then reflect the events that occurred after PMC1 became negative and before PMC*n* becomes negative. This use requires the following MMCR0 bit settings: –TRIGGER = 1 –PMC1CE = 0 –PMC*n*CE = 1 –TBEE = 0 –FCECE = 1 –PMXE = 1 (if a performance monitor exception is desired) • Resume counting in the PMC*n*s when PMC1 becomes negative and cause a performance monitor exception without freezing any PMCs. The PMC*n*s then reflect the events that occurred between the time PMC1 became negative and the time the exception handler reads them. This use requires the following MMCR0 bit settings: –TRIGGER = 1 –PMC1CE = 1 –TBEE = 0 –FCECE = 0 –PMXE = 1 The use of the trigger and freeze counter conditions depends on the enabled conditions and events described in Section 11.2, “Performance Monitor Exception.”
19–25	PMC1SEL	PMC1 selector. Contains a code (one of at most 128 values) that identifies the event to be counted in PMC1. See Table 11-9.
26–31	PMC2SEL	PMC2 selector. Contains a code (one of at most 64 values) that identifies the event to be counted in PMC2. See Table 11-10.

MMCR0 can be accessed with the **mtspr** and **mfspr** instructions using SPR 952.

11.3.2.1 User Monitor Mode Control Register 0 (UMMCR0)

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

11.3.3 Monitor Mode Control Register 1 (MMCR1)

The monitor mode control register 1 (MMCR1) functions as an event selector for performance monitor counter registers 3, 4, 5, and 6 (PMC3, PMC4, PMC5, PMC6). The MMCR1 register is shown in Figure 11-2.

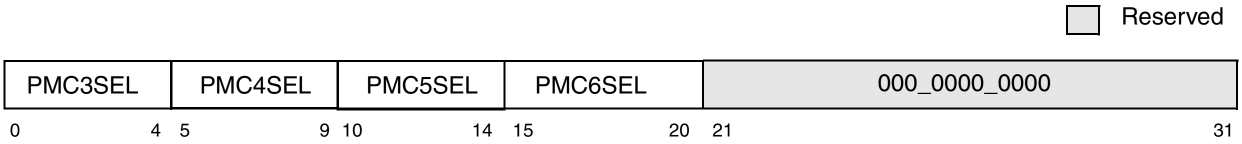


Figure 11-2. Monitor Mode Control Register 1 (MMCR1)

Bit settings for MMCR1 are shown in Table 11-4. The corresponding events are described in Section 11.3.6, “Performance Monitor Counter Registers (PMC1–PMC6).”

Table 11-4. MMCR1 Field Descriptions

Bits	Name	Description
0–4	PMC3SEL	PMC3 selector. Contains a code (one of at most 32 values) that identifies the event to be counted in PMC3. See Table 11-11.
5–9	PMC4SEL	PMC4 selector. Contains a code (one of at most 32 values) that identifies the event to be counted in PMC4. See Table 11-12.
10–14	PMC5SEL	PMC5 selector. Contains a code (one of at most 32 values) that identifies the event to be counted in PMC5. See Table 11-13.
15–20	PMC6SEL	PMC6 selector. Contains a code (one of at most 32 values) that identifies the event to be counted in PMC6. See Table 11-14.
21–31	—	Reserved

MMCR1 can be accessed with the **mtspr** and **mfspr** instructions using SPR 956. User-level software can read the contents of MMCR1 by issuing an **mfspr** instruction to UMMCR1, described in Section 11.3.4.1, “User Monitor Mode Control Register 2 (UMMCR2).”

11.3.3.1 User Monitor Mode Control Register 1 (UMMCR1)

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

11.3.4 Monitor Mode Control Register 2 (MMCR2)

The monitor mode control register 2 (MMCR2) contains only one bit. This bit is used to scale the value in the MMCR0[THRESHOLD] field for certain threshold events. If MMCR2[THRESMULT] = 0, it scales the value by 2 times. If MMCR2[THRESMULT] = 1, it scales the value by 32 times.

The MMCR2 register is shown in Figure 11-3.

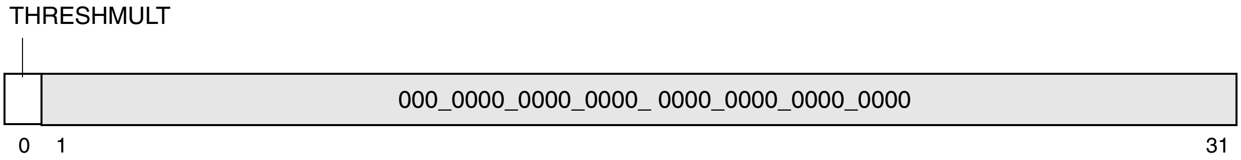


Figure 11-3. Monitor Mode Control Register 2 (MMCR2)

Table 11-5 describes MMCR2 fields.

Table 11-5. MMCR2 Field Descriptions

Bits	Name	Description
0	THRESHMULT	Threshold multiplier. Used to extend the range of the THRESHOLD field, MMCR0[10–15]. 0 Threshold field is multiplied by 2. 1 Threshold field is multiplied by 32.
1–31	—	Reserved

MMCR2 can be accessed with **mtspr** and **mfspr** using SPR 944. User-level software can read the contents of MMCR2 by issuing an **mfspr** instruction to UMMCR2, described in Section 11.3.4.1, “User Monitor Mode Control Register 2 (UMMCR2).”

11.3.4.1 User Monitor Mode Control Register 2 (UMMCR2)

The contents of MMCR2 are reflected to UMMCR2, which can be read by user-level software. UMMCR2 can be accessed with the **mfspr** instructions using SPR 928.

11.3.5 Breakpoint Address Mask Register (BAMR)

The breakpoint address mask register (BAMR), shown in Figure 11-4, is used in conjunction with the events that monitor IABR hits.

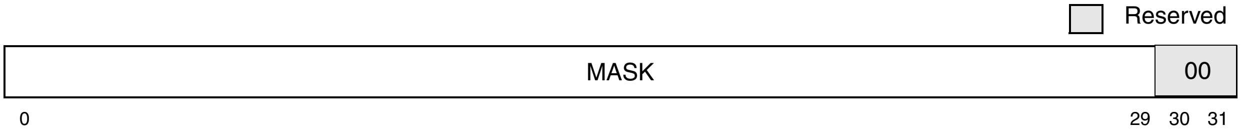


Figure 11-4. Breakpoint Address Mask Register (BAMR)

Table 11-6 describes BAMR fields.

Table 11-6. BAMR Field Descriptions

Bit	Name	Description
0–29	MASK	Used with PMC1 event (PMC1 event 42) that monitor IABR hits The addresses to be compared for an IABR match are affected by the value in BAMR: • IABR hit (PMC1, event 42) occurs if IABR_CMP (that is, IABR AND BAMR) = instruction_address_compare (that is, EA AND BAMR) • IABR_CMP[0–29] = IABR[0–29] AND BAMR[0–29] instruction_addr_cmp[0–29] = instruction_addr[0–29] AND BAMR[0–29] Be aware that breakpoint event 42 of PMC1 can be used to trigger ISI exceptions when the performance monitor detects an enabled overflow. This feature supports debug purposes and occurs only when IABR[30] is set. To avoid taking one of the above exceptions, IABR[30] should be cleared.
30–31	—	Reserved

BAMR can be accessed with **mtspr** and **mfspr** using SPR 951. For synchronization requirements on the BAMR register see Table 2-46.

11.3.6 Performance Monitor Counter Registers (PMC1–PMC6)

PMC1–PMC6, shown in Figure 11-5, are 32-bit counters that can be programmed to generate a performance monitor exception when they overflow.

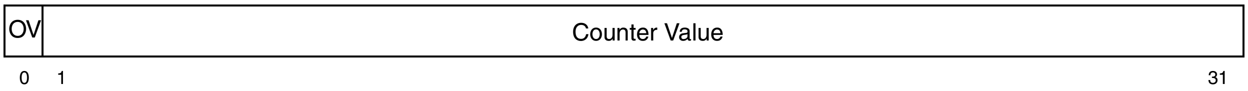


Figure 11-5. Performance Monitor Counter Registers (PMC1–PMC6)

The bits contained in the PMC registers are described in Table 11-7.

Table 11-7. PMC*n* Field Descriptions

Bits	Name	Description
0	OV	Overflow. When this bit is set, it indicates that this counter has overflowed and reached its maximum value so that PMCn[OV] = 1.
1–31	Counter value	Counter value. Indicates the number of occurrences of the specified event.

Counters overflow when the high-order (sign) bit becomes set; that is, they reach the value 2,147,483,648 (0x8000_0000). However, an exception is not generated unless both MMCR0[PMXE] and either MMCR0[PMC1CE] or MMCR0[PMC*c*CE] are also set as appropriate.

Note that the exception can be masked by clearing MSR[EE]; the performance monitor condition may occur with MSR[EE] cleared, but the exception is not taken until MSR[EE] is set. Setting MMCR0[FCECE] forces counters to stop counting when a counter exception or any enabled condition or event occurs. Setting MMCR0[TRIGGER] forces counters PMC*n* (*n* > 1), to begin counting when PMC1 goes negative or an enabled condition or event occurs.

Software is expected to use the **mtspr** instruction to explicitly set PMC to non-overflowed values. Setting an overflowed value may cause an erroneous exception. For example, if both MMCR0[PMXE] and either MMCR0[PMC1CE] or MMCR0[PMC*n*CE] are set and the **mtspr** instruction loads an overflow value, an exception may be taken without an event-counting having taken place.

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

- PMC1 is SPR 953
- PMC2 is SPR 954
- PMC3 is SPR 957
- PMC4 is SPR 958
- PMC5 is SPR 945
- PMC6 is SPR 946

11.3.6.1 User Performance Monitor Counter Registers (UPMC1–UPMC6)

The contents of the PMC1–PMC6 are reflected to UPMC1–UPMC6, which can be read by user-level software. The UPMC registers can be read with the **mfspr** instructions using the following SPR numbers:

- UPMC1 is SPR 937
- UPMC2 is SPR 938
- UPMC3 is SPR 941
- UPMC4 is SPR 942
- UPMC5 is SPR 929
- UPMC6 is SPR 930

11.3.7 Sampled Instruction Address Register (SIAR)

The sampled instruction address register (SIAR) is a supervisor-level register that contains the effective address of the last instruction to complete before the performance monitor exception is generated. The SIAR is shown in Figure 11-6.

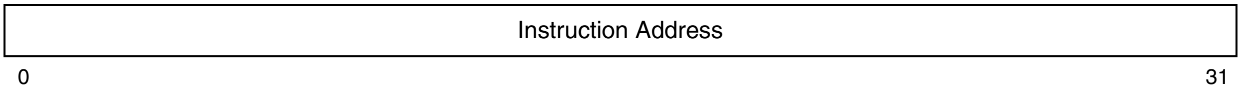


Figure 11-6. Sampled Instruction Address Register (SIAR)

Note that SIAR is not updated:

- if performance monitor counting has been disabled by setting MMCR0[FC] or
- if the performance monitor exception has been disabled by clearing MMCR0[PMXE].

SIAR can be accessed with the **mtspr** and **mfspr** instructions using SPR 955.

11.3.7.1 User Sampled Instruction Address Register (USIAR)

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

11.4 Event Counting

Counting can be enabled if conditions in the processor state match a software-specified condition. Because a software task scheduler may switch a processor's execution among multiple processes and because statistics only on a particular process may be of interest, a facility is provided to mark a process. The performance monitor bit, MSR[PMM], is used for this purpose. System software may set this bit when a marked process is running. This enables statistics to be gathered only during the execution of the marked process. The states of MSR[PR] and MSR[PMM] together define a state that the processor (supervisor or user) and the process (marked or unmarked) may be in at any time. If this state matches a state specified in the MMCR (the state in which monitoring is enabled), counting is enabled. Table 11-8 lists the states that can be monitored.

Table 11-8. Monitorable States

MSR[PR]	MSR[PMM]	Process Marked	Process UnMarked	User Mode	Supervisor Mode
0	—	—	—	√	√
1	—	—	—	√	—
—	1	√	—	—	—
—	0	—	√	—	—
0	0	—	√	√	√
0	1	√	—	√	√

Table 11-8. Monitorable States (continued)

MSR[PR]	MSR[PMM]	Process Marked	Process UnMarked	User Mode	Supervisor Mode
1	0	—	√	√	—
1	1	√	—	√	—

In addition, one of two unconditional counting modes may be specified:

- Counting is unconditionally enabled regardless of the states of MSR[PMM] and MSR[PR]. This can be accomplished by clearing MMCR0[0–4].
- Counting is unconditionally disabled regardless of the states of MSR[PMM] and MSR[PR]. This is done by setting MMCR0[0] = 1. Note that SIAR is not updated if MMCR0[0] = 1.

11.5 Event Selection

Event selection is handled through the MMCR*n* registers, described in subsequent tables.

- The six event-select fields in MMCR0 and MMCR1 are as follows:
    - MMCR0[PMC1SEL]—PMC1 input selector, 128 events selectable. See Table 11-9.
    - MMCR0[PMC2SEL]—PMC2 input selector, 64 events selectable. See Table 11-10.
    - MMCR1[PMC3SEL]—PMC3 input selector, 32 events selectable. See Table 11-11.
    - MMCR1[PMC4SEL]—PMC4 input selector, 32 events selectable. See Table 11-12.
    - MMCR1[PMC5SEL]—PMC5 input selector, 32 events selectable. See Table 11-13.
    - MMCR1[PMC6SEL]—PMC6 input selector, 64 events selectable. See Table 11-14.
- In Table 11-9 through Table 11-14, a correlation is established between each counter, events to be traced, and the pattern required for the desired selection.
- As shown in Table 11-9 through Table 11-12, the first five events are common to all six counters and are considered to be reference events. These are as follows:
    - 00000—Register holds current value
    - 00001—Number of processor cycles
    - 00010—Number of completed instructions, not including folded branches
    - 00011—Number of times the TBL bit transitions from 0 to 1. The TBL bit is specified through MMCR0[TBSEL]
        - 00 = uses the TBL[31] bit to count
        - 01 = uses the TBL[23] bit to count
        - 10 = uses the TBL[19] bit to count
        - 11 = uses the TBL[15] bit to count
    - 00100—Number of instructions dispatched: 0, 1, 2, or 3 per cycle

11.5.1 PMC1 Events

The event to be monitored can be chosen by setting MMCR0[PMC1SEL]. The selected events are counted beginning when MMCR0 is set until either MMCR0 is reset or a performance monitor exception is generated. Table 11-9 lists the selectable events and their encodings.

Table 11-9. PMC1 Events—MMCR0[PMC1SEL] Select Encodings

Number	Event	Description
0 (000_0000)	Nothing	Register counter holds current value
1 (000_0001)	Processor cycles	Counts every processor cycle
2 (000_0010)	Instructions completed	Counts all completed PowerPC and AltiVec instructions. Load/store multiple instructions (**lmw**, **stmw**) and load/store string instructions (**lswl**, **lswx**, **stswl**, **stswx**) are only counted once. Does not include folded branches. The counter can increment by 0, 1, 2, or 3, depending on the number of completed instructions per cycle. Branch folding must be disabled (HID0[FOLD] = 0) in order to count all the instructions.
3 (000_0011)	TBL bit transitions	Counts transitions from 0 to 1 of TBL bits specified through MMCR0[TBSEL]. 00 = uses the TBL[31] bit to count 01 = uses the TBL[23] bit to count 10 = uses the TBL[19] bit to count 11 = uses the TBL[15] bit to count
4 (000_0100)	Instructions dispatched	Counts dispatched instructions.The counter can increment by 0, 1, 2, or 3, depending on the number of dispatched instructions per cycle. Load/store multiple instructions (**lmw**, **stmw**) and load/store string instructions (**lswl**, **lswx**, **stswl**, **stswx**) are only counted once. This event includes instructions dispatched directly to the completion queue,
5 (000_0101)	Processor performance monitor exception	Counts times the processor begins to generate its performance monitor exception condition. The performance monitor exception condition is set when the processor performance monitor counter is negative and its exception signaling is enabled via MMCR0[PMC1CE] or MMCR0[PMC*n*CE]. The MPC7450 does not require MMCR0[PMXE] to be set to allow the exception to occur.
6 (000_0110)	—	Reserved
7 (000_0111)	External performance monitor signal	Counts times the external performance monitor signal ($\overline{\text{PMON\_IN}}$) transitions from negated to asserted.
8 (000_1000)	VPU instructions completed	Counts VPU instructions completed.
9 (000_1001)	VFPU instructions completed	Counts VFPU instruction completed.
10 (000_1010)	VIU1 instructions completed	Counts VIU1 instructions completed.
11 (000_1011)	VIU2 instructions completed	Counts VIU2 instructions completed.
12 (000_1100)	**mtvscr** instructions completed	Counts completed AltiVec **mtvscr** instructions.
13 (000_1101)	**mtvrsave** instructions completed	Counts completed AltiVec **mtvrsave** instructions.
14 (000_1110)	Cycles a VPU instruction	Counts the cycles an AltiVec instruction in the VPU reservation station is waiting for an operand.

Table 11-9. PMC1 Events—MMCR0[PMC1SEL] Select Encodings (continued)

Number	Event	Description
15 (000_1111)	Cycles a VFPU instruction	Counts the cycles an AltiVec instruction in the VFPU reservation station is waiting for an operand.
16 (001_0000)	Cycles a VIU1 instruction	Counts the cycles an AltiVec instruction in the VIU1 reservation station is waiting for an operand.
17 (001_0001)	Cycles an instruction in VIU2 reservation station waits for operand	Counts the cycles an AltiVec instruction in the VIU2 reservation station is waiting for an operand.
18 (001_0010)	**mfvscr** synchronization	Counts cycles when the VFPU has a valid **mfvscr** instruction issued, but the instruction cannot start execution because it is not at the bottom of the CQ.
19 (001_0011)	VSCR[SAT] set	Counts whenever VSCR[SAT] goes from 0 to 1.
20 (001_0100)	Store instructions	Counts completed store instructions. Store string and multiples count as single instructions. This count does not include the following instructions, which do not perform a load or store: **sync**, **eciwx**, **ecowx**, **eieio**, **dcbf**, **dcbi**, **dcbst**, **dcbt**, **dcbtst**, **dcbz**, **icbi**, **tlbie**, **tlbld**, **tlbli**, **tlbsync**, **dcba**, **dst**, **dstt**, **dstst**, **dststt**, **dss**, and **dssall**.
21 (001_0101)	L1 instruction cache misses	Counts L1 instruction cache misses. This does not include cache-inhibited or cache-disabled accesses.
22 (001_0110)	L1 data snoops	Counts the snoop accesses to the L1 data cache.
23 (001_0111)	Unresolved branches	Counts branches that are unresolved when processed. This includes branches in a speculative path that might later be thrown away due to another previously predicted branch that mispredicts.
24 (001_1000)	Cycles first speculation buffer active	Counts cycles that a predicted branch is active in the first speculation buffer.
25 (001_1001)	Branch unit stall	Counts cycles the branch unit cannot process new branches. This includes waiting on speculative branches that are not resolved.
26 (001_1010)	True branch target instruction hits	Counts the true branch target instruction hits for taken branches. Note that this count includes speculative branches that have been taken.
27 (001_1011)	Branch link stack predicted	Counts the branches that use link stack prediction. This count includes branches that are in speculative paths. This count may be greater than the sum of link-stack-correctly-resolved and link-stack-mispredicted because another branch may mispredict and cause this branch to be thrown off the link stack before the resolution occurs.
28 (001_1100)	Dispatches to GPR issue queue	Counts instructions dispatched to the GPR issue queue. This includes instructions of speculative paths. Instructions that are executed by the IUs or LSU are dispatched to the GPR issue queue.
29 (001_1.101)	Cycles where 3 instructions are dispatched	Counts cycles where three instructions are dispatched from the dispatch unit.
30 (001_1110)	Counts instruction queue entries over MMCR0[THRESHOLD]	Counts the cycles when the number of valid instruction queue entries is greater than or equal to the MMCR0[THRESHOLD] value. This event does not scale the MMCR0[THRESHOLD] value.
31 (001_1111)	Counts AltiVec issue queue entries over MMCR0[THRESHOLD]	Counts the cycles when the number of valid AltiVec issue queue entries is greater than or equal to the MMCR0[THRESHOLD] value. This event does not scale the MMCR0[THRESHOLD] value.

Table 11-9. PMC1 Events—MMCR0[PMC1SEL] Select Encodings (continued)

Number	Event	Description
32 (010_0000)	Cycles where no instructions completed	Counts the cycles where no instructions are completed.
33 (010_0001)	Completed IU2 instructions	Counts IU2 instructions completed.
34 (010_0010)	Completed branch instructions	Counts branches completed, but it does not include folded branches. To count all branches, branch folding must be disabled by clearing HID0[FOLD].
35 (010_0011)	**eieio** instructions completed	Counts completed **eieio** instructions.
36 (010_0100)	**mtspr** instructions completed	Counts completed **mtspr** instructions. This count does not include **mtvscr** instructions.
37 (010_0101)	**sc** instructions completed	Counts completed system call (**sc**) instructions.
38 (010_0110)	Load string and load multiple instructions completed	Counts completed load string (**lswl**, **lswx**) and load multiple (**lm**) instructions. Load strings and load multiples are only counted once regardless of how many pieces they are broken into. An **lswx** instruction of length zero is counted once if MSR[SE] is set; otherwise, it is not counted.
39 (010_0111)	ITLB hardware table search cycles	Counts cycles spent performing a hardware table search operation for the instruction TLB. A hardware table search begins when the ITLB determines that it has missed and all instructions ahead of the ITLB miss have completed. A hardware table search ends when the page table entry (PTE) or a page fault signal is returned by the table search. Note that the cycles do not include the time it takes the MPC7450 to drain before the hardware table search operation begins.
40 (010_1000)	DTLB hardware table search cycles over MMCR0[THRESHOLD] value	Counts cycles beyond MMCR0[THRESHOLD] value that are required to perform a hardware data TLB search operation for a data access. This includes table search operations that do not find a matching PTE entry in the page table. This also includes table search operations caused by **dst**, **dstt**, **dstst**, and **dststt** instructions. This event scales the MMCR0[THRESHOLD] value as specified by MMCR2[THRESHMULT].
41 (010_1001)	L1 instruction cache accesses	Counts the L1 instruction cache accesses. This does not include cache-inhibited or cache-disabled accesses.
42 (010_1010)	Instruction breakpoint matches	Counts when the address of an instruction being completed matches the instruction address breakpoint register (IABR). A match is determined by the following equation: Match = ((IABR[0–29] & BAMR[0–29]) == (completion_address[0–29] & BAMR[0–29])) The counter can increment by 0, 1, 2, or 3 depending on the number of completed instructions per cycle
43 (010_1011)	L1 data cache load miss cycles over MMCR0[THRESHOLD] value	Counts the cycles an L1 data cache load miss requires beyond the MMCR0[THRESHOLD] value to write back to the rename registers. The cycle count compared to the threshold value represents the number of cycles starting with allocation in the L1 Miss Queue. This event scales the MMCR0[THRESHOLD] value as specified by MMCR2[THRESHMULT]. The miss latency threshold is measured per dispatched operation, not per instruction. Load strings and multiples may have multiple dispatches per instruction. Misaligned loads have multiple accesses, but only one dispatch. The measurement for a misaligned load is from the first piece that misses until the entire load finishes. Note that only the oldest entry of the LMQ is counted.
44 (010_1100)	L1 data snoop hit on modified	Counts snoop accesses to the L1 data cache that hit in a modified cache line.

Table 11-9. PMC1 Events—MMCR0[PMC1SEL] Select Encodings (continued)

Number	Event	Description
45 (010_1101)	Load miss alias	Counts loads that alias against an entry already in the L1 miss queue and stalled.
46 (010_1110)	Load miss alias on touch	Counts loads that alias against a touch in the L1 miss queue.
47 (010_1111)	Touch alias	Counts touches that alias against an entry already in the L1 miss queue.
48 (011_0000)	L1 data snoop hit in L1 castout queue	Counts snoop accesses to the L1 data cache that hit in the L1 castout queue and create a push.
49 (011_0001)	L1 data snoop hit castout	Counts snoop accesses to the L1 data cache that hit in a castout and were retried (pre-L1 castout queue or bottom of L1 castout queue).
50 (11_0010)	L1 data snoop hits	Counts snoop accesses to the L1 data cache that hit regardless of the cache state (shared, exclusive, or modified).
51 (11_0011)	Write-through stores	Counts write-through stores sent to the memory subsystem after gathering.
52 (11_0100)	Cache-inhibited stores	Counts cache-inhibited stores sent to the memory subsystem after gathering.
53 (11_0101)	L1 data load hit	Counts each L1 data cache load hit. It does not include MMU table search lookups or touches.
54 (11_0110)	L1 data touch hit	Counts once per **dcbt** or **dcbtst** instruction or **dst***x* cache line fetch that hits in the L1 data cache.
55 (11_0111)	L1 data store hit	Counts write-back store hit attempts that successfully hit in the L1 data cache. Does not count if a store hits on a shared cache line. A gathered stores is considered a hit.
56 (11_1000)	L1 data total hits	Counts L1 data cache load, store, or touch hits.
57 (11_1001)	**dst** instructions dispatched	Counts **dst** instructions dispatched to VTQ. Includes speculative **dst** instructions that are canceled.
58 (11_1010)	Refreshed **dst**s	Counts **dst** operations issued to already active streams.
59 (11_1011)	Successful **dst**, **dstt**, **dstst**, and **dststt** table search operations	Counts non-faulting table search operations caused by data stream touch instructions (**dst**, **dstt**, **dstst**, and **dststt**).
60 (11_1100)	**dss** instructions completed	Counts **dss** instructions completed.
61(011_1101)	**dst** stream 0 cache line fetches	Counts **dst** stream 0 cache line fetches from the data stream engine (VT0) within the vector-touch queue (VTQ). This includes accesses that hit or miss in the L1 data cache.
62 (011_1110)	VTQ suspends due to change of context	Counts any number of VTQ streams that pause due to a change in MSR[PR] or MSR[DR].
63 (011_1111)	VTQ line fetch hit	Number of VTQ generated accesses that hit in the L1 data cache.
64 (100_0000	AltiVec load instructions completed	Counts completed AltiVec load instructions.
65 (100_0001)	Floating-point store instructions completed in LSU	Counts aligned floating-point store instructions completed. All misaligned floating-point store instructions completed are counted under PMC1, event number 88 (101_1000).
66 (100_0010)	Floating-point renormalization	Counts times a floating-point store single requires renormalization.

Table 11-9. PMC1 Events—MMCR0[PMC1SEL] Select Encodings (continued)

Number	Event	Description
67 (100_0011)	Floating-point denormalization	Counts times a floating-point store double requires denormalization.
68 (100_0100)	Floating-Point store causes stall in LSU	Counts cycles a floating-point store in the FSQ results in a store not being able to complete.
69 (100_0101)	—	Reserved
70 (100_0110)	Load/store true alias stall	Counts times the load or store is stalled due to a true alias.
71 (100_0111)	LSU indexed alias stall	Counts times the LSU RA latch is stalled due to a true or indexed alias.
72 (100_1000)	LSU alias versus FSQ/WB0/WB1	Counts times an alias occurs in LSU for a load versus the finished store queue or write-back stages.
73 (100_1001)	LSU alias versus CSQ	Counts times alias occurs in LSU for a load versus the completed store queue (CSQ).
74 100_1010)	LSU load-hit line alias versus CSQ0	Counts times line alias occurs in LSU for a load hit versus a completed store in CSQ0.
75 100_1011)	LSU load-miss line alias versus CSQ0	Counts times line alias occurs in LSU for a load miss versus completed store in CSQ0.
76 (100_1100)	LSU touch alias versus FSQ/WB0/WB1	Counts times alias occurs in LSU for a touch versus the finished store queue or write-back stages.
77 (100_1101)	LSU touch alias versus CSQ	Counts times alias occurs in the LSU for a touch versus the completed store queue.
78 (100_1110)	LSU LMQ full stall	Counts times the LSU RA latch is stalled while L1 miss queue (LMQ) is full.
79 (100_1111)	Floating-point load instruction completed in LSU	Counts times a floating-point load instruction is completed in the LSU.
80 (101_0000)	Floating-point load single instruction completed in LSU	Counts times a floating-point load single instruction is completed in the LSU.
81 (101_0001)	Floating-point load double completed in LSU	Counts times a floating-point load double instruction is finished in the LSU.
82 (101_0010)	LSU RA latch stall	Counts times the LSU RA latch is stalled for any reason.
83 (101_0011)	LSU load versus store queue alias stall	Counts times an LSU stall exists due to a true alias between a load or touch and the store queue (both Finished store queue(FSQ) and completed store queue (CSQ)).
84 (101_0100)	LSU LMQ index alias	Counts times an LSU stall exists due to an index alias against the LMQ.
85 (101_0101)	LSU store queue index alias	Counts times an LSU stall exists due to an index alias against the store queue.
86 (101_0110)	LSU CSQ forwarding	Counts times the completed store queue forwards to a load in the LSU.
87 (101_0111)	LSU misalign load finish	Counts times a misaligned load finishes in the LSU.
88 (101_1000)	LSU misalign store complete	Counts times a misaligned store completes in the LSU.
89 (101_1001)	LSU misalign stall	Counts times reservation station 0 (RS0) is misaligned and pending a misalign stall.
90 (101_1010)	Floating-point 1/4 FPSCR renames busy	Counts times the FPSCR rename is 1/4 busy.

Table 11-9. PMC1 Events—MMCR0[PMC1SEL] Select Encodings (continued)

Number	Event	Description
91 (101_1011)	Floating-point 1/2 FPSCR renames busy	Counts times the FPSCR rename is 1/2 busy.
92 (101_1100)	Floating-point 3/4 FPSCR renames busy	Counts times the FPSCR rename is 3/4 busy.
93 (101_1101)	Floating-point all FPSCR renames busy	Counts times the FPSCR renames are completely busy.
94 (101_1110)	Floating-point denormalized result	Counts times when a floating-point calculation results in a denormalized result.
95-127	—	Reserved

11.5.2 PMC2 Events

MMCR0[PMC2SEL] specify the events associated with PMC2, as shown in Table 11-10.

Table 11-10. PMC2 Events—MMCR0[PMC2SEL] Select Encodings

Number	Event	Description
0 (00_0000)	Nothing	Register counter holds current value.
1 (00_0001)	Processor cycles	Counts every processor cycle.
2 (00_0010)	Instructions completed	Counts all completed PowerPC and AltiVec instructions. Load/store multiple instructions (**lmw**, **stmw**) and load/store string instructions (**lswi**, **lswx**, **stswi**, **stswx**) are counted only once. Does not include folded branches. The counter can increment by 0, 1, 2, or 3, depending on the number of completed instructions per cycle. Branch folding must be disabled (HID0[FOLD] = 0) in order to count all the instructions.
3 (00_0011)	TBL bit transitions	Counts transitions from 0 to 1 of TBL bits specified through MMCR0[TBSEL] 00 = uses the TBL[31] bit to count 01 = uses the TBL[23] bit to count 10 = uses the TBL[19] bit to count 11 = uses the TBL[15] bit to count
4 (00_0100)	Instructions dispatched	Counts dispatched instructions.The counter can increment by 0, 1, 2, or 3, depending on the number of completed instructions per cycle. Load/store multiple instructions (**lmw**, **stmw**) and load/store string instructions (**lswi**, **lswx**, **stswi**, **stswx**) are counted only once. This event includes instructions that are dispatched directly to the completion queue.
5 (00_0101)	Processor performance monitor exception	Counts the times the processor begins to generate its performance monitor exception condition. The performance monitor exception condition is set when the processor performance monitor counter is negative and the exception signaling is enabled via MMCR0[PMC1CE] or MMCR0[PMC*n*CE]. The MPC7450 does not require MMCR0[PMXE] to be set to allow the exception to occur.
6 (00_0110)	—	Reserved. Read as zero.
7 (00_0111)	External performance monitor signal	Counts times the external performance monitor signal ($\overline{\text{PMON\_IN}}$) transitions from negated to asserted.

Table 11-10. PMC2 Events—MMCR0[PMC2SEL] Select Encodings (continued)

Number	Event	Description
8 (00_1000)	VPU instructions completed	Counts VPU instructions completed.
9 (00_1001)	VFPU instructions completed	Counts VFPU instructions completed.
10 (00_1010)	VIU1 instructions completed	Counts VIU1 instructions completed.
11 (00_1011)	VIU2 instructions completed	Counts VIU2 instructions completed.
12 (00_1100)	**mtvscr** instructions completed	Counts completed **mtvscr** instructions.
13 (00_1101)	**mtvrsave** instructions completed	Counts completed **mtvrsave** instructions.
14 (00_1110)	Cycles a VPU instruction in the reservation station is waiting for an operand	Counts cycles an AltiVec instruction in the vector permute unit reservation station is waiting for an operand.
15 (00_1111)	Cycles a VFPU instruction in the reservation station is waiting for operand	Counts the cycles an AltiVec instruction in the VFPU reservation station is waiting for an operand.
16 (01_0000)	Cycles a VIU1 instruction in the reservation station is waiting for operand	Counts the cycles an AltiVec instruction in the VIU1 reservation station is waiting for an operand.
17 (01_0001)	Cycles a VIU2 instruction in the reservation station is waiting for operand	Counts the cycles an AltiVec instruction in the VIU2 reservation station is waiting for an operand.
18 (01_0010)	**mfvscr** synchronization	Counts cycles when the VFPU has a valid **mfvscr** instruction dispatched, but the instruction cannot be completed because it is not at the bottom of the completion queue.
19 (01_0011)	VSCR[SAT] set	Counts whenever VSCR[SAT] goes from 0 to 1.
20 (01_0100)	Store instructions	Counts store instructions completed. Store strings and store multiples count only once. This count does not include instructions like cache operations that do not actually perform a load or store. Instructions not counted in the store instructions event are: **sync**, **eciwx**, **ecowx**, **eieio**, **dcbf**, **dcbi**, **dcbst**, **dcbt**, **dcbtst**, **dcbz**, **icbi**, **tlbie**, **tlbld**, **tlbli**, **tlbsync**, **dcba**, **dst**, **dstt**, **dstst**, **dststt**, **dss**, and **dssall**.
21 (01_0101)	L1 instruction cache misses	Counts the L1 instruction cache misses. This does not include cache inhibited or cache-disabled accesses.

Table 11-10. PMC2 Events—MMCR0[PMC2SEL] Select Encodings (continued)

Number	Event	Description
22 (01_0110)	L1 data snoops	Counts snoop accesses to the L1 data cache.
23 (01_0111)	L1 data total misses	Counts L1 data cache load, store, or touch misses.
24 (01_1000)	Dispatches to FPR issue queue	Counts the instructions dispatched to the FPR issue queue. This includes instructions in the speculative paths. Instructions executed by the FPU are dispatched to the FPR issue queue.
25 (01_1001)	LSU instructions completed	Counts LSU instructions completed
26 (01_1010)	Load instructions	Counts the load instructions completed. Load strings and load multiples only count once. This count does not include instructions like cache operations that do not actually perform a load or store. Instructions not counted in the load instructions event are: **sync**, **eciwx**, **ecowx**, **eieio**, **dcbf**, **dcbi**, **dcbst**, **dcbt**, **dcbtst**, **dcbz**, **icbi**, **tlbie**, **tlbld**, **tlbli**, **tlbsync**, **dcba**, **dst**, **dstt**, **dstst**, **dststt**, **dss**, **dssall**, and **lswx** with length zero (XER[25–31] = 0).
27 (01_1011)	Store string and store multiple instructions	Counts store string and store multiple instructions completed. Store strings and store multiples are counted only once. A **stswx** instruction with length zero (XER[25–31] = 0) is counted once.
28 (01_1100)	**tlbie** instructions completed	Counts **tlbie** instructions completed.
29 (01_1101)	**lwarx** instructions completed	Counts **lwarx** instructions completed.
30 (01_1110)	**mfspr** instructions completed	Counts **mfspr** instructions completed.This count does not include **mfvscr** instructions.
31 (01_1111)	Refetch serialization	Counts when a refetch serialization occurs for the following cases: • **isync** completes • **sc** completes • **rfi** completes • When an instruction that sets the XER[SO] bit completes (Changes XER[S0] from zero to one.) and when XER[SO] is cleared by a **mtspr** instruction • Exceptions taken • Tracing is enabled (MSR[SE] = 1 or MSR[BE] = 1) and a branch is speculative at branch processing time • Floating-point exception cases where the *Programming Environments Manual* specifies that the target FPR is unchanged. • **dcba** to a page marked as write-through or cache-inhibited.
32 (10_0000)	Completion queue entries over MMCR0[THRESHOLD] value	Counts the cycles when the valid completion queue entries are greater than or equal to the MMCR0[THRESHOLD] value. This event does not scale the MMCR0[THRESHOLD] value.
33 (10_0001)	Completing one instruction	Counts cycles in which exactly 1 instruction is completed.
34 (10_0010)	Two instructions dispatched	Counts cycles in which exactly 2 instructions are dispatched.

Table 11-10. PMC2 Events—MMCR0[PMC2SEL] Select Encodings (continued)

Number	Event	Description
35 (10_0011)	ITLB non-speculative misses	Counts times that a requested non-speculative address translation was not in the instruction TLB.
36 (10_0100)	Cycles waiting from L1 instruction cache miss	Counts cycles spent waiting for L1 instruction cache miss. This includes all instruction fetches, both cacheable and cache-inhibited. It counts from when the miss is detected until either the data is returned or the request is cancelled.
37 (10_0101)	L1 data load access miss	Counts L1 data cache load access misses. This does not include MMU table search lookups or touches.
38 (10_0110)	L1 data touch miss	Counts once for every **dst***x* cache line fetch, **dcbt**, or **dcbtst** L1 data cache miss that causes an L1 data cache reload
39 (10_0111)	L1 data store miss	Counts write-back store attempts that missed in the L1 data cache. This counts only once on gathered stores, if gathered before the cache access.
40 (10_1000)	L1 data touch miss cycles	Counts cycles spent waiting for L1 data cache touch misses from when the miss is detected until either the data is returned or the request is cancelled.
41 (10_1001)	L1 data cycles used	Counts cycles when the L1 data cache is used for any reason but does not include snoop accesses. The count value indicates the L1 data cache bandwidth consumed when compared to the number of processor cycles elapsed.
42 (10_1010)	**dst** stream 1 cache line fetches	Counts **dst** stream 1 cache line fetches from the data stream engine (VT1) within the vector touch queue (VTQ). This includes accesses that hit or miss in the L1 data cache.
43 (10_1011)	VTQ stream cancelled prematurely	Counts times when a VTQ stream is cancelled due to branch speculation cancel, inappropriate translation protection, or WIMG. This does not include cases where the VTQ stream is cancelled by reaching the end of a stream, refresh, or **dss**. This counter can increment by 0,1, 2, 3 or 4 at a time.
44 (10_1100)	VTQ resumes due to change of context	Counts any time the VTQ streams resume due to change in MSR[PR] or MSR[DR].
45 (10_1101)	VTQ line fetch miss	Counts VTQ generated accesses that miss in the L1 data cache.
46 (10_1110)	VTQ line fetch	Counts all VTQ fetch attempts. This includes all VTQ generated accesses that hit or miss in the L1 data cache.
47 (10_1111)	TLBIE snoops	Counts the TLB invalidations performed due to another master's TLBIE broadcast.
48 (11_0000)	L1 instruction cache reloads	Counts times that the L1 instruction cache is reloaded with a new cache line. This does not include cache-inhibited accesses. It does include accesses with the cache disabled and accesses that miss in the data cache when all ways are locked.
49 (11_0001)	L1 data cache reloads	Counts times that the L1 data cache is reloaded with a new cache line. This does not include cache-inhibited accesses, accesses with the cache disabled, or accesses that miss in the data cache when all ways are locked.
50 (011_0010)	L1 data cache castouts to L2	Counts L1 data cache castouts to the L2 cache.

Table 11-10. PMC2 Events—MMCR0[PMC2SEL] Select Encodings (continued)

Number	Event	Description
51 (011_0011)	Store merge/gather	Counts store operations that are merged with other store operations in the completed store queue (CSQ).
52 (011_0100)	Cacheable store merge to 32 bytes	Counts times all 32 bytes have been merged in the completed store queue (CSQ) to allow a full cache line write operation.
53 (011_0101)	Data breakpoint matches	Counts times a data address breakpoint exception is signalled.
54 (011_0110)	Fall-through branches processed	Counts branches that were either predicted or resolved as not-taken. This includes branches that are in a speculative path that might later be thrown away due to another previously predicted branch that mispredicts.
55 (011_0111)	First speculative branch buffer resolved correctly	Counts branches in the first prediction buffer that resolve correctly. out of order branch resolution means that some parts of this count may be due to branches in a speculative path that resolve correctly, but the speculative path is later mispredicted.
56 (011_1000)	Second speculation buffer active	Counts the cycles that a predicted branch is active in the second speculation buffer.
57 (011_1001)	BPU Stall on LR dependency	Counts the cycles the branch processing unit (BPU) stalls due to the link register (LR) being unresolved. If the link stack is enabled, a stall on LR dependency occurs only when the LR is unavailable and the link stack is empty. The count includes stalls down speculative paths.
58 (011_1010)	BTIC miss	Counts branch target instruction cache (BTIC) misses for taken branches. Note that this count includes taken branches that are in speculative paths.
59 (011_1011)	Branch link stack correctly resolved	Counts branches that use link stack prediction and resolve correctly. This count includes branches that are in speculative paths.
60 (011_1100)	FPR issue stalled	Counts times an instruction in the FPR issue queue could not be issued. This should only occur if the FPU is busy when an instruction is ready to issue.
61(011_1101)	Switches between Privileged and User	Counts times the MSR[PR] bit gets set and cleared.
62 (011_1110)	LSU completes floating-point store single	Counts aligned floating-point store single instructions completed. All misaligned floating-point store instructions completed are counted under PMC1, event number 88 (101_1000).
63 (011_1111)	—	Reserved

11.5.3 PMC3 Events

Bits MMCR1[PMC3SEL] specify events associated with PMC3, as shown in Table 11-11.

Table 11-11. PMC3 Events—MMCR1[PMC3SEL] Select Encodings

Number	Event	Description
0 (0_0000)	Nothing	Register counter holds current value.
1 (0_0001)	Processor cycles	Counts every processor clock cycle.

Table 11-11. PMC3 Events—MMCR1[PMC3SEL] Select Encodings (continued)

Number	Event	Description
2 (0_0010)	Instructions completed	Counts all completed PowerPC and AltiVec instructions. Load/store multiple instructions (**lmw**, **stmw**) and load/store string instructions (**lswl**, **lswx**, **stswl**, **stswx**) are only counted once. Does not include folded branches. The counter can increment by 0, 1, 2, or 3, depending on the number of completed instructions per cycle. Branch folding must be disabled (HID0[FOLD] = 0) in order to count all the instructions.
3 (0_0011)	TBL bit transitions	Counts transitions from 0 to 1 of TBL bits specified through MMCR0[TBSEL] 00 = uses the TBL[31] bit to count 01 = uses the TBL[23] bit to count 10 = uses the TBL[19] bit to count 11 = uses the TBL[15] bit to count
4 (0_0100)	Instructions dispatched	Counts dispatched instructions.The counter can increment by 0, 1, 2, or 3, depending on the number of completed instructions per cycle. Load/store multiple instructions (**lmw**, **stmw**) and load/store string instructions (**lswl**, **lswx**, **stswl**, **stswx**) are counted only once. This event includes instructions that are dispatched directly to the completion queue.
5 (0_0101)	Processor performance monitor exception	Counts times the processor begins to generate its performance monitor exception condition. The performance monitor exception condition is set when the processor performance monitor counter is negative and its exception signaling is enabled via MMCR0[PMC1CE] or MMCR0[PMC*n*CE]. The MPC7450 does not require MMCR0[PMXE] to be set to allow the exception to occur.
6 (0_0110)	—	Reserved Read as zero.
7 (0_0111)	External performance monitor signal	Counts times the external performance monitor signal ($\overline{\text{PMON\_IN}}$) transitions from negated to asserted.
8 (0_1000)	Completing two instruction	Counts cycles when exactly two instruction are completed.
9 (0_1001)	One instruction dispatched	Counts cycles when exactly one instruction is dispatched.
10 (0_1010)	Dispatches to VR issue queue	Counts the instructions dispatched to the vector register (VR) issue queue. This includes instructions in speculative paths. AltiVec instructions are executed by the VPU, VIU1, VIU2, and VFPU are dispatched to the VR issue queue.
11 (0_1011)	VR Stalls	Counts when an instruction in the vector register (VR) issue queue could not be issued. This counter can be incremented by 0, 1, or 2. An AltiVec instruction cannot be issued when its vector execution unit is busy or the AltiVec instruction ahead in the AltiVec issue queue could not be issued.
12 (0_1100)	GPR rename buffer entries over MMCR0[THRESHOLD]	Counts the cycles when the number of valid GPR rename buffers is greater than or equal to the MMCR0[THRESHOLD] value. This event does not scale the MMCR0[THRESHOLD] value.
13 (0_1101)	FPR issue queue entries	Counts the number of valid FPR issue queue entries each cycle.
14 (0_1110)	FPU instructions	Counts FPU instructions completed.
15 (0_1111)	**stwcx**. instructions	Counts **stwcx.** instructions completed.

Table 11-11. PMC3 Events—MMCR1[PMC3SEL] Select Encodings (continued)

Number	Event	Description
16 (1_0000)	Load string and multiple instruction pieces	Counts pieces of load string and load multiple instructions that are completed. An **lswx** instruction of length zero is counted once if MSR[SE] is set; otherwise it is not counted.
17 (1_0001)	ITLB hardware table search cycles over threshold	Counts times an instruction TBL hardware search operation for an instruction fetch requires more than the threshold number of cycles to complete. This includes table search operations that do not find any matching PTE entry in the page table. This event scales the MMCR0 threshold value as specified by MMCR2[THRESHMULT].
18 (1_0010)	DTLB misses	Counts times a needed non-speculative data address translation was not in the DTLB.
19 (1_0011)	Cancelled L1 instruction cache misses	Counts cacheable instruction accesses that miss in the instruction cache, but are cancelled before they are accepted by the memory subsystem.
20 (1_0100)	L1 data cache operation hit	Counts cache operations that hit in the L1 data cache (**dcbf**, **dcbst**).
21 (1_0101)	L1 data load miss cycles	Counts cycles spent waiting for L1 data cache misses in the LMQ. It counts from when the miss is detected until either the data is returned or the request is cancelled. Misses in the LMQ include all load and touch operations. Note that a load miss is only counted if it is the oldest entry of the LMQ.
22 (1_0110)	L1 data Pushes	Counts L1 data pushes caused by snoops to modified cache lines.
23 (1_0111)	L1 data total miss	Counts L1 data cache load, store, or touch misses.
24 (1_1000)	VT2 fetches	Counts fetch attempts from the data stream engine 2 (VT2) within the vector-touch queue (VTQ). This includes accesses that hit or miss in the L1 data cache.
25 (1_1001)	Taken branches that are processed	Counts branches that were either predicted or resolved taken. This includes branches that are in a speculative path. This also includes branches that are in a speculative path that might later be thrown away due to another previously predicted branch that mispredicts.
26 (1_1010)	Branch flushes	Counts flushes for clearing mispredicted instructions out of the completion queue.
27 (1_1011)	Second speculative branch buffer resolved correctly	Counts branches in the second prediction buffer that resolve correctly. Out of order branch resolution means that some parts of this count may be due to branches in a speculative path that resolve correctly, but the speculative path is later mispredicted.
28 (1_1100)	Third speculation buffer active	Counts cycles that a third predicted branch is active.
29 (1_1101)	Branch unit stall on CTR dependency	Counts cycles the branch unit is stalled due to the counter register (CTR) being unresolved. Includes stalls down speculative paths.
30 (1_1110)	Fast BTIC hit	Counts FBTIC hits for taken branches. This number should be greater than or equal to the BTIC hit count. The difference between this count and the BTIC hit count provides the number of aliased BTIC hits. Aliased BTIC hits force a hiccup in the fetch pipe, delaying when the instructions at the branch target address are available for dispatch. Note that this count includes taken branches that are in speculative paths.
31 (1_1111)	Branch Link Stack Mispredicted	Counts branches that use Link Stack Prediction and resolve incorrectly. This count includes branches that are in speculative paths.

11.5.4 PMC4 Events

Bits MMCR1PMC4SEL] specify events associated with PMC4, as shown in Table 11-12.

Table 11-12. PMC4 Events—MMCR1[PMC4SEL] Select Encodings

Number	Event	Description
0 (0_0000)	Nothing	Register counter holds current value.
1 (0_0001)	Processor cycles	Counts every processor cycle.
2 (0_0010)	Instructions completed	Counts all completed PowerPC and AltiVec instructions. Load/store multiple/string instructions are only counted once even though they are broken up into pieces. Does not include folded branches. To count all instructions, HID0[FOLD] must be cleared to disable branch folding.
3 (0_0011)	TBL bit transitions	Counts transitions from 0 to 1 of TBL bits specified through MMCR0[TBSEL] 00 = uses the TBL[31] bit to count 01 = uses the TBL[23] bit to count 10 = uses the TBL[19] bit to count 11 = uses the TBL[15] bit to count
4 (0_0100)	Instructions dispatched	Counts dispatched instructions.The counter can increment by 0, 1, 2, or 3, depending on the number of completed instructions per cycle. Load/store multiple instructions (**lmw**, **stmw**) and load/store string instructions (**lswl**, **lswx**, **stswl**, **stswx**) are only counted once. This event includes instructions that are dispatched directly to the completion queue.
5 (0_0101)	Processor performance monitor exception	Counts the times the processor begins to generate its performance monitor exception condition. The performance monitor exception condition is set when the processor performance monitor counter is negative and its exception signaling is enabled via MMCR0[PMC1CE] or MMCR0[PMC*n*CE]. The MPC7450 does not require MMCR0[PMXE] to be set to allow the exception to occur.
6 (0_0110)	—	Reserved. Read as zero.
7 (0_0111)	External performance monitor signal	Counts times the external performance monitor signal ($\overline{\text{PMON\_IN}}$) transitions from negated to asserted.
8 (0_1000)	Instructions completed in VPU	Counts completed VPU instructions.
9 (0_1001)	Instructions completed in VFPU	Counts completed vector VFPU instructions.
10 (0_1010)	VIU1 instructions completed	Counts completed VIU1 instructions.
11 (0_1011)	VIU2 Instructions completed	Counts completed VIU2 instructions.
12 (0_1100)	**mtvscr** Instructions completed	Counts completed **mtvscr** instructions.
13 (0_1101)	**mtvrsave** Instructions completed	Counts completed **mtvrsave** instructions.

Table 11-12. PMC4 Events—MMCR1[PMC4SEL] Select Encodings (continued)

Number	Event	Description
14 (0_1110)	Completing 3 instructions	Counts cycles where three instructions are completed.
15 (0_1111)	Dispatching 0 instructions	Counts cycles where zero instructions are dispatched.
16 (1_0000)	GPR issue queue entries over threshold	Counts cycles when the valid GPR issue queue entries are greater than or equal to the MMCR0[THRESHOLD] value. This event cannot scale the MMCR0[THRESHOLD] value.
17 (1_0001)	GPR issue queue stalled	Counts cycles that instructions in the GPR issue queue are not issued. This value only increments by 1 on any given cycle. A GPR instruction is not issued when its unit is busy, or when an instruction ahead of it in the GPR issue queue could not issue. An IU1 instruction goes to any non-busy IU1 so it only stalls if more IU1 instructions are trying to issue than there are non-busy IU1 units.
18 (1_0010)	IU1 instructions	Counts completed IU1 instructions.
19 (1_0011)	dssall instructions	Counts completed **dssall** instructions.
20 (1_0100)	**tlbsync** instructions	Counts completed **tlbsync** instructions.
21 (1_0101)	**sync** instructions	Counts completed **sync** instructions.
22 (1_0110)	Store string and multiple instruction pieces	Counts completed pieces of store string and store multiple instructions. An **stswx** instruction of length zero is counted once.
23 (1_0111)	DTLB hardware table search cycles	Counts cycles spent performing hardware table search operations for DTLB misses. A hardware table search begins when the DTLB determines that it has missed and all instructions ahead of the DTLB miss have completed. A hardware table search ends when the page table entry (PTE) or a page fault signal is returned by the table search engine. The number of cycles does NOT include the time it takes the machine to drain before the hardware table search begins.
24 (1_1000)	Snoop retries	Counts the number of load-store snoops that are retried by the load-store. This includes external snoops which are retried because of a load-store collision, as well as internal load-store self-snoop retries. It does not include snoops which are retried because of an MSS collision or busy condition. An example of an internal self-snoop collision is a load L1 miss which collides with a castout in the L1 castout queue. This type of collision is handled through internal snoop retry instead of load-store pipeline stall.
25 (1_1001)	Successful **stwcx**.	Counts **stwcx**. instructions that completed with reservation intact.
26 (1_1010)	**dst** stream 3 cache line fetches	Counts **dst** stream 3 cache line fetches from the data stream engine (VT3) within the vector-touch queue (VTQ). This includes accesses that hit or miss in the L1 data cache.
27 (1_1011)	Third speculative branch buffer resolved correctly	Counts branches in the third prediction buffer that resolve correctly. Out-of-order branch resolution means that some parts of this count may be due to branches in a speculative path that resolve correctly, but the speculative path is later mispredicted.
28 (1_1100)	Mispredicted branches	Counts mispredicted branches. Due to out-of-order branch resolution, this count includes mispredicted branches down speculative paths that may later be mispredicted themselves.

Table 11-12. PMC4 Events—MMCR1[PMC4SEL] Select Encodings (continued)

Number	Event	Description
29 (1_1101)	Folded branches	Counts branches actually folded in the instruction queue. Note that this count includes branches that are on speculative paths.
30 (1_1110)	Floating-point store double completes in LSU	Counts aligned floating-point store double instructions completed. All misaligned floating-point store instructions completed are counted under PMC1, event number 88 (0x101_1000).
31 (1_1111)	—	Reserved

11.5.5 PMC5 Events

Bits MMCR1[PMC5SEL] specify events associated with PMC5, as shown in Table 11-13.

Table 11-13. PMC5 Events—MMCR1[PMC5SEL] Select Encodings

Number	Event	Description
0 (0_0000)	Nothing	Register counter holds current value.
1 (0_0001)	Processor cycles	Counts every processor clock cycle.
2 (0_0010)	L2 cache hits	Counts accesses from the processor that hit in the L2 for loads, caching-allowed write-back stores, **dcbz**, instruction fetches, and touches.
3 (0_0011) [1]	L3 cache hits	Counts accesses from the processor that hit in the L3 for loads, caching-allowed write-back stores, **dcbz**, instruction fetches, and touches.
4 (0_0100) [1]	L2 instruction cache misses	Counts instruction accesses from the processor that miss in the L2 cache.
5 (0_0101) [1]	L3 instruction cache misses	Counts instruction accesses from the processor that miss in the L3 cache.
6 (0_0110)	L2 data cache misses	Counts data accesses from the processor that miss in the L2 cache (loads, caching-allowed write-back stores, **dcbz**, and touches).
7 (0_0111) [1]	L3 data cache misses	Counts data accesses from the processor that miss in the L3 cache (loads, caching-allowed write-back stores, **dcbz**, and touches).
8 (0_1000)	L2 load hits	Counts load accesses from the processor that hit in the L2 cache.
9 (0_1001)	L2 store hits	Counts caching-allowed write-back store accesses from the processor that hit in the L2 cache.
10 (0_1010) [1]	L3 load hits	Counts load accesses from the processor that hit in the L3 cache.
11 (0_1011) [1]	L3 store hits	Counts caching-allowed write-back store accesses from the processor that hit in the L3 cache.
12 (0_1100)	—	Reserved
13 (0_1101)	L2 touch hits	Counts touch accesses (**dcbt**, **dcbtst**, and VTQ) from the processor that hit in the L2 cache.
14 (0_1110) [1]	L3 touch hits	Counts touch accesses (**dcbt**, **dcbtst**, and VTQ) from the processor that hit in the L3 cache.

Table 11-13. PMC5 Events—MMCR1[PMC5SEL] Select Encodings (continued)

Number	Event	Description
15 (0_1111)	Snoop retries	Counts the number of internal requests that are internally retried. This includes load-store retries as well as some MSS collision cases (that would prevent an L2 hit from being considered good).
16 (1_0000)	Snoop modified	Counts times a snoop response to an access made by the processor is modified (internal snooping).
17 (1_0001)	Snoop valid	Counts times a snoop response to an access made by the processor is valid (internal snooping).
18 (1_0010)	Intervention	Counts local interventions serviced by the processor (internal snooping).
19 (1_0011)	L2 cache misses	Counts accesses from the processor that miss in the L2 for loads, caching-allowed write-back stores, **dcbz**, instruction fetches, and touches.
20 (1_0100) [1]	L3 cache misses	Counts accesses from the processor that miss in the L3 for loads, caching-allowed write-back stores, **dcbz**, instruction fetches, and touches.
21–31	—	Reserved

[1] Note that the L3 cache is not supported on the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448.

11.5.6 PMC6 Events

The event to be monitored can be chosen by setting MMCR1[15–20]. The selected events are counted beginning when MMCR1 is set until either MMCR1 is reset or a performance monitor exception is generated. Table 11-14 lists the selectable events and their encodings.

Table 11-14. PMC6 Events—MMCR1[PMC6SEL] Select Encodings

Number	Event	Description
0 (00_0000)	Nothing	Register counter holds current value.
1 (00_0001)	Processor cycles	Counts every processor cycle.
2 (00_0010)	L2 cache hits	Counts accesses from the processor that hit in the L2 for loads, caching-allowed write-back stores, **dcbz**, instruction fetches, and touches.
3 (00_0011)[1]	L3 cache hits	Counts accesses from the processor that hit in the L3 for loads, caching-allowed write-back stores, **dcbz**, instruction fetches, and touches.
4 (00_0100)	L2 instruction cache misses	Counts instruction accesses from the processor that miss in the L2 cache.
5 (00_0101) [1]	L3 instruction cache misses	Counts instruction accesses from the processor that miss in the L3 cache.
6 (00_0110)	L2 data cache misses	Counts data accesses from the processor that miss in the L2 cache (loads, caching-allowed write-back stores, **dcbz**, and touches).
7 (00_0111) [1]	L3 data cache misses	Counts data accesses from the processor that miss in the L3 (loads, caching-allowed write-back stores, **dcbz**, and touches).
8 (00_1000)	L2 cache castouts	Counts L2 cache castouts.

Table 11-14. PMC6 Events—MMCR1[PMC6SEL] Select Encodings (continued)

Number	Event	Description
9 (00_1001) [1]	L3 castouts	Counts L3 cache castouts.
10 (00_1010)	L2SQ full cycles	Counts cycles the L2 castout queue (L2SQ) is full (not counting the reserved push slot).
11 (00_1011) [1]	L3SQ full cycles	Counts cycles the L3 castout queue (L3SQ) is full (not counting the reserved push slot).
12 (00_1100)	—	Reserved
13 (00_1101)	L2 touch hits	Counts touch accesses (**dcbt**, **dcbtst**, and VTQ) from the processor that hit in the L2 cache.
14 (00_1110) [1]	L3 touch hits	Counts touch accesses (**dcbt**, **dcbtst**, and VTQ) from the processor that hit in the L3 cache.
15 (00_1111)	Snoop retries	Counts times a snoop response to any access is “retry.”
16 (01_0000) [1]	RAQ full cycles	Counts cycles the L3 read queue is full.
17 (01_0001) [1]	WAQ full cycles	Counts cycles the L3 write queue is full.
18 (01_0010)	Intervention	Counts local interventions serviced by the processor (internal snooping).
19 (01_0011)	L1 external Interventions	Counts L1 external interventions (External snoop hits modified in the L1 data cache).
20 (01_0100)	L2 external Interventions	Counts L2 interventions caused by external snoops to modified blocks.
21 (01_0101) [1]	L3 external Interventions	Counts L3 interventions caused by external snoops to modified blocks.
22 (01_0110) [1]	External interventions	Counts external interventions serviced. This is the sum of the L1, L2, and L3 external interventions.
23 (01_0111)	External pushes	Counts times an external snoop causes a push or upgraded castout.
24 (01_1000)	External snoop retry	Counts the number of external snoops that get a retry response.
25 (01_1001)	DTQ full cycles	Counts cycles the DTQ is full (not counting reserved push slot).
26 (01_1010)	Bus retry	Counts transactions that were initiated by this processor that were retried on the system interface.
27 (01_1011)	L2 valid request	Counts requests serviced by the L2 cache.
28 (01_1100)	BORDQ full	Counts cycles the BORDQ (bus outstanding read queue) is full. The entries in BORDQ correspond directly to the addresses of entries in the LLQ.
29 (01_1101)	L2 cache misses	Counts accesses from the processor that miss in the L2 for loads, caching-allowed write-back stores, **dcbz**, instruction fetches, and touches.
30 (01_1110) [1]	L3 cache misses	Counts accesses from the processor that miss in the L3 for loads, caching-allowed write-back stores, **dcbz**, instruction fetches, and touches.
31 (01_1111) [1]	L3 cache hits	Counts accesses from the processor that hit in the L3 for loads, caching-allowed write-back stores, **dcbz**, instruction fetches, and touches.
32 (10_0000) [1]	L3 cache misses	Counts accesses from the processor that miss in the L3 for loads, caching-allowed write-back stores, **dcbz**, instruction fetches, and touches.

Table 11-14. PMC6 Events—MMCR1[PMC6SEL] Select Encodings (continued)

Number	Event	Description
33 (10_0001) [1]	L3 instruction cache misses	Counts instruction accesses from the processor that miss in the L3 cache.
34 (10_0010) [1]	L3 data cache misses	Counts data accesses from the processor that miss in the L3 cache (loads, caching-allowed write-back stores, **dcbz**, and touches).
35 (10_0011) [1]	L3 load hits	Counts load accesses from the processor that hit in the L3 cache.
36 (10_0100) [1]	L3 store hits	Counts caching-allowed store accesses from the processor that hit in the L3 cache.
37 (10_0101) [1]	L3 touch hits	Counts touch accesses (**dcbt**, **dcbtst**, and VTQ) from the processor that hit in the L3 cache.
38 (10_0110)	—	Reserved
39 (10_0111)	—	Reserved
40 (10_1000)	—	Reserved
41 (10_1001)	—	Reserved
42 (10_1010)	Bus $\overline{\text{TA}}$s for reads	Counts external $\overline{\text{TA}}$s received on the bus for all read operations initiated by the processor.
43 (10_1011)	Bus $\overline{\text{TA}}$s for writes	Counts external $\overline{\text{TA}}$s received on the bus for all write operations initiated by the processor. This includes $\overline{\text{TA}}$s to which the processor is providing intervention data.
44 (10_1100)	Bus reads not retried	Counts load-type operations initiated by the processor on the external bus that complete with a non-retry response.
45 (10_1101)	Bus writes not retried	Counts store-type operations initiated by the processor on the external bus that complete with a non-retry response. This event does not include external push operations that are counted in another event.
46 (10_1110)	Bus reads/writes not retries	Counts the total load-type, store-type, and external push operations initiated by the processor on the external bus that complete with a non-retry response.
47 (10_1111)	Bus retry due to L1 retry	Counts times retry is asserted on the external bus due to an internal L1 retry condition.
48 (11_0000)	Bus retry due to previous adjacent	Counts times retry is asserted on the external bus due to an internal previous adjacent retry condition.
49 (11_0001)	Bus retry due to collision	Counts times retry is asserted on the external bus due to an internal collision.
50 (11_0010)	Bus retry due to intervention ordering	Counts times retry is asserted on the external bus due to an intervention ordering condition.
51 (11_0011)	Snoop requests	Counts qualified snoop requests processed by the snooper.
52 (11_0100)	Prefetch engine request	Counts new prefetches allocated in the prefetch unit.
53 (11_0101)	Prefetch engine collision vs. load	Counts times a load collides with an outstanding prefetch request from the L2 prefetch engine while accessing L2 or L3[1].

Table 11-14. PMC6 Events—MMCR1[PMC6SEL] Select Encodings (continued)

Number	Event	Description
54 (11_0110)	Prefetch engine collision vs. store	Counts times a store collides with an outstanding prefetch request from the L2 prefetch engine while accessing L2 or L3[1].
55 (11_0111)	Prefetch engine collision vs. i instruction fetch	Counts times an instruction fetch collides against an outstanding request from the L2 prefetch engine while accessing L2 or L3[1].
56 (11_1000)	Prefetch engine collision vs. load/store/instruction fetch	Counts times the L2 prefetch engine collides against an outstanding load, store, or instruction fetch in the load miss queue while accessing L2 or L3[1].
57 (11_1001)	Prefetch engine full	Counts times an L2 prefetch is not initiated because the prefetch engine is full while accessing L2 or L3[1].
58–63	—	Reserved

[1] Note that the L3 cache is not supported on the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448.

Appendix A
MPC7450 Instruction Set Listings

This appendix lists the MPC7450 microprocessor's instruction set as well as the additional PowerPC instructions not implemented in the MPC7450. The instructions are sorted by mnemonic, opcode, function, and form. This appendix also includes 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.

NOTES

> The MPC7450 is a 32-bit microprocessor, and does not implement any 64-bit instructions.
>
> 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 (Decimal and Hexadecimal)

Table A-1 shows the instructions implemented in the MPC7450. The instructions are listed in alphabetical order by their mnemonic name. The primary opcode (0–5) and secondary opcode (21-31) are decimal and hexadecimal values.

Key:

▨ Reserved bits

Table A-1. Instructions by Mnemonic (Dec, Hex)

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 (0x1F)	D	A	B	OE	0266 (0x10A)	Rc
addc*x*	31 (0x1F)	D	A	B	OE	0010 (0x00A)	Rc
adde*x*	31 (0x1F)	D	A	B	OE	0138 (0x08A)	Rc
addi	14 (0xE)	D	A	SIMM			
addic	12 (0xC)	D	A	SIMM			
addic.	13 (0xD)	D	A	SIMM			
addis	15 (0xF)	D	A	SIMM			
addme*x*	31 (0x1F)	D	A	0_0000	OE	0234 (0x0EA)	Rc
addze*x*	31 (0x1F)	D	A	0_0000	OE	0202 (0x0CA)	Rc
and*x*	31 (0x1F)	S	A	B	0028 (0x01C)		Rc

Table A-1. Instructions by Mnemonic (Dec, Hex) (continued)

Name	0–5	6	7–8	9	10	11–15	16–20	21	22–29	30	31
andc*x*	31 (0x1F)	S				A	B	0060 (0x03C)			Rc
andi.	28 (0x1C)	S				A	UIMM				
andis.	29 (0x1D)	S				A	UIMM				
b*x*	18 (0x12)	LI								AA	LK
bc*x*	16 (0x10)	BO				BI	BD			AA	LK
bcctr*x*	19 (0x13)	BO				BI	0_0000	00528 (0x210)			LK
bclr*x*	19 (0x13)	BO				BI	0_0000	0016 (0x010)			LK
cmp	31 (0x1F)	crfD		0	L	A	B	0000 (0x000)			0
cmpi	11 (0x0B)	crfD		0	L	A	SIMM				
cmpl	31 (0x1F)	crfD		0	L	A	B	0032 (0x020)			0
cmpli	10 (0x0A)	crfD		0	L	A	UIMM				
cntlzw*x*	31 (0x1F)	S				A	0_0000	0026 (0x01A)			Rc
crand	19 (0x13)	crbD				crbA	crbB	0257 (0x101)			0
crandc	19 (0x13)	crbD				crbA	crbB	0129 (0x081)			0
creqv	19 (0x13)	crbD				crbA	crbB	0289 (0x121)			0
crnand	19 (0x13)	crbD				crbA	crbB	0225 (0x0E1)			0
crnor	19 (0x13)	crbD				crbA	crbB	0033 (0x21)			0
cror	19 (0x13)	crbD				crbA	crbB	0449 (0x1C1)			0
crorc	19 (0x13)	crbD				crbA	crbB	0417 (0x1A1)			0
crxor	19 (0x13)	crbD				crbA	crbB	0193 (0C1)			0
dcba[1]	31 (0x1F)	000_00				A	B	0758 (0x2F6)			0
dcbf	31 (0x1F)	000_00				A	B	0086 (0x056)			0
dcbi[2]	31 (0x1F)	000_00				A	B	0470 (0x1D6)			0
dcbst	31 (0x1F)	000_00				A	B	0054 (0x036)			0
dcbt	31 (0x1F)	000_00				A	B	0278 (0x116)			0
dcbtst	31 (0x1F)	000_00				A	B	0246 (0x0F6)			0
dcbz	31 (0x1F)	000_00				A	B	1014 (0x3F6)			0
divw*x*	31 (0x1F)	D				A	B	OE	0491 (0x1EB)		Rc
divwu*x*	31 (0x1F)	D				A	B	OE	0459 (0x1CB)		Rc
dss[3]	31 (0x1F)	0	00	STRM		00_000	0_0000	0822 (0x336)			0
dssall[3]	31 (0x1F)	1	00	STRM		00_000	0_0000	0822 (0x336)			0
dst[3]	31 (0x1F)	0	00	STRM		A	B	0342 (0x156)			0
dstst[3]	31 (0x1F)	0	00	STRM		A	B	0374 (0x09C)			0

Table A-1. Instructions by Mnemonic (Dec, Hex) (continued)

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
dststt[3]	31 (0x1F)	1	00	STRM	A	B	0374 (0x176)		0
dstt[3]	31 (0x1F)	1	00	STRM	A	B	0342 (0x0B0)		0
eciwx[1]	31 (0x1F)	D			A	B	0310 (0x136)		0
ecowx[1]	31 (0x1F)	S			A	B	0438 (0x1B6)		0
eieio	31 (0x1F)	000_00			00_000	0_0000	0854 (0x356)		0
eqv*x*	31 (0x1F)	S			A	B	0284 (0x11C)		Rc
extsb*x*	31 (0x1F)	S			A	0_0000	0954 (0x3BA)		Rc
extsh*x*	31 (0x1F)	S			A	0_0000	0922 (0x39A)		Rc
fabs*x*	63 (0x3F)	D			00_000	B	0264 (0x108)		Rc
fadd*x*	63 (0x3F)	D			A	B	0000_0	0021 (0x015)	Rc
fadds*x*	59 (0x3B)	D			A	B	0000_0	0021 (0x015)	Rc
fcmpo	63 (0x3F)	crfD		00	A	B	0032 (0x020)		0
fcmpu	63 (0x3F)	crfD		00	A	B	0000 (0x000)		0
fctiw*x*	63 (0x3F)	D			00_000	B	0014 (0x00E)		Rc
fctiwz*x*	63 (0x3F)	D			00_000	B	0015 (0x00F)		Rc
fdiv*x*	63 (0x3F)	D			A	B	0000_0	0018 (0x012)	Rc
fdivs*x*	59 (0x3B)	D			A	B	0000_0	0018 (0x012)	Rc
fmadd*x*	63 (0x3F)	D			A	B	C	0029 (0x01D)	Rc
fmadds*x*	59 (0x3B)	D			A	B	C	0029 (0x01D)	Rc
fmr*x*	63 (0x3F)	D			00_000	B	0072 (0x48)		Rc
fmsub*x*	63 (0x3F)	D			A	B	C	0028 (0x01C)	Rc
fmsubs*x*	59 (0x3B)	D			A	B	C	0028 (0x01C)	Rc
fmul*x*	63 (0x3F)	D			A	0_0000	C	0025 (0x019)	Rc
fmuls*x*	59 (0x3B)	D			A	0_0000	C	0025 (0x019)	Rc
fnabs*x*	63 (0x3F)	D			00_000	B	0136 (0x88)		Rc
fneg*x*	63 (0x3F)	D			00_000	B	0040 (0x28)		Rc
fnmadd*x*	63 (0x3F)	D			A	B	C	0031 (0x01F)	Rc
fnmadds*x*	59 (0x3B)	D			A	B	C	0031 (0x01F)	Rc
fnmsub*x*	63 (0x3F)	D			A	B	C	0030 (0x01E)	Rc
fnmsubs*x*	59 (0x3B)	D			A	B	C	0030 (0x01E)	Rc
fres*x* [1]	59 (0x3B)	D			00_000	B	0000_0	0024 (0x018)	Rc
frsp*x*	63 (0x3F)	D			00_000	B	0012 (0xC)		Rc
frsqrte*x* [1]	63 (0x3F)	D			00_000	B	0000_0	0026 (0x01A)	Rc

Table A-1. Instructions by Mnemonic (Dec, Hex) (continued)

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
fsel_x_[1]	63 (0x3F)	D	A	B	C	0023 (0x017)	Rc
fsqrt_x_[4]	63 (0x3F)	D	00_000	B	0000_0	0022 (0x016)	Rc
fsqrts_x_[4]	59 (0x3B)	D	00_000	B	0000_0	0022 (0x016)	Rc
fsub_x_	63 (0x3F)	D	A	B	0000_0	0020 (0x014)	Rc
fsubs_x_	59 (0x3B)	D	A	B	0000_0	0020 (0x014)	Rc
icbi	31 (0x1F)	000_00	A	B	0982 (0x3D6)		0
isync	19 (0x13)	000_00	00_000	0_0000	0150 (0x096)		0
lbz	34 (0x22)	D	A	d			
lbzu	35 (0x23)	D	A	d			
lbzu_x_	31 (0x1F)	D	A	B	0119 (0x077)		0
lbzx	31 (0x1F)	D	A	B	087 (0x057)		0
lfd	50 (0x32)	D	A	d			
lfdu	51 (0x33)	D	A	d			
lfdux	31 (0x1F)	D	A	B	0631 (0x277)		0
lfdx	31 (0x1F)	D	A	B	0599 (0x257)		0
lfs	48 (0x30)	D	A	d			
lfsu	49 (0x31)	D	A	d			
lfsux	31 (0x1F)	D	A	B	0567 (0x237)		0
lfsx	31 (0x1F)	D	A	B	0535 (0x217)		0
lha	42 (0x2A)	D	A	d			
lhau	43 (0x2B)	D	A	d			
lhaux	31 (0x1F)	D	A	B	0375 (0x177)		0
lhax	31 (0x1F)	D	A	B	0343 (0x157)		0
lhbrx	31 (0x1F)	D	A	B	0790 (0x316)		0
lhz	40 (0x28)	D	A	d			
lhzu	41 (0x29)	D	A	d			
lhzux	31 (0x1F)	D	A	B	0311 (0x137)		0
lhzx	31 (0x1F)	D	A	B	0279 (0x117)		0
lmw[5]	46 (0x2E)	D	A	d			
lswi[5]	31 (0x1F)	D	A	NB	0597 (0x255)		0
lswx[5]	31 (0x1F)	D	A	B	0533 (0x215)		0
lvebx[3]	31 (0x1F)	**v**D	A	B	0007 (0x007)		0
lvehx[3]	31 (0x1F)	**v**D	A	B	0039 (0x027)		0

Table A-1. Instructions by Mnemonic (Dec, Hex) (continued)

<table>
<tr><th>Name</th><th>0</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th><th>17</th><th>18</th><th>19</th><th>20</th><th>21</th><th>22</th><th>23</th><th>24</th><th>25</th><th>26</th><th>27</th><th>28</th><th>29</th><th>30</th><th>31</th></tr>
<tr><td>lvewx[3]</td><td colspan="2">31 (0x1F)</td><td colspan="5">vD</td><td colspan="5">A</td><td colspan="5">B</td><td colspan="10">0071 (0x047)</td><td>0</td></tr>
<tr><td>lvsl[3]</td><td colspan="2">31 (0x1F)</td><td colspan="5">vD</td><td colspan="5">A</td><td colspan="5">B</td><td colspan="10">0006 (0x006)</td><td>0</td></tr>
<tr><td>lvsr[3]</td><td colspan="2">31 (0x1F)</td><td colspan="5">vD</td><td colspan="5">A</td><td colspan="5">B</td><td colspan="10">0038 (0x026)</td><td>0</td></tr>
<tr><td>lvx[3]</td><td colspan="2">31 (0x1F)</td><td colspan="5">vD</td><td colspan="5">A</td><td colspan="5">B</td><td colspan="10">0103 (0x067)</td><td>0</td></tr>
<tr><td>lvxl[3]</td><td colspan="2">31 (0x1F)</td><td colspan="5">vD</td><td colspan="5">A</td><td colspan="5">B</td><td colspan="10">0359 (0x167)</td><td>0</td></tr>
<tr><td>lwarx</td><td colspan="2">31 (0x1F)</td><td colspan="5">D</td><td colspan="5">A</td><td colspan="5">B</td><td colspan="10">0020 (0x014)</td><td>0</td></tr>
<tr><td>lwbrx</td><td colspan="2">31 (0x1F)</td><td colspan="5">D</td><td colspan="5">A</td><td colspan="5">B</td><td colspan="10">0534 (0x216)</td><td>0</td></tr>
<tr><td>lwz</td><td colspan="2">32 (0x20)</td><td colspan="5">D</td><td colspan="5">A</td><td colspan="16">d</td></tr>
<tr><td>lwzu</td><td colspan="2">33 (0x21)</td><td colspan="5">D</td><td colspan="5">A</td><td colspan="16">d</td></tr>
<tr><td>lwzux</td><td colspan="2">31 (0x1F)</td><td colspan="5">D</td><td colspan="5">A</td><td colspan="5">B</td><td colspan="10">0055 (0x037)</td><td>0</td></tr>
<tr><td>lwzx</td><td colspan="2">31 (0x1F)</td><td colspan="5">D</td><td colspan="5">A</td><td colspan="5">B</td><td colspan="10">0023 (0x017)</td><td>0</td></tr>
<tr><td>mcrf</td><td colspan="2">19 (0x13)</td><td colspan="3">crfD</td><td colspan="2">00</td><td colspan="3">crfS</td><td colspan="2">00</td><td colspan="5">0_0000</td><td colspan="10">0000 (0x000)</td><td>0</td></tr>
<tr><td>mcrfs</td><td colspan="2">63 (0x3F)</td><td colspan="3">crfD</td><td colspan="2">00</td><td colspan="3">crfS</td><td colspan="2">00</td><td colspan="5">0_0000</td><td colspan="10">0064 (0x040)</td><td>0</td></tr>
<tr><td>mcrxr</td><td colspan="2">31 (0x1F)</td><td colspan="3">crfD</td><td colspan="2">00</td><td colspan="5">00_000</td><td colspan="5">0_0000</td><td colspan="10">0512 (0x200)</td><td>0</td></tr>
<tr><td>mfcr</td><td colspan="2">31 (0x1F)</td><td colspan="5">D</td><td colspan="5">00_000</td><td colspan="5">0_0000</td><td colspan="10">0019 (0x013)</td><td>0</td></tr>
<tr><td>mffsx</td><td colspan="2">63 (0x3F)</td><td colspan="5">D</td><td colspan="5">00_000</td><td colspan="5">0_0000</td><td colspan="10">0583 (0x247)</td><td>Rc</td></tr>
<tr><td>mfmsr[2]</td><td colspan="2">31 (0x1F)</td><td colspan="5">D</td><td colspan="5">00_000</td><td colspan="5">0_0000</td><td colspan="10">0083 (0x053)</td><td>0</td></tr>
<tr><td>mfspr[6]</td><td colspan="2">31 (0x1F)</td><td colspan="5">D</td><td colspan="10">spr</td><td colspan="10">0339 (0x153)</td><td>0</td></tr>
<tr><td>mfsr[2]</td><td colspan="2">31 (0x1F)</td><td colspan="5">D</td><td>0</td><td colspan="4">SR</td><td colspan="5">0_0000</td><td colspan="10">0595 (0x099)</td><td>0</td></tr>
<tr><td>mfsrin[2]</td><td colspan="2">31 (0x1F)</td><td colspan="5">D</td><td colspan="5">00_000</td><td colspan="5">B</td><td colspan="10">0659 (0x293)</td><td>0</td></tr>
<tr><td>mftb</td><td colspan="2">31 (0x1F)</td><td colspan="5">D</td><td colspan="10">tbr</td><td colspan="10">0371 (0x173)</td><td>0</td></tr>
<tr><td>mfvscr[3]</td><td colspan="2">04 (0x04)</td><td colspan="5">vD</td><td colspan="5">00_000</td><td colspan="5">0_0000</td><td colspan="10">1540 (0x604)</td><td>0</td></tr>
<tr><td>mtcrf</td><td colspan="2">31 (0x1F)</td><td colspan="5">S</td><td>0</td><td colspan="8">CRM</td><td>0</td><td colspan="10">0144 (0x090)</td><td>0</td></tr>
<tr><td>mtfsb0x</td><td colspan="2">63 (0x3F)</td><td colspan="5">crbD</td><td colspan="5">00_000</td><td colspan="5">0_0000</td><td colspan="10">0070 (0x046)</td><td>Rc</td></tr>
<tr><td>mtfsb1x</td><td colspan="2">63 (0x3F)</td><td colspan="5">crbD</td><td colspan="5">00_000</td><td colspan="5">0_0000</td><td colspan="10">0038 (0x026)</td><td>Rc</td></tr>
<tr><td>mtfsfx</td><td colspan="2">63 (0x3F)</td><td>0</td><td colspan="8">FM</td><td>0</td><td colspan="5">B</td><td colspan="10">0711 (0x2C7)</td><td>Rc</td></tr>
<tr><td>mtfsfix</td><td colspan="2">63 (0x3F)</td><td colspan="3">crfD</td><td colspan="2">00</td><td colspan="5">00_000</td><td colspan="4">IMM</td><td>0</td><td colspan="10">0134 (0x086)</td><td>Rc</td></tr>
<tr><td>mtmsr[2]</td><td colspan="2">31 (0x1F)</td><td colspan="5">S</td><td colspan="5">00_000</td><td colspan="5">0_0000</td><td colspan="10">0146 (0x092)</td><td>0</td></tr>
<tr><td>mtspr[6]</td><td colspan="2">31 (0x1F)</td><td colspan="5">S</td><td colspan="10">spr</td><td colspan="10">0467 (0x1D3)</td><td>0</td></tr>
<tr><td>mtsr[2]</td><td colspan="2">31 (0x1F)</td><td colspan="5">S</td><td>0</td><td colspan="4">SR</td><td colspan="5">0_0000</td><td colspan="10">0210 (0x001)</td><td>0</td></tr>
<tr><td>mtsrin[2]</td><td colspan="2">31 (0x1F)</td><td colspan="5">S</td><td colspan="5">00_000</td><td colspan="5">B</td><td colspan="10">0242 (0x0F2)</td><td>0</td></tr>
<tr><td>mtvscr[3]</td><td colspan="2">04 (0x04)</td><td colspan="5">000_00</td><td colspan="5">00_000</td><td colspan="5">vB</td><td colspan="10">1604 (0x644)</td><td>0</td></tr>
<tr><td>mulhwx</td><td colspan="2">31(0x1F)</td><td colspan="5">D</td><td colspan="5">A</td><td colspan="5">B</td><td>0</td><td colspan="9">0075 (0x04B)</td><td>Rc</td></tr>
</table>

Table A-1. Instructions by Mnemonic (Dec, Hex) (continued)

<table>
<tr><th>Name</th><th>0 5</th><th>6 7 8 9 10</th><th>11 12 13 14 15</th><th>16 17 18 19 20</th><th>21</th><th>22 23 24 25</th><th>26 27 28 29</th><th>30</th><th>31</th></tr>
<tr><td>mulhwu<i>x</i></td><td>31 (0x1F)</td><td>D</td><td>A</td><td>B</td><td>0</td><td colspan="3">0011 (0x00B)</td><td>Rc</td></tr>
<tr><td>mulli</td><td>07 (0x07)</td><td>D</td><td>A</td><td colspan="6">SIMM</td></tr>
<tr><td>mullw<i>x</i></td><td>31 (0x1F)</td><td>D</td><td>A</td><td>B</td><td>OE</td><td colspan="3">0235 (0x0EB)</td><td>Rc</td></tr>
<tr><td>nand<i>x</i></td><td>31 (0x1F)</td><td>S</td><td>A</td><td>B</td><td colspan="4">0476 (0x1DC)</td><td>Rc</td></tr>
<tr><td>neg<i>x</i></td><td>31 (0x1F)</td><td>D</td><td>A</td><td>0_0000</td><td>OE</td><td colspan="3">0104 (0x068)</td><td>Rc</td></tr>
<tr><td>nor<i>x</i></td><td>31 (0x1F)</td><td>S</td><td>A</td><td>B</td><td colspan="4">0124 (0x07C)</td><td>Rc</td></tr>
<tr><td>or<i>x</i></td><td>31 (0x1F)</td><td>S</td><td>A</td><td>B</td><td colspan="4">0444 (0x1BC)</td><td>Rc</td></tr>
<tr><td>orc<i>x</i></td><td>31 (0x1F)</td><td>S</td><td>A</td><td>B</td><td colspan="4">0412 (0x19C)</td><td>Rc</td></tr>
<tr><td>ori</td><td>24 (0x18)</td><td>S</td><td>A</td><td colspan="6">UIMM</td></tr>
<tr><td>oris</td><td>25 (0x19)</td><td>S</td><td>A</td><td colspan="6">UIMM</td></tr>
<tr><td>rfi[2]</td><td>19 (0x13)</td><td>000_00</td><td>00_000</td><td>0_0000</td><td colspan="4">0050 (0x032)</td><td>0</td></tr>
<tr><td>rlwimi<i>x</i></td><td>20 (0x14)</td><td>S</td><td>A</td><td>SH</td><td colspan="2">MB</td><td colspan="2">ME</td><td>Rc</td></tr>
<tr><td>rlwinm<i>x</i></td><td>21 (0x15)</td><td>S</td><td>A</td><td>SH</td><td colspan="2">MB</td><td colspan="2">ME</td><td>Rc</td></tr>
<tr><td>rlwnm<i>x</i></td><td>23 (0x17)</td><td>S</td><td>A</td><td>B</td><td colspan="2">MB</td><td colspan="2">ME</td><td>Rc</td></tr>
<tr><td>sc</td><td>17 (0x11)</td><td colspan="6">000_0000_0000_0000_0000_0000_00</td><td>1</td><td>0</td></tr>
<tr><td>slw<i>x</i></td><td>31 (0x1F)</td><td>S</td><td>A</td><td>B</td><td colspan="4">0024 (0x018)</td><td>Rc</td></tr>
<tr><td>sraw<i>x</i></td><td>31 (0x1F)</td><td>S</td><td>A</td><td>B</td><td colspan="4">0792 (0x318)</td><td>Rc</td></tr>
<tr><td>srawi<i>x</i></td><td>31 (0x1F)</td><td>S</td><td>A</td><td>SH</td><td colspan="4">0824 (0x338)</td><td>Rc</td></tr>
<tr><td>srw<i>x</i></td><td>31 (0x1F)</td><td>S</td><td>A</td><td>B</td><td colspan="4">0536 (0x218)</td><td>Rc</td></tr>
<tr><td>stb</td><td>38 (0x26)</td><td>S</td><td>A</td><td colspan="6">d</td></tr>
<tr><td>stbu</td><td>39 (0x27)</td><td>S</td><td>A</td><td colspan="6">d</td></tr>
<tr><td>stbux</td><td>31 (0x1F)</td><td>S</td><td>A</td><td>B</td><td colspan="4">0247 (0x0F7)</td><td>0</td></tr>
<tr><td>stbx</td><td>31 (0x1F)</td><td>S</td><td>A</td><td>B</td><td colspan="4">0215 (0x0D7)</td><td>0</td></tr>
<tr><td>stfd</td><td>54 (0x36)</td><td>S</td><td>A</td><td colspan="6">d</td></tr>
<tr><td>stfdu</td><td>55 (0x37)</td><td>S</td><td>A</td><td colspan="6">d</td></tr>
<tr><td>stfdux</td><td>31 (0x1F)</td><td>S</td><td>A</td><td>B</td><td colspan="4">0759 (0x2F7)</td><td>0</td></tr>
<tr><td>stfdx</td><td>31 (0x1F)</td><td>S</td><td>A</td><td>B</td><td colspan="4">0727 (0x2D7)</td><td>0</td></tr>
<tr><td>stfiwx[1]</td><td>31 (0x1F)</td><td>S</td><td>A</td><td>B</td><td colspan="4">0983 (0x3D7)</td><td>0</td></tr>
<tr><td>stfs</td><td>52 (0x34)</td><td>S</td><td>A</td><td colspan="6">d</td></tr>
<tr><td>stfsu</td><td>53 (0x35)</td><td>S</td><td>A</td><td colspan="6">d</td></tr>
<tr><td>stfsux</td><td>31 (0x1F)</td><td>S</td><td>A</td><td>B</td><td colspan="4">0695 (0x2B7)</td><td>0</td></tr>
<tr><td>stfsx</td><td>31 (0x1F)</td><td>S</td><td>A</td><td>B</td><td colspan="4">0663 (0x297)</td><td>0</td></tr>
<tr><td>sth</td><td>44 (0x2C)</td><td>S</td><td>A</td><td colspan="6">d</td></tr>
</table>

Table A-1. Instructions by Mnemonic (Dec, Hex) (continued)

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
sthbrx	31 (0x1F)	S	A	B	0918 (0x396)		0
sthu	45 (0x2D)	S	A	d			
sthux	31 (0x1F)	S	A	B	0439 (0x1B7)		0
sthx	31 (0x1F)	S	A	B	0407 (0x197)		0
stmw[5]	47 (0x2F)	S	A	d			
stswi[5]	31 (0x1F)	S	A	NB	0725 (0x2D5)		0
stswx[5]	31 (0x1F)	S	A	B	0661 (0x295)		0
stvebx[3]	31 (0x1F)	**v**S	A	B	0135 (0x127)		0
stvehx[3]	31 (0x1F)	**v**S	A	B	0167 (0x0A7)		0
stvewx[3]	31 (0x1F)	**v**S	A	B	0199 (0x0C7)		0
stvx[3]	31 (0x1F)	**v**S	A	B	0231 (0x01F)		0
stvxl[3]	31 (0x1F)	**v**S	A	B	0487 (0x1E7)		0
stw	36 (0x24)	S	A	d			
stwbrx	31 (0x1F)	S	A	B	0662 (0x296)		0
stwcx.	31 (0x1F)	S	A	B	0150 (0x096)		1
stwu	37 (0x25)	S	A	d			
stwux	31 (0x1F)	S	A	B	0183 (0x0B7)		0
stwx	31 (0x1F)	S	A	B	0151 (0x097)		0
subf*x*	31 (0x1F)	D	A	B	OE	0040 (0x028)	Rc
subfc*x*	31 (0x1F)	D	A	B	OE	0008 (0x008)	Rc
subfe*x*	31 (0x1F)	D	A	B	OE	0136 (0x088)	Rc
subfic	08 (0x08)	D	A	SIMM			
subfme*x*	31 (0x1F)	D	A	0_0000	OE	0232 (0x0E8)	Rc
subfze*x*	31 (0x1F)	D	A	0_0000	OE	0200 (0x0C8)	Rc
sync	31 (0x1F)	000_00	00_000	0_0000	0598 (0x256)		0
tlbia[4]	31 (0x1F)	000_00	00_000	0_0000	0370 (0x172)		0
tlbie[1, 2]	31 (0x1F)	000_00	00_000	B	0306 (0x132)		0
tlbld[1, 2]	31 (0x1F)	000_00	00_000	B	0978 (0x3D2)		0
tlbli[1, 2]	31 (0x1F)	000_00	00_000	B	1010 (0x3F2)		0
tlbsync[1, 2]	31 (0x1F)	000_00	00_000	0_0000	0566 (0x236)		0
tw	31 (0x1F)	TO	A	B	0004 (0x004)		0
twi	03 (0x03)	TO	A	SIMM			
vaddcuw[3]	04 (0x04)	**v**D	**v**A	**v**B	084 (0x180)		0

Table A-1. Instructions by Mnemonic (Dec, Hex) (continued)

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
vaddfp^3	04 (0x04)	**v**D	**v**A	**v**B	0010 (0x0B4)		0
vaddsbs^3	04 (0x04)	**v**D	**v**A	**v**B	0768 (0x300)		0
vaddshs^3	04 (0x04)	**v**D	**v**A	**v**B	0832 (0x340)		0
vaddsws^3	04 (0x04)	**v**D	**v**A	**v**B	0896 (0x154)		0
vaddubm^3	04 (0x04)	**v**D	**v**A	**v**B	0000 (0x000)		0
vaddubs^3	04 (0x04)	**v**D	**v**A	**v**B	0512 (0x200)		0
vadduhm^3	04 (0x04)	**v**D	**v**A	**v**B	0064 (0x040)		0
vadduhs^3	04 (0x04)	**v**D	**v**A	**v**B	0576 (0x240)		0
vadduwm^3	04 (0x04)	**v**D	**v**A	**v**B	0128 (0x0F0)		0
vadduws^3	04 (0x04)	**v**D	**v**A	**v**B	0640 (0x280)		0
vand^3	04 (0x04)	**v**D	**v**A	**v**B	1028 (0x118)		0
vandc^3	04 (0x04)	**v**D	**v**A	**v**B	1092 (0x444)		0
vavgsb^3	04 (0x04)	**v**D	**v**A	**v**B	1282 (0x502)		0
vavgsh^3	04 (0x04)	**v**D	**v**A	**v**B	1346 (0x542)		0
vavgsw^3	04 (0x04)	**v**D	**v**A	**v**B	1410 (0x582)		0
vavgub^3	04 (0x04)	**v**D	**v**A	**v**B	1026 (0x402)		0
vavguh^3	04 (0x04)	**v**D	**v**A	**v**B	1090 (0x442)		0
vavguw^3	04 (0x04)	**v**D	**v**A	**v**B	1154 (0x482)		0
vcfsx^3	04 (0x04)	**v**D	UIMM	**v**B	0842 (0x1E2)		
vcfux^3	04 (0x04)	**v**D	UIMM	**v**B	0778 (0x30A)		0
vcmpbfp_x_^{3}	04 (0x04)	**v**D	**v**A	**v**B	Rc	0966 (0x3C6)	
vcmpeqfp_x_^{3}	04 (0x04)	**v**D	**v**A	**v**B	Rc	0198 (0x0C6)	
vcmpequb_x_^{3}	04 (0x04)	**v**D	**v**A	**v**B	Rc	0006 (0x006)	
vcmpequh_x_^{3}	04 (0x04)	**v**D	**v**A	**v**B	Rc	0070 (0x046)	
vcmpequw_x_^{3}	04 (0x04)	**v**D	**v**A	**v**B	Rc	0134 (0x086)	
vcmpgefp_x_^{3}	04 (0x04)	**v**D	**v**A	**v**B	Rc	0454 (0x1C6)	
vcmpgtfp_x_^{3}	04 (0x04)	**v**D	**v**A	**v**B	Rc	0710 (0x2C6)	
vcmpgtsb_x_^{3}	04 (0x04)	**v**D	**v**A	**v**B	Rc	0774 (0x306)	
vcmpgtsh_x_^{3}	04 (0x04)	**v**D	**v**A	**v**B	Rc	0838 (0x346)	
vcmpgtsw_x_^{3}	04 (0x04)	**v**D	**v**A	**v**B	Rc	0902 (0x386)	
vcmpgtub_x_^{3}	04 (0x04)	**v**D	**v**A	**v**B	Rc	0518 (0x206)	
vcmpgtuh_x_^{3}	04 (0x04)	**v**D	**v**A	**v**B	Rc	0582 (0x246)	
vcmpgtuw_x_^{3}	04 (0x04)	**v**D	**v**A	**v**B	Rc	0646 (0x286)	

Table A-1. Instructions by Mnemonic (Dec, Hex) (continued)

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
vctsxs[3]	04 (0x04)	**v**D	UIMM	**v**B	0970 (0x3CA)	
vctuxs[3]	04 (0x04)	**v**D	UIMM	**v**B	0906 (0x38A)	
vexptefp[3]	04 (0x04)	**v**D	00_000	**v**B	0394 (0x18A)	
vlogefp[3]	04 (0x04)	**v**D	00_000	**v**B	0458 (0x1CA)	
vmaddfp[3]	04 (0x04)	**v**D	**v**A	**v**B	**v**C	0046 (0x002E)
vmaxfp[3]	04 (0x04)	**v**D	**v**A	**v**B	1034 (0x040A)	
vmaxsb[3]	04 (0x04)	**v**D	**v**A	**v**B	0258 (0x028)	
vmaxsh[3]	04 (0x04)	**v**D	**v**A	**v**B	0322 (0x01C)	
vmaxsw[3]	04 (0x04)	**v**D	**v**A	**v**B	0386 (0x182)	
vmaxub[3]	04 (0x04)	**v**D	**v**A	**v**B	0002 (0x002)	
vmaxuh[3]	04 (0x04)	**v**D	**v**A	**v**B	0066 (0x042)	
vmaxuw[3]	04 (0x04)	**v**D	**v**A	**v**B	0130 (0x082)	
vmhaddshs[3]	04 (0x04)	**v**D	**v**A	**v**B	**v**C	0032 (0x020)
vmhraddshs[3]	04 (0x04)	**v**D	**v**A	**v**B	**v**C	0033 (0x021)
vminfp[3]	04 (0x04)	**v**D	**v**A	**v**B	1098 (0x44A)	
vminsb[3]	04 (0x04)	**v**D	**v**A	**v**B	0770 (0x302)	
vminsh[3]	04 (0x04)	**v**D	**v**A	**v**B	0834 (0x342)	
vminsw[3]	04 (0x04)	**v**D	**v**A	**v**B	0898 (0x382)	
vminub[3]	04 (0x04)	**v**D	**v**A	**v**B	0514 (0x202)	
vminuh[3]	04 (0x04)	**v**D	**v**A	**v**B	0578 (0x242)	
vminuw[3]	04 (0x04)	**v**D	**v**A	**v**B	0642 (0x282)	
vmladduhm[3]	04 (0x04)	**v**D	**v**A	**v**B	**v**C	0034 (0x022)
vmrghb[3]	04 (0x04)	**v**D	**v**A	**v**B	0012 (0x00C)	
vmrghh[3]	04 (0x04)	**v**D	**v**A	**v**B	0076 (0x04C)	
vmrghw[3]	04 (0x04)	**v**D	**v**A	**v**B	0140 (0x08C)	
vmrglb[3]	04 (0x04)	**v**D	**v**A	**v**B	0268 (0x008)	
vmrglh[3]	04 (0x04)	**v**D	**v**A	**v**B	0332 (0x14C)	
vmrglw[3]	04 (0x04)	**v**D	**v**A	**v**B	0396 (0x18C)	
vmsummbm[3]	04 (0x04)	**v**D	**v**A	**v**B	**v**C	0037 (0x025)
vmsumshm[3]	04 (0x04)	**v**D	**v**A	**v**B	**v**C	0040 (0x028)
vmsumshs[3]	04 (0x04)	**v**D	**v**A	**v**B	**v**C	0041 (0x029)
vmsumubm[3]	04 (0x04)	**v**D	**v**A	**v**B	**v**C	0036 (0x024)
vmsumuhm[3]	04 (0x04)	**v**D	**v**A	**v**B	**v**C	0038 (0x026)

Table A-1. Instructions by Mnemonic (Dec, Hex) (continued)

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
vmsumuhs[3]	04 (0x04)	vD	vA	vB	vC	0039 (0x027)
vmulesb[3]	04 (0x04)	vD	vA	vB	0776 (0x308)	
vmulesh[3]	04 (0x04)	vD	vA	vB	0840 (0x348)	
vmuleub[3]	04 (0x04)	vD	vA	vB	0520 (0x208)	
vmuleuh[3]	04 (0x04)	vD	vA	vB	0584 (0x248)	
vmulosb[3]	04 (0x04)	vD	vA	vB	0264 (0x108)	
vmulosh[3]	04 (0x04)	vD	vA	vB	0328 (0x148)	
vmuloub[3]	04 (0x04)	vD	vA	vB	0008 (0x008)	
vmulouh[3]	04 (0x04)	vD	vA	vB	0072 (0x048)	
vnmsubfp[3]	04 (0x04)	vD	vA	vB	vC	0047 (0x02F)
vnor[3]	04 (0x04)	vD	vA	vB	1284 (0x504)	
vor[3]	04 (0x04)	vD	vA	vB	1156 (0x484)	
vperm[3]	04 (0x04)	vD	vA	vB	vC	0043 (0x02B)
vpkpx[3]	04 (0x04)	vD	vA	vB	0782 (0x30E)	
vpkshss[3]	04 (0x04)	vD	vA	vB	0398 (0x18E)	
vpkshus[3]	04 (0x04)	vD	vA	vB	0270 (0x012)	
vpkswss[3]	04 (0x04)	vD	vA	vB	0462 (0x00C)	
vpkswus[3]	04 (0x04)	vD	vA	vB	0334 (0x14E)	
vpkuhum[3]	04 (0x04)	vD	vA	vB	0014 (0x00E)	
vpkuhus[3]	04 (0x04)	vD	vA	vB	0142 (0x08E)	
vpkuwum[3]	04 (0x04)	vD	vA	vB	0078 (0x04E)	
vpkuwus[3]	04 (0x04)	vD	vA	vB	0206 (0x0CE)	
vrefp[3]	04 (0x04)	vD	00_000	vB	0266 (0x10A)	
vrfim[3]	04 (0x04)	vD	00_000	vB	0714 (0x2CA)	
vrfin[3]	04 (0x04)	vD	00_000	vB	0522 (0x20A)	
vrfip[3]	04 (0x04)	vD	00_000	vB	0650 (0x28A)	
vrfiz[3]	04 (0x04)	vD	00_000	vB	0586 (0x24A)	
vrlb[3]	04 (0x04)	vD	vA	vB	0004 (0x004)	
vrlh[3]	04 (0x04)	vD	vA	vB	0068 (0x044)	
vrlw[3]	04 (0x04)	vD	vA	vB	0132 (0x084)	
vrsqrtefp[3]	04 (0x04)	vD	00_000	vB	0330 (0x14A)	
vsel[3]	04 (0x04)	vD	vA	vB	vC	0042 (0x02A)
vsl[3]	04 (0x04)	vD	vA	vB	0452 (0x1C4)	

Table A-1. Instructions by Mnemonic (Dec, Hex) (continued)

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
vslb[3]	04 (0x04)	**v**D	**v**A	**v**B	0260 (0x104)		
vsldoi[3]	04 (0x04)	**v**D	**v**A	**v**B	0	SH	0044 (0x02C)
vslh[3]	04 (0x04)	**v**D	**v**A	**v**B	0324 (0x144)		
vslo[3]	04 (0x04)	**v**D	**v**A	**v**B	1036 (0x40C)		
vslw[3]	04 (0x04)	**v**D	**v**A	**v**B	0388 (0x184)		
vspltb[3]	04 (0x04)	**v**D	UIMM	**v**B	0524 (0x20C)		
vsplth[3]	04 (0x04)	**v**D	UIMM	**v**B	0588 (0x24C)		
vspltisb[3]	04 (0x04)	**v**D	SIMM	0_0000	0780 (0x30C)		
vspltish[3]	04 (0x04)	**v**D	SIMM	0_0000	0844 (0x34C)		
vspltisw[3]	04 (0x04)	**v**D	SIMM	0_0000	0908 (0x38C)		
vspltw[3]	04 (0x04)	**v**D	UIMM	**v**B	0652 (0x28C)		
vsr[3]	04 (0x04)	**v**D	**v**A	**v**B	0708 (0x2C4)		
vsrab[3]	04 (0x04)	**v**D	**v**A	**v**B	0772 (0x304)		
vsrah[3]	04 (0x04)	**v**D	**v**A	**v**B	0836 (0x344)		
vsraw[3]	04 (0x04)	**v**D	**v**A	**v**B	0900 (0x384)		
vsrb[3]	04 (0x04)	**v**D	**v**A	**v**B	0516 (0x204)		
vsrh[3]	04 (0x04)	**v**D	**v**A	**v**B	0580 (0x244)		
vsro[3]	04 (0x04)	**v**D	**v**A	**v**B	1100 (0x44C)		
vsrw[3]	04 (0x04)	**v**D	**v**A	**v**B	0644 (0x284)		
vsubcuw[3]	04 (0x04)	**v**D	**v**A	**v**B	1408 (0x580)		
vsubfp[3]	04 (0x04)	**v**D	**v**A	**v**B	0074 (0x4A)		
vsubsbs[3]	04 (0x04)	**v**D	**v**A	**v**B	1792 (0x700)		
vsubshs[3]	04 (0x04)	**v**D	**v**A	**v**B	1856 (0x740)		
vsubsws[3]	04 (0x04)	**v**D	**v**A	**v**B	1920 (0x780)		
vsububm[3]	04 (0x04)	**v**D	**v**A	**v**B	1024 (0x400)		
vsububs[3]	04 (0x04)	**v**D	**v**A	**v**B	1536 (0x600)		
vsubuhm[3]	04 (0x04)	**v**D	**v**A	**v**B	1088 (0x440)		
vsubuhs[3]	04 (0x04)	**v**D	**v**A	**v**B	1600 (0x640)		
vsubuwm[3]	04 (0x04)	**v**D	**v**A	**v**B	1152 (0x480)		
vsubuws[3]	04 (0x04)	**v**D	**v**A	**v**B	1664 (0x680)		
vsumsws[3]	04 (0x04)	**v**D	**v**A	**v**B	1928 (0x788)		
vsum2sws[3]	04 (0x04)	**v**D	**v**A	**v**B	1672 (0x688)		
vsum4sbs[3]	04 (0x04)	**v**D	**v**A	**v**B	1800 (0x708)		

Table A-1. Instructions by Mnemonic (Dec, Hex) (continued)

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
vsum4shs[3]	04 (0x04)	**v**D	**v**A	**v**B	1608 (0x648)	
vsum4ubs[3]	04 (0x04)	**v**D	**v**A	**v**B	1544 (0x608)	
vupkhpx[3]	04 (0x04)	**v**D	00_000	**v**B	0846 (0x34E)	
vupkhsb[3]	04 (0x04)	**v**D	00_000	**v**B	0526 (0x20E)	
vupkhsh[3]	04 (0x04)	**v**D	00_000	**v**B	0590 (0x24E)	
vupklpx[3]	04 (0x04)	**v**D	00_000	**v**B	0974 (0x3CE)	
vupklsb[3]	04 (0x04)	**v**D	00_000	**v**B	0654 (0x28E)	
vupklsh[3]	04 (0x04)	**v**D	00_000	**v**B	0718 (0x2CE)	
vxor[3]	04 (0x04)	**v**D	**v**A	**v**B	1220 (0x4C4)	
xor*x*	31 (0x1F)	S	A	B	0316 (0x13C)	Rc
xori	26 (0x1A)	S	A	UIMM		
xoris	27 (0x1B)	S	A	UIMM		

[1] Optional to the PowerPC architecture but implemented by the MPC7450.

[2] Supervisor-level instruction.

[3] AltiVec technology-specific instruction.

[4] Optional instruction not implemented by the MPC7450.

[5] Load/store string/multiple instruction.

[6] Supervisor- and user-level instruction.

A.2 Instructions Sorted by Primary and Secondary Opcodes (Decimal and Hexadecimal)

Table A-2 shows the instructions implemented in the MPC7450. The instructions are listed by their primary (0–5) and secondary (21–31) opcodes in decimal and hexadecimal format.

Key:

▨ Reserved bits

Table A-2. Instructions by Primary and Secondary Opcodes (Dec, Hex)

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
twi	03 (0x03)	TO	A	SIMM				
vaddubm[1]	04 (0x04)	vD	vA	vB	0000 (0x000)			0
vmaxub[1]	04 (0x04)	vD	vA	vB	0002 (0x002)			
vrlb[1]	04 (0x04)	vD	vA	vB	0004 (0x004)			
vcmpequb*x*[1]	04 (0x04)	vD	vA	vB	Rc	0006 (0x006)		
vmuloub[1]	04 (0x04)	vD	vA	vB	0008 (0x008)			
vaddfp[1]	04 (0x04)	vD	vA	vB	0010 (0x0B4)			0
vmrghb[1]	04 (0x04)	vD	vA	vB	0012 (0x00C)			
vpkuhum[1]	04 (0x04)	vD	vA	vB	0014 (0x00E)			
vmhaddshs[1]	04 (0x04)	vD	vA	vB	vC		0032 (0x020)	
vmhraddshs[1]	04 (0x04)	vD	vA	vB	vC		0033 (0x021)	
vmladduhm[1]	04 (0x04)	vD	vA	vB	vC		0034 (0x022)	
vmsumubm[1]	04 (0x04)	vD	vA	vB	vC		0036 (0x024)	
vmsummbm[1]	04 (0x04)	vD	vA	vB	vC		0037 (0x025)	
vmsumuhm[1]	04 (0x04)	vD	vA	vB	vC		0038 (0x026)	
vmsumuhs[1]	04 (0x04)	vD	vA	vB	vC		0039 (0x027)	
vmsumshm[1]	04 (0x04)	vD	vA	vB	vC		0040 (0x028)	
vmsumshs[1]	04 (0x04)	vD	vA	vB	vC		0041 (0x029)	
vsel[1]	04 (0x04)	vD	vA	vB	vC		0042 (0x02A)	
vperm[1]	04 (0x04)	vD	vA	vB	vC		0043 (0x02B)	
vsldoi[1]	04 (0x04)	vD	vA	vB	0	SH	0044 (0x02C)	
vmaddfp[1]	04 (0x04)	vD	vA	vB	vC		0046 (0x002E)	
vnmsubfp[1]	04 (0x04)	vD	vA	vB	vC		0047 (0x02F)	
vadduhm[1]	04 (0x04)	vD	vA	vB	0064 (0x040)			0
vmaxuh[1]	04 (0x04)	vD	vA	vB	0066 (0x042)			
vrlh[1]	04 (0x04)	vD	vA	vB	0068 (0x044)			

Table A-2. Instructions by Primary and Secondary Opcodes (Dec, Hex) (continued)

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
vcmpequh*x*[1]	04 (0x04)	**v**D	**v**A	**v**B	Rc	0070 (0x046)	
vmulouh[1]	04 (0x04)	**v**D	**v**A	**v**B	0072 (0x048)		
vsubfp[1]	04 (0x04)	**v**D	**v**A	**v**B	0074 (0x4A)		
vmrghh[1]	04 (0x04)	**v**D	**v**A	**v**B	0076 (0x04C)		
vpkuwum[1]	04 (0x04)	**v**D	**v**A	**v**B	0078 (0x04E)		
vadduwm[1]	04 (0x04)	**v**D	**v**A	**v**B	0128 (0x0F0)		0
vmaxuw[1]	04 (0x04)	**v**D	**v**A	**v**B	0130 (0x082)		
vrlw[1]	04 (0x04)	**v**D	**v**A	**v**B	0132 (0x084)		
vcmpequw*x*[1]	04 (0x04)	**v**D	**v**A	**v**B	Rc	0134 (0x086)	
vmrghw[1]	04 (0x04)	**v**D	**v**A	**v**B	0140 (0x08C)		
vpkuhus[1]	04 (0x04)	**v**D	**v**A	**v**B	0142 (0x08E)		
vcmpeqfp*x*[1]	04 (0x04)	**v**D	**v**A	**v**B	Rc	0198 (0x0C6)	
vpkuwus[1]	04 (0x04)	**v**D	**v**A	**v**B	0206 (0x0CE)		
vmaxsb[1]	04 (0x04)	**v**D	**v**A	**v**B	0258 (0x028)		
vslb[1]	04 (0x04)	**v**D	**v**A	**v**B	0260 (0x104)		
vmulosb[1]	04 (0x04)	**v**D	**v**A	**v**B	0264 (0x108)		
vrefp[1]	04 (0x04)	**v**D	00_000	**v**B	0266 (0x10A)		
vmrglb[1]	04 (0x04)	**v**D	**v**A	**v**B	0268 (0x008)		
vpkshus[1]	04 (0x04)	**v**D	**v**A	**v**B	0270 (0x012)		
vmaxsh[1]	04 (0x04)	**v**D	**v**A	**v**B	0322 (0x01C)		
vslh[1]	04 (0x04)	**v**D	**v**A	**v**B	0324 (0x144)		
vmulosh[1]	04 (0x04)	**v**D	**v**A	**v**B	0328 (0x148)		
vrsqrtefp[1]	04 (0x04)	**v**D	00_000	**v**B	0330 (0x14A)		
vmrglh[1]	04 (0x04)	**v**D	**v**A	**v**B	0332 (0x14C)		
vpkswus[1]	04 (0x04)	**v**D	**v**A	**v**B	0334 (0x14E)		
vaddcuw[1]	04 (0x04)	**v**D	**v**A	**v**B	0384 (0x180)		0
vmaxsw[1]	04 (0x04)	**v**D	**v**A	**v**B	0386 (0x182)		
vslw[1]	04 (0x04)	**v**D	**v**A	**v**B	0388 (0x184)		
vexptefp[1]	04 (0x04)	**v**D	00_000	**v**B	0394 (0x18A)		
vmrglw[1]	04 (0x04)	**v**D	**v**A	**v**B	0396 (0x18C)		
vpkshss[1]	04 (0x04)	**v**D	**v**A	**v**B	0398 (0x18E)		
vsl[1]	04 (0x04)	**v**D	**v**A	**v**B	0452 (0x1C4)		
vcmpgefp*x*[1]	04 (0x04)	**v**D	**v**A	**v**B	Rc	0454 (0x1C6)	

Table A-2. Instructions by Primary and Secondary Opcodes (Dec, Hex) (continued)

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
vlogefp[1]	04 (0x04)	vD	00_000	vB	0458 (0x1CA)		
vpkswss[1]	04 (0x04)	vD	vA	vB	0462 (0x00C)		
vaddubs[1]	04 (0x04)	vD	vA	vB	0512 (0x200)		0
vminub[1]	04 (0x04)	vD	vA	vB	0514 (0x202)		
vsrb[1]	04 (0x04)	vD	vA	vB	0516 (0x204)		
vcmpgtub*x*[1]	04 (0x04)	vD	vA	vB	Rc	0518 (0x206)	
vmuleub[1]	04 (0x04)	vD	vA	vB	0520 (0x208)		
vrfin[1]	04 (0x04)	vD	00_000	vB	0522 (0x20A)		
vspltb[1]	04 (0x04)	vD	UIMM	vB	0524 (0x20C)		
vupkhsb[1]	04 (0x04)	vD	00_000	vB	0526 (0x20E)		
vadduhs[1]	04 (0x04)	vD	vA	vB	0576 (0x240)		0
vminuh[1]	04 (0x04)	vD	vA	vB	0578 (0x242)		
vsrh[1]	04 (0x04)	vD	vA	vB	0580 (0x244)		
vcmpgtuh*x*[1]	04 (0x04)	vD	vA	vB	Rc	0582 (0x246)	
vmuleuh[1]	04 (0x04)	vD	vA	vB	0584 (0x248)		
vrfiz[1]	04 (0x04)	vD	00_000	vB	0586 (0x24A)		
vsplth[1]	04 (0x04)	vD	UIMM	vB	0588 (0x24C)		
vupkhsh[1]	04 (0x04)	vD	00_000	vB	0590 (0x24E)		
vadduws[1]	04 (0x04)	vD	vA	vB	0640 (0x280)		0
vminuw[1]	04 (0x04)	vD	vA	vB	0642 (0x282)		
vsrw[1]	04 (0x04)	vD	vA	vB	0644 (0x284)		
vcmpgtuw*x*[1]	04 (0x04)	vD	vA	vB	Rc	0646 (0x286)	
vrfip[1]	04 (0x04)	vD	00_000	vB	0650 (0x28A)		
vspltw[1]	04 (0x04)	vD	UIMM	vB	0652 (0x28C)		
vupklsb[1]	04 (0x04)	vD	00_000	vB	0654 (0x28E)		
vsr[1]	04 (0x04)	vD	vA	vB	0708 (0x2C4)		
vcmpgtfp*x*[1]	04 (0x04)	vD	vA	vB	Rc	0710 (0x2C6)	
vrfim[1]	04 (0x04)	vD	00_000	vB	0714 (0x2CA)		
vupklsh[1]	04 (0x04)	vD	00_000	vB	0718 (0x2CE)		
vaddsbs[1]	04 (0x04)	vD	vA	vB	0768 (0x300)		0
vminsb[1]	04 (0x04)	vD	vA	vB	0770 (0x302)		
vsrab[1]	04 (0x04)	vD	vA	vB	0772 (0x304)		
vcmpgtsb*x*[1]	04 (0x04)	vD	vA	vB	Rc	0774 (0x306)	

Table A-2. Instructions by Primary and Secondary Opcodes (Dec, Hex) (continued)

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
vmulesb[1]	04 (0x04)	**v**D	**v**A	**v**B	0776 (0x308)		
vcfux[1]	04 (0x04)	**v**D	UIMM	**v**B	0778 (0x30A)		0
vspltisb[1]	04 (0x04)	**v**D	SIMM	0_0000	0780 (0x30C)		
vpkpx[1]	04 (0x04)	**v**D	**v**A	**v**B	0782 (0x30E)		
vaddshs[1]	04 (0x04)	**v**D	**v**A	**v**B	0832 (0x340)		0
vminsh[1]	04 (0x04)	**v**D	**v**A	**v**B	0834 (0x342)		
vsrah[1]	04 (0x04)	**v**D	**v**A	**v**B	0836 (0x344)		
vcmpgtsh*x*[1]	04 (0x04)	**v**D	**v**A	**v**B	Rc	0838 (0x346)	
vmulesh[1]	04 (0x04)	**v**D	**v**A	**v**B	0840 (0x348)		
vcfsx[1]	04 (0x04)	**v**D	UIMM	**v**B	0842 (0x1E2)		
vspltish[1]	04 (0x04)	**v**D	SIMM	0_0000	0844 (0x34C)		
vupkhpx[1]	04 (0x04)	**v**D	00_000	**v**B	0846 (0x34E)		
vaddsws[1]	04 (0x04)	**v**D	**v**A	**v**B	0896 (0x154)		0
vminsw[1]	04 (0x04)	**v**D	**v**A	**v**B	0898 (0x382)		
vsraw[1]	04 (0x04)	**v**D	**v**A	**v**B	0900 (0x384)		
vcmpgtsw*x*[1]	04 (0x04)	**v**D	**v**A	**v**B	Rc	0902 (0x386)	
vctuxs[1]	04 (0x04)	**v**D	UIMM	**v**B	0906 (0x38A)		
vspltisw[1]	04 (0x04)	**v**D	SIMM	0_0000	0908 (0x38C)		
vcmpbfp*x*[1]	04 (0x04)	**v**D	**v**A	**v**B	Rc	0966 (0x3C6)	
vctsxs[1]	04 (0x04)	**v**D	UIMM	**v**B	0970 (0x3CA)		
vupklpx[1]	04 (0x04)	**v**D	00_000	**v**B	0974 (0x3CE)		
vsububm[1]	04 (0x04)	**v**D	**v**A	**v**B	1024 (0x400)		
vavgub[1]	04 (0x04)	**v**D	**v**A	**v**B	1026 (0x402)		0
vand[1]	04 (0x04)	**v**D	**v**A	**v**B	1028 (0x118)		0
vmaxfp[1]	04 (0x04)	**v**D	**v**A	**v**B	1034 (0x040A)		
vslo[1]	04 (0x04)	**v**D	**v**A	**v**B	1036 (0x40C)		
vsubuhm[1]	04 (0x04)	**v**D	**v**A	**v**B	1088 (0x440)		
vavguh[1]	04 (0x04)	**v**D	**v**A	**v**B	1090 (0x442)		0
vandc[1]	04 (0x04)	**v**D	**v**A	**v**B	1092 (0x444)		0
vminfp[1]	04 (0x04)	**v**D	**v**A	**v**B	1098 (0x44A)		
vsro[1]	04 (0x04)	**v**D	**v**A	**v**B	1100 (0x44C)		
vsubuwm[1]	04 (0x04)	**v**D	**v**A	**v**B	1152 (0x480)		
vavguw[1]	04 (0x04)	**v**D	**v**A	**v**B	1154 (0x482)		0

Table A-2. Instructions by Primary and Secondary Opcodes (Dec, Hex) (continued)

Name	0–5	6–8	9	10	11–13	14–15	16–20	21–29	30	31
vor[1]	04 (0x04)	vD			vA		vB	1156 (0x484)		
vxor[1]	04 (0x04)	vD			vA		vB	1220 (0x4C4)		
vavgsb[1]	04 (0x04)	vD			vA		vB	1282 (0x502)		0
vnor[1]	04 (0x04)	vD			vA		vB	1284 (0x504)		
vavgsh[1]	04 (0x04)	vD			vA		vB	1346 (0x542)		0
vsubcuw[1]	04 (0x04)	vD			vA		vB	1408 (0x580)		
vavgsw[1]	04 (0x04)	vD			vA		vB	1410 (0x582)		0
vsububs[1]	04 (0x04)	vD			vA		vB	1536 (0x600)		
mfvscr[1]	04 (0x04)	vD			00_000		0_0000	1540 (0x604)		0
vsum4ubs[1]	04 (0x04)	vD			vA		vB	1544 (0x608)		
vsubuhs[1]	04 (0x04)	vD			vA		vB	1600 (0x640)		
mtvscr[1]	04 (0x04)	000_00			00_000		vB	1604 (0x644)		0
vsum4shs[1]	04 (0x04)	vD			vA		vB	1608 (0x648)		
vsubuws[1]	04 (0x04)	vD			vA		vB	1664 (0x680)		
vsum2sws[1]	04 (0x04)	vD			vA		vB	1672 (0x688)		
vsubsbs[1]	04 (0x04)	vD			vA		vB	1792 (0x700)		
vsum4sbs[1]	04 (0x04)	vD			vA		vB	1800 (0x708)		
vsubshs[1]	04 (0x04)	vD			vA		vB	1856 (0x740)		
vsubsws[1]	04 (0x04)	vD			vA		vB	1920 (0x780)		
vsumsws[1]	04 (0x04)	vD			vA		vB	1928 (0x788)		
mulli	07 (0x07)	D			A		SIMM			
subfic	08 (0x08)	D			A		SIMM			
cmpli	10 (0x0A)	crfD	0	L	A		UIMM			
cmpi	11 (0x0B)	crfD	0	L	A		SIMM			
addic	12 (0xC)	D			A		SIMM			
addic.	13 (0xD)	D			A		SIMM			
addi	14 (0xE)	D			A		SIMM			
addis	15 (0xF)	D			A		SIMM			
bc*x*	16 (0x10)	BO			BI		BD		AA	LK
sc	17 (0x11)	000_0000_0000_0000_0000_0000_00							1	0
b*x*	18 (0x12)	LI							AA	LK
mcrf	19 (0x13)	crfD	00		crfS	00	0_0000	0000 (0x000)		0
bclr*x*	19 (0x13)	BO			BI		0_0000	0016 (0x010)		LK

Table A-2. Instructions by Primary and Secondary Opcodes (Dec, Hex) (continued)

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
crnor	19 (0x13)	crbD	crbA	crbB	0033 (0x21)		0
rfi[2]	19 (0x13)	000_00	00_000	0_0000	0050 (0x032)		0
crandc	19 (0x13)	crbD	crbA	crbB	0129 (0x081)		0
isync	19 (0x13)	000_00	00_000	0_0000	0150 (0x096)		0
crxor	19 (0x13)	crbD	crbA	crbB	0193 (0C1)		0
crnand	19 (0x13)	crbD	crbA	crbB	0225 (0x0E1)		0
crand	19 (0x13)	crbD	crbA	crbB	0257 (0x101)		0
creqv	19 (0x13)	crbD	crbA	crbB	0289 (0x121)		0
crorc	19 (0x13)	crbD	crbA	crbB	0417 (0x1A1)		0
cror	19 (0x13)	crbD	crbA	crbB	0449 (0x1C1)		0
bcctr_x_	19 (0x13)	BO	BI	0_0000	0528 (0x210)		LK
rlwimi_x_	20 (0x14)	S	A	SH	MB	ME	Rc
rlwinm_x_	21 (0x15)	S	A	SH	MB	ME	Rc
rlwnm_x_	23 (0x17)	S	A	B	MB	ME	Rc
ori	24 (0x18)	S	A	UIMM			
oris	25 (0x19)	S	A	UIMM			
xori	26 (0x1A)	S	A	UIMM			
xoris	27 (0x1B)	S	A	UIMM			
andi.	28 (0x1C)	S	A	UIMM			
andis.	29 (0x1D)	S	A	UIMM			
cmp	31 (0x1F)	crfD 0 L	A	B	0000 (0x000)		0
tw	31 (0x1F)	TO	A	B	0004 (0x004)		0
lvsl[1]	31 (0x1F)	**v**D	A	B	0006 (0x006)		0
lvebx[1]	31 (0x1F)	**v**D	A	B	0007 (0x007)		0
subfc_x_	31 (0x1F)	D	A	B	OE	0008 (0x008)	Rc
addc_x_	31 (0x1F)	D	A	B	OE	0010 (0x00A)	Rc
mulhwu_x_	31 (0x1F)	D	A	B	0	0011 (0x00B)	Rc
mfcr	31 (0x1F)	D	00_000	0_0000	0019 (0x013)		0
lwarx	31 (0x1F)	D	A	B	0020 (0x014)		0
lwzx	31 (0x1F)	D	A	B	0023 (0x017)		0
slw_x_	31 (0x1F)	S	A	B	0024 (0x018)		Rc
cntlzw_x_	31 (0x1F)	S	A	0_0000	0026 (0x01A)		Rc
and_x_	31 (0x1F)	S	A	B	0028 (0x01C)		Rc

Table A-2. Instructions by Primary and Secondary Opcodes (Dec, Hex) (continued)

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
cmpl	31 (0x1F)	crfD 0 L	A	B	0032 (0x020)	0
lvsr[1]	31 (0x1F)	**v**D	A	B	0038 (0x026)	0
lvehx[1]	31 (0x1F)	**v**D	A	B	0039 (0x027)	0
subf*x*	31 (0x1F)	D	A	B	OE 0040 (0x028)	Rc
dcbst	31 (0x1F)	000_00	A	B	0054 (0x036)	0
lwzux	31 (0x1F)	D	A	B	0055 (0x037)	0
andc*x*	31 (0x1F)	S	A	B	0060 (0x03C)	Rc
lvewx[1]	31 (0x1F)	**v**D	A	B	0071 (0x047)	0
mulhw*x*	31(0x1F)	D	A	B	0 0075 (0x04B)	Rc
mfmsr[2]	31 (0x1F)	D	00_000	0_0000	0083 (0x053)	0
dcbf	31 (0x1F)	000_00	A	B	0086 (0x056)	0
lbzx	31 (0x1F)	D	A	B	0087 (0x057)	0
lvx[1]	31 (0x1F)	**v**D	A	B	0103 (0x067)	0
neg*x*	31 (0x1F)	D	A	0_0000	OE 0104 (0x068)	Rc
lbzux	31 (0x1F)	D	A	B	0119 (0x077)	0
nor*x*	31 (0x1F)	S	A	B	0124 (0x07C)	Rc
stvebx[1]	31 (0x1F)	**v**S	A	B	0135 (0x127)	0
subfe*x*	31 (0x1F)	D	A	B	OE 0136 (0x088)	Rc
adde*x*	31 (0x1F)	D	A	B	OE 0138 (0x08A)	Rc
mtcrf	31 (0x1F)	S	0 CRM	0	0144 (0x090)	0
mtmsr[2]	31 (0x1F)	S	00_000	0_0000	0146 (0x092)	0
stwcx.	31 (0x1F)	S	A	B	0150 (0x096)	1
stwx	31 (0x1F)	S	A	B	0151 (0x097)	0
stvehx[1]	31 (0x1F)	**v**S	A	B	0167 (0x0A7)	0
stwux	31 (0x1F)	S	A	B	0183 (0x0B7)	0
stvewx[1]	31 (0x1F)	**v**S	A	B	0199 (0x0C7)	0
subfze*x*	31 (0x1F)	D	A	0_0000	OE 0200 (0x0C8)	Rc
addze*x*	31 (0x1F)	D	A	0_0000	OE 0202 (0x0CA)	Rc
mtsr[2]	31 (0x1F)	S	0 SR	0_0000	0210 (0x001)	0
stbx	31 (0x1F)	S	A	B	0215 (0x0D7)	0
stvx[1]	31 (0x1F)	**v**S	A	B	0231 (0x01F)	0
subfme*x*	31 (0x1F)	D	A	0_0000	OE 0232 (0x0E8)	Rc
addme*x*	31 (0x1F)	D	A	0_0000	OE 0234 (0x0EA)	Rc

Table A-2. Instructions by Primary and Secondary Opcodes (Dec, Hex) (continued)

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
mullw*x*	31 (0x1F)	D			A	B	OE	0235 (0x0EB)	Rc
mtsrin[2]	31 (0x1F)	S			00_000	B	0242 (0x0F2)		0
dcbtst	31 (0x1F)	000_00			A	B	0246 (0x0F6)		0
stbux	31 (0x1F)	S			A	B	0247 (0x0F7)		0
add*x*	31 (0x1F)	D			A	B	OE	0266 (0x10A)	Rc
dcbt	31 (0x1F)	000_00			A	B	0278 (0x116)		0
lhzx	31 (0x1F)	D			A	B	0279 (0x117)		0
eqv*x*	31 (0x1F)	S			A	B	0284 (0x11C)		Rc
tlbie[2, 3]	31 (0x1F)	000_00			00_000	B	0306 (0x132)		0
eciwx[3]	31 (0x1F)	D			A	B	0310 (0x136)		0
lhzux	31 (0x1F)	D			A	B	0311 (0x137)		0
xor*x*	31 (0x1F)	S			A	B	0316 (0x13C)		Rc
mfspr[4]	31 (0x1F)	D			spr		0339 (0x153)		0
dst[1]	31 (0x1F)	0	00	STRM	A	B	0342 (0x156)		0
dstt[1]	31 (0x1F)	1	00	STRM	A	B	0342 (0x156)		0
lhax	31 (0x1F)	D			A	B	0343 (0x157)		0
lvxl[1]	31 (0x1F)	**v**D			A	B	0359 (0x167)		0
tlbia[5]	31 (0x1F)	000_00			00_000	0_0000	0370 (0x172)		0
mftb	31 (0x1F)	D			tbr		0371 (0x173)		0
dstst[1]	31 (0x1F)	0	00	STRM	A	B	0374 (0x176)		0
dststt[1]	31 (0x1F)	1	00	STRM	A	B	0374 (0x176)		0
lhaux	31 (0x1F)	D			A	B	0375 (0x177)		0
sthx	31 (0x1F)	S			A	B	0407 (0x197)		0
orc*x*	31 (0x1F)	S			A	B	0412 (0x19C)		Rc
ecowx[3]	31 (0x1F)	S			A	B	0438 (0x1B6)		0
sthux	31 (0x1F)	S			A	B	0439 (0x1B7)		0
or*x*	31 (0x1F)	S			A	B	0444 (0x1BC)		Rc
divwu*x*	31 (0x1F)	D			A	B	OE	0459 (0x1CB)	Rc
mtspr[4]	31 (0x1F)	S			spr		0467 (0x1D3)		0
dcbi[2]	31 (0x1F)	000_00			A	B	0470 (0x1D6)		0
nand*x*	31 (0x1F)	S			A	B	0476 (0x1DC)		Rc
stvxl[1]	31 (0x1F)	**v**S			A	B	0487 (0x1E7)		0
divw*x*	31 (0x1F)	D			A	B	OE	0491 (0x1EB)	Rc

Table A-2. Instructions by Primary and Secondary Opcodes (Dec, Hex) (continued)

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 (0x1F)	crfD 00	00_000	0_0000	0512 (0x200)	0
lswx[6]	31 (0x1F)	D	A	B	0533 (0x215)	0
lwbrx	31 (0x1F)	D	A	B	0534 (0x216)	0
lfsx	31 (0x1F)	D	A	B	0535 (0x217)	0
srw*x*	31 (0x1F)	S	A	B	0536 (0x218)	Rc
tlbsync[2, 3]	31 (0x1F)	000_00	00_000	0_0000	0566 (0x236)	0
lfsux	31 (0x1F)	D	A	B	0567 (0x237)	0
mfsr[2]	31 (0x1F)	D	0 SR	0_0000	0595 (0x099)	0
lswi[6]	31 (0x1F)	D	A	NB	0597 (0x255)	0
sync	31 (0x1F)	000_00	00_000	0_0000	0598 (0x256)	0
lfdx	31 (0x1F)	D	A	B	0599 (0x257)	0
lfdux	31 (0x1F)	D	A	B	0631 (0x277)	0
mfsrin [2]	31 (0x1F)	D	00_000	B	0659 (0x293)	0
stswx[6]	31 (0x1F)	S	A	B	0661 (0x295)	0
stwbrx	31 (0x1F)	S	A	B	0662 (0x296)	0
stfsx	31 (0x1F)	S	A	B	0663 (0x297)	0
stfsux	31 (0x1F)	S	A	B	0695 (0x2B7)	0
stswi[6]	31 (0x1F)	S	A	NB	0725 (0x2D5)	0
stfdx	31 (0x1F)	S	A	B	0727 (0x2D7)	0
dcba[3]	31 (0x1F)	000_00	A	B	0758 (0x2F6)	0
stfdux	31 (0x1F)	S	A	B	0759 (0x2F7)	0
lhbrx	31 (0x1F)	D	A	B	0790 (0x316)	0
sraw*x*	31 (0x1F)	S	A	B	0792 (0x318)	Rc
dss[1]	31 (0x1F)	0 00 STRM	00_000	0_0000	0822 (0x336)	0
dssall[1]	31 (0x1F)	1 00 STRM	00_000	0_0000	0822 (0x336)	0
srawi*x*	31 (0x1F)	S	A	SH	0824 (0x338)	Rc
eieio	31 (0x1F)	000_00	00_000	0_0000	0854 (0x356)	0
sthbrx	31 (0x1F)	S	A	B	0918 (0x396)	0
extsh*x*	31 (0x1F)	S	A	0_0000	0922 (0x39A)	Rc
extsb*x*	31 (0x1F)	S	A	0_0000	0954 (0x3BA)	Rc
tlbld[2, 3]	31 (0x1F)	000_00	00_000	B	0978 (0x3D2)	0
icbi	31 (0x1F)	000_00	A	B	0982 (0x3D6)	0
stfiwx[3]	31 (0x1F)	S	A	B	0983 (0x3D7)	0

Table A-2. Instructions by Primary and Secondary Opcodes (Dec, Hex) (continued)

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
tlbli[2, 3]	31 (0x1F)	000_00	00_000	B	1010 (0x3F2)		0
dcbz	31 (0x1F)	000_00	A	B	1014 (0x3F6)		0
lwz	32 (0x20)	D	A	d			
lwzu	33 (0x21)	D	A	d			
lbz	34 (0x22)	D	A	d			
lbzu	35 (0x23)	D	A	d			
stw	36 (0x24)	S	A	d			
stwu	37 (0x25)	S	A	d			
stb	38 (0x26)	S	A	d			
stbu	39 (0x27)	S	A	d			
lhz	40 (0x28)	D	A	d			
lhzu	41 (0x29)	D	A	d			
lha	42 (0x2A)	D	A	d			
lhau	43 (0x2B)	D	A	d			
sth	44 (0x2C)	S	A	d			
sthu	45 (0x2D)	S	A	d			
lmw[6]	46 (0x2E)	D	A	d			
stmw[6]	47 (0x2F)	S	A	d			
lfs	48 (0x30)	D	A	d			
lfsu	49 (0x31)	D	A	d			
lfd	50 (0x32)	D	A	d			
lfdu	51 (0x33)	D	A	d			
stfs	52 (0x34)	S	A	d			
stfsu	53 (0x35)	S	A	d			
stfd	54 (0x36)	S	A	d			
stfdu	55 (0x37)	S	A	d			
fdivs*x*	59 (0x3B)	D	A	B	0000_0	0018 (0x012)	Rc
fsubs*x*	59 (0x3B)	D	A	B	0000_0	0020 (0x014)	Rc
fadds*x*	59 (0x3B)	D	A	B	0000_0	0021 (0x015)	Rc
fsqrts*x*[5]	59 (0x3B)	D	00_000	B	0000_0	0022 (0x016)	Rc
fres*x*[3]	59 (0x3B)	D	00_000	B	0000_0	0024 (0x018)	Rc
fmuls*x*	59 (0x3B)	D	A	0_0000	C	0025 (0x019)	Rc
fmsubs*x*	59 (0x3B)	D	A	B	C	0028 (0x01C)	Rc

Table A-2. Instructions by Primary and Secondary Opcodes (Dec, Hex) (continued)

<table>
<tr><th>Name</th><th>0 5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th><th>17</th><th>18</th><th>19</th><th>20</th><th>21</th><th>22</th><th>23</th><th>24</th><th>25</th><th>26</th><th>27</th><th>28</th><th>29</th><th>30</th><th>31</th></tr>
<tr><td>fmadds<i>x</i></td><td>59 (0x3B)</td><td colspan="5">D</td><td colspan="5">A</td><td colspan="5">B</td><td colspan="5">C</td><td colspan="5">0029 (0x01D)</td><td>Rc</td></tr>
<tr><td>fnmsubs<i>x</i></td><td>59 (0x3B)</td><td colspan="5">D</td><td colspan="5">A</td><td colspan="5">B</td><td colspan="5">C</td><td colspan="5">0030 (0x01E)</td><td>Rc</td></tr>
<tr><td>fnmadds<i>x</i></td><td>59 (0x3B)</td><td colspan="5">D</td><td colspan="5">A</td><td colspan="5">B</td><td colspan="5">C</td><td colspan="5">0031 (0x01F)</td><td>Rc</td></tr>
<tr><td>fcmpu</td><td>63 (0x3F)</td><td colspan="3">crfD</td><td colspan="2">00</td><td colspan="5">A</td><td colspan="5">B</td><td colspan="10">0000 (0x000)</td><td>0</td></tr>
<tr><td>frsp<i>x</i></td><td>63 (0x3F)</td><td colspan="5">D</td><td colspan="5">00_000</td><td colspan="5">B</td><td colspan="10">0012 (0xC)</td><td>Rc</td></tr>
<tr><td>fctiw<i>x</i></td><td>63 (0x3F)</td><td colspan="5">D</td><td colspan="5">00_000</td><td colspan="5">B</td><td colspan="10">0014 (0x00E)</td><td>Rc</td></tr>
<tr><td>fctiwz<i>x</i></td><td>63 (0x3F)</td><td colspan="5">D</td><td colspan="5">00_000</td><td colspan="5">B</td><td colspan="10">0015 (0x00F)</td><td>Rc</td></tr>
<tr><td>fdiv<i>x</i></td><td>63 (0x3F)</td><td colspan="5">D</td><td colspan="5">A</td><td colspan="5">B</td><td colspan="5">0000_0</td><td colspan="5">0018 (0x012)</td><td>Rc</td></tr>
<tr><td>fsub<i>x</i></td><td>63 (0x3F)</td><td colspan="5">D</td><td colspan="5">A</td><td colspan="5">B</td><td colspan="5">0000_0</td><td colspan="5">0020 (0x014)</td><td>Rc</td></tr>
<tr><td>fadd<i>x</i></td><td>63 (0x3F)</td><td colspan="5">D</td><td colspan="5">A</td><td colspan="5">B</td><td colspan="5">0000_0</td><td colspan="5">0021 (0x015)</td><td>Rc</td></tr>
<tr><td>fsqrt<i>x</i> [5]</td><td>63 (0x3F)</td><td colspan="5">D</td><td colspan="5">00_000</td><td colspan="5">B</td><td colspan="5">0000_0</td><td colspan="5">0022 (0x016)</td><td>Rc</td></tr>
<tr><td>fsel<i>x</i> [3]</td><td>63 (0x3F)</td><td colspan="5">D</td><td colspan="5">A</td><td colspan="5">B</td><td colspan="5">C</td><td colspan="5">0023 (0x017)</td><td>Rc</td></tr>
<tr><td>fmul<i>x</i></td><td>63 (0x3F)</td><td colspan="5">D</td><td colspan="5">A</td><td colspan="5">0_0000</td><td colspan="5">C</td><td colspan="5">0025 (0x019)</td><td>Rc</td></tr>
<tr><td>frsqrte<i>x</i> [3]</td><td>63 (0x3F)</td><td colspan="5">D</td><td colspan="5">00_000</td><td colspan="5">B</td><td colspan="5">0000_0</td><td colspan="5">0026 (0x01A)</td><td>Rc</td></tr>
<tr><td>fmsub<i>x</i></td><td>63 (0x3F)</td><td colspan="5">D</td><td colspan="5">A</td><td colspan="5">B</td><td colspan="5">C</td><td colspan="5">0028 (0x01C)</td><td>Rc</td></tr>
<tr><td>fmadd<i>x</i></td><td>63 (0x3F)</td><td colspan="5">D</td><td colspan="5">A</td><td colspan="5">B</td><td colspan="5">C</td><td colspan="5">0029 (0x01D)</td><td>Rc</td></tr>
<tr><td>fnmsub<i>x</i></td><td>63 (0x3F)</td><td colspan="5">D</td><td colspan="5">A</td><td colspan="5">B</td><td colspan="5">C</td><td colspan="5">0030 (0x01E)</td><td>Rc</td></tr>
<tr><td>fnmadd<i>x</i></td><td>63 (0x3F)</td><td colspan="5">D</td><td colspan="5">A</td><td colspan="5">B</td><td colspan="5">C</td><td colspan="5">0031 (0x01F)</td><td>Rc</td></tr>
<tr><td>fcmpo</td><td>63 (0x3F)</td><td colspan="3">crfD</td><td colspan="2">00</td><td colspan="5">A</td><td colspan="5">B</td><td colspan="10">0032 (0x020)</td><td>0</td></tr>
<tr><td>mtfsb1<i>x</i></td><td>63 (0x3F)</td><td colspan="5">crbD</td><td colspan="5">00_000</td><td colspan="5">0_0000</td><td colspan="10">0038 (0x026)</td><td>Rc</td></tr>
<tr><td>fneg<i>x</i></td><td>63 (0x3F)</td><td colspan="5">D</td><td colspan="5">00_000</td><td colspan="5">B</td><td colspan="10">0040 (0x28)</td><td>Rc</td></tr>
<tr><td>mcrfs</td><td>63 (0x3F)</td><td colspan="3">crfD</td><td colspan="2">00</td><td colspan="3">crfS</td><td colspan="2">00</td><td colspan="5">0_0000</td><td colspan="10">0064 (0x040)</td><td>0</td></tr>
<tr><td>mtfsb0<i>x</i></td><td>63 (0x3F)</td><td colspan="5">crbD</td><td colspan="5">00_000</td><td colspan="5">0_0000</td><td colspan="10">0070 (0x046)</td><td>Rc</td></tr>
<tr><td>fmr<i>x</i></td><td>63 (0x3F)</td><td colspan="5">D</td><td colspan="5">00_000</td><td colspan="5">B</td><td colspan="10">0072 (0x48)</td><td>Rc</td></tr>
<tr><td>mtfsfi<i>x</i></td><td>63 (0x3F)</td><td colspan="3">crfD</td><td colspan="2">00</td><td colspan="5">00_000</td><td colspan="4">IMM</td><td>0</td><td colspan="10">0134 (0x086)</td><td>Rc</td></tr>
<tr><td>fnabs<i>x</i></td><td>63 (0x3F)</td><td colspan="5">D</td><td colspan="5">00_000</td><td colspan="5">B</td><td colspan="10">0136 (0x88)</td><td>Rc</td></tr>
<tr><td>fabs<i>x</i></td><td>63 (0x3F)</td><td colspan="5">D</td><td colspan="5">00_000</td><td colspan="5">B</td><td colspan="10">0264 (0x108)</td><td>Rc</td></tr>
<tr><td>mffs<i>x</i></td><td>63 (0x3F)</td><td colspan="5">D</td><td colspan="5">00_000</td><td colspan="5">0_0000</td><td colspan="10">0583 (0x247)</td><td>Rc</td></tr>
<tr><td>mtfsf<i>x</i></td><td>63 (0x3F)</td><td>0</td><td colspan="8">FM</td><td>0</td><td colspan="5">B</td><td colspan="10">0711 (0x2C7)</td><td>Rc</td></tr>
</table>

[1] AltiVec technology-specific instruction.

[2] Supervisor-level instruction.

[3] Optional to the PowerPC architecture but implemented by the MPC7450.

[4] Supervisor- and user-level instructions.

[5] Optional instruction not implemented by the MPC7450.

[6] Load/store string/multiple instruction.

A.3 Instructions Sorted by Mnemonic (Binary)

Table A-3 shows instructions listed in alphabetical order by mnemonic with binary values.

Key:

▨ Reserved bits

Table A-3. Instructions by Mnemonic (Bin)

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*	011111	D			A	B	OE	100001 010		Rc
addc*x*	011111	D			A	B	OE	000001010		Rc
adde*x*	011111	D			A	B	OE	010001010		Rc
addi	001110	D			A	SIMM				
addic	001100	D			A	SIMM				
addic.	001101	D			A	SIMM				
addis	001111	D			A	SIMM				
addme*x*	011111	D			A	0_0000	OE	11101010		Rc
addze*x*	011111	D			A	0_0000	OE	11001010		Rc
and*x*	011111	S			A	B	000011100			Rc
andc*x*	011111	S			A	B	000111100			Rc
andi.	011100	S			A	UIMM				
andis.	011101	S			A	UIMM				
b*x*	010010	LI							AA	LK
bc*x*	010000	BO			BI	BD			AA	LK
bcctr*x*	010011	BO			BI	0_0000	1000010000			LK
bclr*x*	010011	BO			BI	0_0000	0000010000			LK
cmp	011111	crfD	0	L	A	B	0000000000			0
cmpi	001011	crfD	0	L	A	SIMM				
cmpl	011111	crfD	0	L	A	B	0000100000			0
cmpli	001010	crfD	0	L	A	UIMM				
cntlzw*x*	011111	S			A	0_0000	0000011010			Rc
crand	010011	crbD			crbA	crbB	0100000001			0
crandc	010011	crbD			crbA	crbB	0010000001			0
creqv	010011	crbD			crbA	crbB	0100100001			0
crnand	010011	crbD			crbA	crbB	0011100001			0
crnor	010011	crbD			crbA	crbB	0000100001			0
cror	010011	crbD			crbA	crbB	0111000001			0

Table A-3. Instructions by Mnemonic (Bin) (continued)

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
crorc	010011	crbD			crbA	crbB	0110100001			0
crxor	010011	crbD			crbA	crbB	0011000001			0
dcba[1]	011111	000_00			A	B	1011110110			0
dcbf	011111	000_00			A	B	0001010110			0
dcbi[2]	011111	000_00			A	B	0111010110			0
dcbst	011111	000_00			A	B	0000110110			0
dcbt	011111	000_00			A	B	0100010110			0
dcbtst	011111	000_00			A	B	0011110110			0
dcbz	011111	000_00			A	B	1111110110			0
divw*x*	011111	D			A	B	OE	11110 1011		Rc
divw*x*	011111	D			A	B	OE	11100 1011		Rc
dss[3]	011111	A	00	STRM	00_000	0_0000	1100110110			0
dssall[3]	011111	A	00	STRM	00_000	0_0000	1100110110			0
dst[3]	011111	T	00	STRM	A	B	0101010110			0
dstst[3]	011111	T	00	STRM	A	B	0101110110			0
dststt[3]	011111	1	00	STRM	A	B	0101110110			0
dstt[3]	011111	1	00	STRM	A	B	0101010110			0
eciwx[1]	011111	D			A	B	0100110110			0
ecowx[1]	011111	S			A	B	0110110110			0
eieio	011111	000_00			00_000	0_0000	1101010110			0
eqv*x*	011111	S			A	B	0100011100			Rc
extsb*x*	011111	S			A	0_0000	1110111010			Rc
extsh*x*	011111	S			A	0_0000	1110011010			Rc
fabs*x*	111111	D			00_000	B	0100001000			Rc
fadd*x*	111111	D			A	B	0000_0		1 0101	Rc
fadds*x*	111011	D			A	B	0000_0		1 0101	Rc
fcmpo	111111	crfD		00	A	B	0000100000			0
fcmpu	111111	crfD		00	A	B	0000000000			0
fctiw*x*	111111	D			00_000	B	0000001110			Rc
fctiwz*x*	111111	D			00_000	B	0000001111			Rc
fdiv*x*	111111	D			A	B	0000_0		1 0010	Rc
fdivs*x*	111011	D			A	B	0000_0		1 0010	Rc
fmadd*x*	111111	D			A	B	C		1 1101	Rc

Table A-3. Instructions by Mnemonic (Bin) (continued)

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*	111011	D	A	B	C	1 1101	Rc
fmr*x*	111111	D	00_000	B	0001001000		Rc
fmsub*x*	111111	D	A	B	C	1 1100	Rc
fmsubs*x*	111011	D	A	B	C	1 1100	Rc
fmul*x*	111111	D	A	0_0000	C	1 1001	Rc
fmuls*x*	111011	D	A	0_0000	C	1 1001	Rc
fnabs*x*	111111	D	00_000	B	0010001000		Rc
fneg*x*	111111	D	00_000	B	0000101000		Rc
fnmadd*x*	111111	D	A	B	C	1 1111	Rc
fnmadds*x*	111011	D	A	B	C	1 1111	Rc
fnmsub*x*	111111	D	A	B	C	1 1110	Rc
fnmsubs*x*	111011	D	A	B	C	1 1110	Rc
fres*x*[1]	111011	D	00_000	B	0000_0	1 1000	Rc
frsp*x*	111111	D	00_000	B	0000001100		Rc
frsqrte*x*[1]	111111	D	00_000	B	0000_0	1 1010	Rc
fsel*x*[1]	111111	D	A	B	C	1 0111	Rc
fsqrt*x*[4]	111111	D	00_000	B	0000_0	1 0110	Rc
fsqrts*x*[4]	111011	D	00_000	B	0000_0	1 0110	Rc
fsub*x*	111111	D	A	B	0000_0	1 0100	Rc
fsubs*x*	111011	D	A	B	0000_0	1 0100	Rc
icbi	011111	000_00	A	B	1111010110		0
isync	010011	000_00	00_000	0_0000	0010010110		0
lbz	100010	D	A	d			
lbzu	100011	D	A	d			
lbzux	011111	D	A	B	0001110111		0
lbzx	011111	D	A	B	0001010111		0
lfd	110010	D	A	d			
lfdu	110011	D	A	d			
lfdux	011111	D	A	B	1001110111		0
lfdx	011111	D	A	B	1001010111		0
lfs	110000	D	A	d			
lfsu	110001	D	A	d			
lfsux	011111	D	A	B	1000110111		0

Table A-3. Instructions by Mnemonic (Bin) (continued)

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	011111	D		A		B	1000010111	0
lha	101010	D		A		d		
lhau	101011	D		A		d		
lhaux	011111	D		A		B	0101110111	0
lhax	011111	D		A		B	0101010111	0
lhbrx	011111	D		A		B	1100010110	0
lhz	101000	D		A		d		
lhzu	101001	D		A		d		
lhzux	011111	D		A		B	0100110111	0
lhzx	011111	D		A		B	0100010111	0
lmw[5]	101110	D		A		d		
lswi[5]	011111	D		A		NB	1001010101	0
lswx[5]	011111	D		A		B	1000010101	0
lvebx[3]	011111	**v**D		A		B	0000000111	0
lvehx[3]	011111	**v**D		A		B	0000100111	0
lvewx[3]	011111	**v**D		A		B	0001000111	0
lvsl[3]	011111	**v**D		A		B	0000000110	0
lvsr[3]	011111	**v**D		A		B	0000100110	0
lvx[3]	011111	**v**D		A		B	0001100111	0
lvxl[3]	011111	**v**D		A		B	0101100111	0
lwarx	011111	D		A		B	0000010100	0
lwbrx	011111	D		A		B	1000010110	0
lwz	100000	D		A		d		
lwzu	100001	D		A		d		
lwzux	011111	D		A		B	0000110111	0
lwzx	011111	D		A		B	0000010111	0
mcrf	010011	crfD	00	crfS	00	0_0000	000000000	0
mcrfs	111111	crfD	00	crfS	00	0_0000	001000000	0
mcrxr	011111	crfD	00	00_000		0_0000	1000000000	0
mfcr	011111	D		00_000		0_0000	0000010011	0
mffs*x*	111111	D		00_000		0_0000	1001000111	Rc
mfmsr[2]	011111	D		00_000		0_0000	0001010011	0
mfspr[6]	011111	D		spr			0101010011	0

Table A-3. Instructions by Mnemonic (Bin) (continued)

<table>
<tr><th>Name</th><th>0 5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th><th>17</th><th>18</th><th>19</th><th>20</th><th>21</th><th>22</th><th>23</th><th>24</th><th>25</th><th>26</th><th>27</th><th>28</th><th>29</th><th>30</th><th>31</th></tr>
<tr><td>mfsr[2]</td><td>011111</td><td colspan="5">D</td><td>0</td><td colspan="4">SR</td><td colspan="5">0_0000</td><td colspan="10">1001010011</td><td>0</td></tr>
<tr><td>mfsrin[2]</td><td>011111</td><td colspan="5">D</td><td colspan="5">00_000</td><td colspan="5">B</td><td colspan="10">1010010011</td><td>0</td></tr>
<tr><td>mftb</td><td>011111</td><td colspan="5">D</td><td colspan="10">tbr</td><td colspan="10">0101110011</td><td>0</td></tr>
<tr><td>mfvscr[3]</td><td>000100</td><td colspan="5">vD</td><td colspan="5">00_000</td><td colspan="5">0_0000</td><td colspan="10">11000000100</td><td>0</td></tr>
<tr><td>mtcrf</td><td>011111</td><td colspan="5">S</td><td>0</td><td colspan="8">CRM</td><td>0</td><td colspan="10">0010010000</td><td>0</td></tr>
<tr><td>mtfsb0<i>x</i></td><td>111111</td><td colspan="5">crbD</td><td colspan="5">00_000</td><td colspan="5">0_0000</td><td colspan="10">0001000110</td><td>Rc</td></tr>
<tr><td>mtfsb1<i>x</i></td><td>111111</td><td colspan="5">crbD</td><td colspan="5">00_000</td><td colspan="5">0_0000</td><td colspan="10">0000100110</td><td>Rc</td></tr>
<tr><td>mtfsf<i>x</i></td><td>111111</td><td>0</td><td colspan="8">FM</td><td>0</td><td colspan="5">B</td><td colspan="10">1011000111</td><td>Rc</td></tr>
<tr><td>mtfsfi<i>x</i></td><td>111111</td><td colspan="3">crfD</td><td colspan="2">00</td><td colspan="5">00_000</td><td colspan="4">IMM</td><td>0</td><td colspan="10">0010000110</td><td>Rc</td></tr>
<tr><td>mtmsr[2]</td><td>011111</td><td colspan="5">S</td><td colspan="5">00_000</td><td colspan="5">0_0000</td><td colspan="10">0010010010</td><td>0</td></tr>
<tr><td>mtspr[6]</td><td>011111</td><td colspan="5">S</td><td colspan="10">spr</td><td colspan="10">0111010011</td><td>0</td></tr>
<tr><td>mtsr[2]</td><td>011111</td><td colspan="5">S</td><td>0</td><td colspan="4">SR</td><td colspan="5">0_0000</td><td colspan="10">0011010010</td><td>0</td></tr>
<tr><td>mtsrin [2]</td><td>011111</td><td colspan="5">S</td><td colspan="5">00_000</td><td colspan="5">B</td><td colspan="10">0011110010</td><td>0</td></tr>
<tr><td>mtvscr[3]</td><td>000100</td><td colspan="5">000_00</td><td colspan="5">00_000</td><td colspan="5">vB</td><td colspan="10">11001000100</td><td>0</td></tr>
<tr><td>mulhw<i>x</i></td><td>011111</td><td colspan="5">D</td><td colspan="5">A</td><td colspan="5">B</td><td>0</td><td colspan="9">001001011</td><td>Rc</td></tr>
<tr><td>mulhwu<i>x</i></td><td>011111</td><td colspan="5">D</td><td colspan="5">A</td><td colspan="5">B</td><td>0</td><td colspan="9">000001011</td><td>Rc</td></tr>
<tr><td>mulli</td><td>000111</td><td colspan="5">D</td><td colspan="5">A</td><td colspan="16">SIMM</td></tr>
<tr><td>mullw<i>x</i></td><td>011111</td><td colspan="5">D</td><td colspan="5">A</td><td colspan="5">B</td><td>OE</td><td colspan="9">011101011</td><td>Rc</td></tr>
<tr><td>nand<i>x</i></td><td>011111</td><td colspan="5">S</td><td colspan="5">A</td><td colspan="5">B</td><td colspan="10">0111011100</td><td>Rc</td></tr>
<tr><td>neg<i>x</i></td><td>011111</td><td colspan="5">D</td><td colspan="5">A</td><td colspan="5">0_0000</td><td>OE</td><td colspan="9">001101000</td><td>Rc</td></tr>
<tr><td>nor<i>x</i></td><td>011111</td><td colspan="5">S</td><td colspan="5">A</td><td colspan="5">B</td><td colspan="10">0001111100</td><td>Rc</td></tr>
<tr><td>or<i>x</i></td><td>011111</td><td colspan="5">S</td><td colspan="5">A</td><td colspan="5">B</td><td colspan="10">0110111100</td><td>Rc</td></tr>
<tr><td>orc<i>x</i></td><td>011111</td><td colspan="5">S</td><td colspan="5">A</td><td colspan="5">B</td><td colspan="10">0110011100</td><td>Rc</td></tr>
<tr><td>ori</td><td>011000</td><td colspan="5">S</td><td colspan="5">A</td><td colspan="16">UIMM</td></tr>
<tr><td>oris</td><td>011001</td><td colspan="5">S</td><td colspan="5">A</td><td colspan="16">UIMM</td></tr>
<tr><td>rfi[2]</td><td>010011</td><td colspan="5">000_00</td><td colspan="5">00_000</td><td colspan="5">0_0000</td><td colspan="10">0000110010</td><td>0</td></tr>
<tr><td>rlwimi<i>x</i></td><td>010100</td><td colspan="5">S</td><td colspan="5">A</td><td colspan="5">SH</td><td colspan="5">MB</td><td colspan="5">ME</td><td>Rc</td></tr>
<tr><td>rlwinm<i>x</i></td><td>010101</td><td colspan="5">S</td><td colspan="5">A</td><td colspan="5">SH</td><td colspan="5">MB</td><td colspan="5">ME</td><td>Rc</td></tr>
<tr><td>rlwnm<i>x</i></td><td>010111</td><td colspan="5">S</td><td colspan="5">A</td><td colspan="5">B</td><td colspan="5">MB</td><td colspan="5">ME</td><td>Rc</td></tr>
<tr><td>sc</td><td>010001</td><td colspan="24">000_0000_0000_0000_0000_0000_00</td><td>1</td><td>0</td></tr>
<tr><td>slw<i>x</i></td><td>011111</td><td colspan="5">S</td><td colspan="5">A</td><td colspan="5">B</td><td colspan="10">0000011000</td><td>Rc</td></tr>
<tr><td>sraw<i>x</i></td><td>011111</td><td colspan="5">S</td><td colspan="5">A</td><td colspan="5">B</td><td colspan="10">1100011000</td><td>Rc</td></tr>
<tr><td>srawi<i>x</i></td><td>011111</td><td colspan="5">S</td><td colspan="5">A</td><td colspan="5">SH</td><td colspan="10">1100011000</td><td>Rc</td></tr>
</table>

Table A-3. Instructions by Mnemonic (Bin) (continued)

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
srw*x*	011111	S	A	B	1000011000	Rc
stb	100110	S	A	d		
stbu	100111	S	A	d		
stbux	011111	S	A	B	0011110111	0
stbx	011111	S	A	B	0011010111	0
stfd	110110	S	A	d		
stfdu	110111	S	A	d		
stfdux	011111	S	A	B	1011110111	0
stfdx	011111	S	A	B	1011010111	0
stfiwx[1]	011111	S	A	B	1111010111	0
stfs	110100	S	A	d		
stfsu	110101	S	A	d		
stfsux	011111	S	A	B	1010110111	0
stfsx	011111	S	A	B	1010010111	0
sth	101100	S	A	d		
sthbrx	011111	S	A	B	1110010110	0
sthu	101101	S	A	d		
sthux	011111	S	A	B	110110111	0
sthx	011111	S	A	B	110010111	0
stmw[5]	101111	S	A	d		
stswi[5]	011111	S	A	NB	1011010101	0
stswx[5]	011111	S	A	B	1010010101	0
stvebx[3]	011111	**v**S	A	B	0010000111	0
stvehx[3]	011111	**v**S	A	B	0010100111	0
stvewx[3]	011111	**v**S	A	B	0011000111	0
stvx[3]	011111	**v**S	A	B	0011100111	0
stvxl[3]	011111	**v**S	A	B	0111100111	0
stw	100100	S	A	d		
stwbrx	011111	S	A	B	1010010110	0
stwcx.	011111	S	A	B	10010110	1
stwu	100101	S	A	d		
stwux	011111	S	A	B	10110111	0
stwx	011111	S	A	B	10010111	0

Table A-3. Instructions by Mnemonic (Bin) (continued)

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
subf*x*	011111	D	A	B	OE	000101000	Rc
subfc*x*	011111	D	A	B	OE	000001000	Rc
subfe*x*	011111	D	A	B	OE	010001000	Rc
subfic	001000	D	A	SIMM			
subfme*x*	011111	D	A	0_0000	OE	011101000	Rc
subfze*x*	011111	D	A	0_0000	OE	011001000	Rc
sync	011111	000_00	00_000	0_0000	1001010110		0
tlbia [4]	011111	000_00	00_000	0_0000	0101110010		0
tlbie[1, 2]	011111	000_00	00_000	B	0100110010		0
tlbld[1, 2]	011111	000_00	00_000	B	1111010010		0
tlbli[1, 2]	011111	000_00	00_000	B	1111110010		0
tlbsync[1, 2]	011111	000_00	00_000	0_0000	1000110110		0
tw	011111	TO	A	B	0000000100		0
twi	000011	TO	A	SIMM			
vaddcuw[3]	000100	**v**D	**v**A	**v**B	0110000000		0
vaddfp[3]	000100	**v**D	**v**A	**v**B	0000001010		0
vaddsbs[3]	000100	**v**D	**v**A	**v**B	1100000000		0
vaddshs[3]	000100	**v**D	**v**A	**v**B	1101000000		0
vaddsws[3]	000100	**v**D	**v**A	**v**B	1110000000		0
vaddubm[3]	000100	**v**D	**v**A	**v**B	0000000000		0
vaddubs[3]	000100	**v**D	**v**A	**v**B	1000000000		0
vadduhm[3]	000100	**v**D	**v**A	**v**B	001000000		0
vadduhs[3]	000100	**v**D	**v**A	**v**B	1001000000		0
vadduwm[3]	000100	**v**D	**v**A	**v**B	0010000000		0
vadduws[3]	000100	**v**D	**v**A	**v**B	1010000000		0
vand[3]	000100	**v**D	**v**A	**v**B	10000000100		0
vandc[3]	000100	**v**D	**v**A	**v**B	10001000100		0
vavgsb[3]	000100	**v**D	**v**A	**v**B	10100000010		0
vavgsh[3]	000100	**v**D	**v**A	**v**B	10101000010		0
vavgsw[3]	000100	**v**D	**v**A	**v**B	10110000010		0
vavgub[3]	000100	**v**D	**v**A	**v**B	10000000010		0
vavguh[3]	000100	**v**D	**v**A	**v**B	10001000010		0
vavguw[3]	000100	**v**D	**v**A	**v**B	10010000010		0

Table A-3. Instructions by Mnemonic (Bin) (continued)

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
vcfsx[3]	000100	vD	UIMM	vB	01101001010	
vcfux[3]	000100	vD	UIMM	vB	1100001010	0
vcmpbfp*x*[3]	000100	vD	vA	vB	Rc	1111000110
vcmpeqfp*x*[3]	000100	vD	vA	vB	Rc	0011000110
vcmpequb*x*[3]	000100	vD	vA	vB	Rc	0000000110
vcmpequh*x*[3]	000100	vD	vA	vB	Rc	0001000110
vcmpequw*x*[3]	000100	vD	vA	vB	Rc	0010000110
vcmpgefp*x*[3]	000100	vD	vA	vB	Rc	0111000110
vcmpgtfp*x*[3]	000100	vD	vA	vB	Rc	1011000110
vcmpgtsb*x*[3]	000100	vD	vA	vB	Rc	1100000110
vcmpgtsh*x*[3]	000100	vD	vA	vB	Rc	1101000110
vcmpgtsw*x*[3]	000100	vD	vA	vB	Rc	1110000110
vcmpgtub*x*[3]	000100	vD	vA	vB	Rc	1000000110
vcmpgtuh*x*[3]	000100	vD	vA	vB	Rc	1001000110
vcmpgtuw*x*[3]	000100	vD	vA	vB	Rc	1010000110
vctsxs[3]	000100	vD	UIMM	vB	1111001010	
vctuxs[3]	000100	vD	UIMM	vB	1110001010	
vexptefp[3]	000100	vD	00_000	vB	110001010	
vlogefp[3]	000100	vD	00_000	vB	111001010	
vmaddfp[3]	000100	vD	vA	vB	vC	101110
vmaxfp[3]	000100	vD	vA	vB	10000001010	
vmaxsb[3]	000100	vD	vA	vB	0100000010	
vmaxsh[3]	000100	vD	vA	vB	0101000010	
vmaxsw[3]	000100	vD	vA	vB	0110000010	
vmaxub[3]	000100	vD	vA	vB	0000000010	
vmaxuh[3]	000100	vD	vA	vB	0001000010	
vmaxuw[3]	000100	vD	vA	vB	0010000010	
vmhaddshs[3]	000100	vD	vA	vB	vC	100000
vmhraddshs[3]	000100	vD	vA	vB	vC	100001
vminfp[3]	000100	vD	vA	vB	10001001010	
vminsb[3]	000100	vD	vA	vB	1100000010	
vminsh[3]	000100	vD	vA	vB	1101000010	
vminsw[3]	000100	vD	vA	vB	1110000010	

Table A-3. Instructions by Mnemonic (Bin) (continued)

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
vminub[3]	000100	vD	vA	vB	1000000010	
vminuh[3]	000100	vD	vA	vB	1001000010	
vminuw[3]	000100	vD	vA	vB	1010000010	
vmladduhm[3]	000100	vD	vA	vB	vC	100010
vmrghb[3]	000100	vD	vA	vB	0000001100	
vmrghh[3]	000100	vD	vA	vB	0001001100	
vmrghw[3]	000100	vD	vA	vB	0010001100	
vmrglb[3]	000100	vD	vA	vB	0100001100	
vmrglh[3]	000100	vD	vA	vB	0101001100	
vmrglw[3]	000100	vD	vA	vB	0110001100	
vmsummbm[3]	000100	vD	vA	vB	vC	100101
vmsumshm[3]	000100	vD	vA	vB	vC	101000
vmsumshs[3]	000100	vD	vA	vB	vC	101001
vmsumubm[3]	000100	vD	vA	vB	vC	100100
vmsumuhm[3]	000100	vD	vA	vB	vC	100110
vmsumuhs[3]	000100	vD	vA	vB	vC	100111
vmulesb[3]	000100	vD	vA	vB	0100001000	
vmulesh[3]	000100	vD	vA	vB	1101001000	
vmuleub[3]	000100	vD	vA	vB	1000001000	
vmuleuh[3]	000100	vD	vA	vB	1001001000	
vmulosb[3]	000100	vD	vA	vB	0100001000	
vmulosh[3]	000100	vD	vA	vB	0101001000	
vmuloub[3]	000100	vD	vA	vB	0000001000	
vmulouh[3]	000100	vD	vA	vB	0001001000	
vnmsubfp[3]	000100	vD	vA	vB	vC	101111
vnor[3]	000100	vD	vA	vB	10100000100	
vor[3]	000100	vD	vA	vB	10010000100	
vperm[3]	000100	vD	vA	vB	vC	101011
vpkpx[3]	000100	vD	vA	vB	1100001110	
vpkshss[3]	000100	vD	vA	vB	0110001110	
vpkshus[3]	000100	vD	vA	vB	0100001110	
vpkswss[3]	000100	vD	vA	vB	0111001110	
vpkswus[3]	000100	vD	vA	vB	0101001110	

Table A-3. Instructions by Mnemonic (Bin) (continued)

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
vpkuhum[3]	000100	**v**D	**v**A	**v**B	0000001110		
vpkuhus[3]	000100	**v**D	**v**A	**v**B	0010001110		
vpkuwum[3]	000100	**v**D	**v**A	**v**B	0001001110		
vpkuwus[3]	000100	**v**D	**v**A	**v**B	0011001110		
vrefp[3]	000100	**v**D	00_000	**v**B	0100001010		
vrfim[3]	000100	**v**D	00_000	**v**B	1011001010		
vrfin[3]	000100	**v**D	00_000	**v**B	1000001010		
vrfip[3]	000100	**v**D	00_000	**v**B	1010001010		
vrfiz[3]	000100	**v**D	00_000	**v**B	1001001010		
vrlb[3]	000100	**v**D	**v**A	**v**B	0000000100		
vrlh[3]	000100	**v**D	**v**A	**v**B	0001000100		
vrlw[3]	000100	**v**D	**v**A	**v**B	0010000100		
vrsqrtefp[3]	000100	**v**D	00_000	**v**B	0101001010		
vsel[3]	000100	**v**D	**v**A	**v**B	**v**C		101010
vsl[3]	000100	**v**D	**v**A	**v**B	0111000100		
vslb[3]	000100	**v**D	**v**A	**v**B	0100000100		
vsldoi[3]	000100	**v**D	**v**A	**v**B	0	SH	101100
vslh[3]	000100	**v**D	**v**A	**v**B	0101000100		
vslo[3]	000100	**v**D	**v**A	**v**B	10000001100		
vslw[3]	000100	**v**D	**v**A	**v**B	0110000100		
vspltb[3]	000100	**v**D	UIMM	**v**B	1000001100		
vsplth[3]	000100	**v**D	UIMM	**v**B	1001001100		
vspltisb[3]	000100	**v**D	SIMM	0_0000	1100001100		
vspltish[3]	000100	**v**D	SIMM	0_0000	1101001100		
vspltisw[3]	000100	**v**D	SIMM	0_0000	1110001100		
vspltw[3]	000100	**v**D	UIMM	**v**B	1010001100		
vsr[3]	000100	**v**D	**v**A	**v**B	1011000100		
vsrab[3]	000100	**v**D	**v**A	**v**B	1100000100		
vsrah[3]	000100	**v**D	**v**A	**v**B	1101000100		
vsraw[3]	000100	**v**D	**v**A	**v**B	1110000100		
vsrb[3]	000100	**v**D	**v**A	**v**B	1000000100		
vsrh[3]	000100	**v**D	**v**A	**v**B	1001000100		
vsro[3]	000100	**v**D	**v**A	**v**B	10001001100		

Table A-3. Instructions by Mnemonic (Bin) (continued)

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
vsrw[3]	000100	**v**D	**v**A	**v**B	1010000100	
vsubcuw[3]	000100	**v**D	**v**A	**v**B	10110000000	
vsubfp[3]	000100	**v**D	**v**A	**v**B	0001001010	
vsubsbs[3]	000100	**v**D	**v**A	**v**B	11100000000	
vsubshs[3]	000100	**v**D	**v**A	**v**B	11101000000	
vsubsws[3]	000100	**v**D	**v**A	**v**B	11110000000	
vsububm[3]	000100	**v**D	**v**A	**v**B	10000000000	
vsububs[3]	000100	**v**D	**v**A	**v**B	11000000000	
vsubuhm[3]	000100	**v**D	**v**A	**v**B	10001000000	
vsubuhs[3]	000100	**v**D	**v**A	**v**B	11001000000	
vsubuwm[3]	000100	**v**D	**v**A	**v**B	10010000000	
vsubuws[3]	000100	**v**D	**v**A	**v**B	11010000000	
vsumsws[3]	000100	**v**D	**v**A	**v**B	11110001000	
vsum2sws[3]	000100	**v**D	**v**A	**v**B	11010001000	
vsum4sbs[3]	000100	**v**D	**v**A	**v**B	11100001000	
vsum4shs[3]	000100	**v**D	**v**A	**v**B	11001001000	
vsum4ubs[3]	000100	**v**D	**v**A	**v**B	11000001000	
vupkhpx[3]	000100	**v**D	00_000	**v**B	1101001110	
vupkhsb[3]	000100	**v**D	00_000	**v**B	1000001110	
vupkhsh[3]	000100	**v**D	00_000	**v**B	1001001110	
vupklpx[3]	000100	**v**D	00_000	**v**B	1111001110	
vupklsb[3]	000100	**v**D	00_000	**v**B	1010001110	
vupklsh[3]	000100	**v**D	00_000	**v**B	1011001110	
vxor[3]	000100	**v**D	**v**A	**v**B	10011000100	
xor*x*	011111	S	A	B	0100111100	Rc
xori	011010	S	A	UIMM		
xoris	011011	S	A	UIMM		

[1] Optional to the PowerPC architecture but implemented by the MPC7450.

[2] Supervisor-level instruction.

[3] AltiVec technology-specific instruction.

[4] Optional instruction not implemented by the MPC7450.

[5] Load/store string/multiple instruction.

[6] Supervisor- and user-level instruction.

A.4 Instructions Sorted by Opcode (Binary)

Table A-4 lists the instructions implemented in the MPC7450 in binary numerical order by opcode.

Key:

▨ Reserved bits

Table A-4. Instructions by Primary and Secondary Opcode (Bin)

Name	0–5	6–10	11–15	16–20	21	22–25	26–30	31
twi	000011	TO	A	SIMM				
vaddubm[1]	000100	**v**D	**v**A	**v**B	0000000000			0
vmaxub[1]	000100	**v**D	**v**A	**v**B	0000000010			
vrlb[1]	000100	**v**D	**v**A	**v**B	0000000100			
vcmpequb*x*[1]	000100	**v**D	**v**A	**v**B	Rc	0000000110		
vmuloub[1]	000100	**v**D	**v**A	**v**B	0000001000			
vaddfp[1]	000100	**v**D	**v**A	**v**B	0000001010			0
vmrghb[1]	000100	**v**D	**v**A	**v**B	0000001100			
vpkuhum[1]	000100	**v**D	**v**A	**v**B	0000001110			
vmhaddshs[1]	000100	**v**D	**v**A	**v**B	**v**C		100000	
vmhraddshs[1]	000100	**v**D	**v**A	**v**B	**v**C		100001	
vmladduhm[1]	000100	**v**D	**v**A	**v**B	**v**C		100010	
vmsumubm[1]	000100	**v**D	**v**A	**v**B	**v**C		100100	
vmsummbm[1]	000100	**v**D	**v**A	**v**B	**v**C		100101	
vmsumuhm[1]	000100	**v**D	**v**A	**v**B	**v**C		100110	
vmsumuhs[1]	000100	**v**D	**v**A	**v**B	**v**C		100111	
vmsumshm[1]	000100	**v**D	**v**A	**v**B	**v**C		101000	
vmsumshs[1]	000100	**v**D	**v**A	**v**B	**v**C		101001	
vsel[1]	000100	**v**D	**v**A	**v**B	**v**C		101010	
vperm[1]	000100	**v**D	**v**A	**v**B	**v**C		101011	
vsldoi[1]	000100	**v**D	**v**A	**v**B	0	SH	101100	
vmaddfp[1]	000100	**v**D	**v**A	**v**B	**v**C		101110	
vnmsubfp[1]	000100	**v**D	**v**A	**v**B	**v**C		101111	
vadduhm[1]	000100	**v**D	**v**A	**v**B	001000000			0
vmaxuh[1]	000100	**v**D	**v**A	**v**B	0001000010			
vrlh[1]	000100	**v**D	**v**A	**v**B	0001000100			
vcmpequh*x*[1]	000100	**v**D	**v**A	**v**B	Rc	0001000110		
vmulouh[1]	000100	**v**D	**v**A	**v**B	0001001000			

Table A-4. Instructions by Primary and Secondary Opcode (Bin) (continued)

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
vsubfp[1]	000100	vD	vA	vB	0001001010		
vmrghh[1]	000100	vD	vA	vB	0001001100		
vpkuwum[1]	000100	vD	vA	vB	0001001110		
vadduwm[1]	000100	vD	vA	vB	0010000000		0
vmaxuw[1]	000100	vD	vA	vB	0010000010		
vrlw[1]	000100	vD	vA	vB	0010000100		
vcmpequw*x*[1]	000100	vD	vA	vB	Rc	0010000110	
vmrghw[1]	000100	vD	vA	vB	0010001100		
vpkuhus[1]	000100	vD	vA	vB	0010001110		
vcmpeqfp*x*[1]	000100	vD	vA	vB	Rc	0011000110	
vpkuwus[1]	000100	vD	vA	vB	0011001110		
vmaxsb[1]	000100	vD	vA	vB	0100000010		
vslb[1]	000100	vD	vA	vB	0100000100		
vmulosb[1]	000100	vD	vA	vB	0100001000		
vrefp[1]	000100	vD	00_000	vB	0100001010		
vmrglb[1]	000100	vD	vA	vB	0100001100		
vpkshus[1]	000100	vD	vA	vB	0100001110		
vmaxsh[1]	000100	vD	vA	vB	0101000010		
vslh[1]	000100	vD	vA	vB	0101000100		
vmulosh[1]	000100	vD	vA	vB	0101001000		
vrsqrtefp[1]	000100	vD	00_000	vB	0101001010		
vmrglh[1]	000100	vD	vA	vB	0101001100		
vpkswus[1]	000100	vD	vA	vB	0101001110		
vaddcuw[1]	000100	vD	vA	vB	0110000000		0
vmaxsw[1]	000100	vD	vA	vB	0110000010		
vslw[1]	000100	vD	vA	vB	0110000100		
vexptefp[1]	000100	vD	00_000	vB	110001010		
vmrglw[1]	000100	vD	vA	vB	0110001100		
vpkshss[1]	000100	vD	vA	vB	0110001110		
vsl[1]	000100	vD	vA	vB	0111000100		
vcmpgefp*x*[1]	000100	vD	vA	vB	Rc	0111000110	
vlogefp[1]	000100	vD	00_000	vB	111001010		
vpkswss[1]	000100	vD	vA	vB	0111001110		

Table A-4. Instructions by Primary and Secondary Opcode (Bin) (continued)

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
vaddubs[1]	000100	**v**D	**v**A	**v**B		1000000000	0
vminub[1]	000100	**v**D	**v**A	**v**B		10000000010	
vsrb[1]	000100	**v**D	**v**A	**v**B		10000000100	
vcmpgtub*x*[1]	000100	**v**D	**v**A	**v**B	Rc	1000000110	
vmuleub[1]	000100	**v**D	**v**A	**v**B		10000001000	
vrfin[1]	000100	**v**D	00_000	**v**B		10000001010	
vspltb[1]	000100	**v**D	UIMM	**v**B		10000001100	
vupkhsb[1]	000100	D	00_000	B		10000001110	
vadduhs[1]	000100	**v**D	**v**A	**v**B		1001000000	0
vminuh[1]	000100	**v**D	**v**A	**v**B		10010000010	
vsrh[1]	000100	**v**D	**v**A	**v**B		10010000100	
vcmpgtuh*x*[1]	000100	**v**D	**v**A	**v**B	Rc	1001000110	
vmuleuh[1]	000100	**v**D	**v**A	**v**B		10010001000	
vrfiz[1]	000100	**v**D	00_000	**v**B		10010001010	
vsplth[1]	000100	**v**D	UIMM	**v**B		10010001100	
vupkhsh[1]	000100	D	00_000	B		10010001110	
vadduws[1]	000100	**v**D	**v**A	**v**B		1010000000	0
vminuw[1]	000100	**v**D	**v**A	**v**B		10100000010	
vsrw[1]	000100	**v**D	**v**A	**v**B		10100000100	
vcmpgtuw*x*[1]	000100	**v**D	**v**A	**v**B	Rc	1010000110	
vrfip[1]	000100	**v**D	00_000	**v**B		10100001010	
vspltw[1]	000100	**v**D	UIMM	**v**B		10100001100	
vupklsb[1]	000100	D	00_000	B		10100001110	
vsr[1]	000100	**v**D	**v**A	**v**B		10110000100	
vcmpgtfp*x*[1]	000100	**v**D	**v**A	**v**B	Rc	1011000110	
vrfim[1]	000100	**v**D	00_000	**v**B		10110001010	
vupklsh[1]	000100	D	00_000	B		10110001110	
vaddsbs[1]	000100	**v**D	**v**A	**v**B		1100000000	0
vminsb[1]	000100	**v**D	**v**A	**v**B		11000000010	
vsrab[1]	000100	**v**D	**v**A	**v**B		11000000100	
vcmpgtsb*x*[1]	000100	**v**D	**v**A	**v**B	Rc	1100000110	
vmulesb[1]	000100	**v**D	**v**A	**v**B		11000001000	
vcfux[1]	000100	**v**D	UIMM	**v**B		1100001010	0

Table A-4. Instructions by Primary and Secondary Opcode (Bin) (continued)

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
vspltisb[1]	000100	vD	SIMM	0_0000		1100001100		
vpkpx[1]	000100	vD	vA	vB		1100001110		
vaddshs[1]	000100	vD	vA	vB		1101000000		0
vminsh[1]	000100	vD	vA	vB		1101000010		
vsrah[1]	000100	vD	vA	vB		1101000100		
vcmpgtsh*x*[1]	000100	vD	vA	vB	Rc	1101000110		
vmulesh[1]	000100	vD	vA	vB		1101001000		
vcfsx[1]	000100	vD	UIMM	vB		01101001010		
vspltish[1]	000100	vD	SIMM	0_0000		1101001100		
vupkhpx[1]	000100	vD	00_000	vB		1101001110		
vaddsws[1]	000100	vD	vA	vB		1110000000		0
vminsw[1]	000100	vD	vA	vB		1110000010		
vsraw[1]	000100	vD	vA	vB		1110000100		
vcmpgtsw*x*[1]	000100	vD	vA	vB	Rc	1110000110		
vctuxs[1]	000100	vD	UIMM	vB		1110001010		
vspltisw[1]	000100	vD	SIMM	0_0000		1110001100		
vcmpbfp*x*[1]	000100	vD	vA	vB	Rc	1111000110		
vctsxs[1]	000100	vD	UIMM	vB		1111001010		
vupklpx[1]	000100	vD	00_000	vB		1111001110		
vsububm[1]	000100	vD	vA	vB		10000000000		
vavgub[1]	000100	vD	vA	vB		10000000010		0
vand[1]	000100	vD	vA	vB		10000000100		0
vmaxfp[1]	000100	vD	vA	vB		10000001010		
vslo[1]	000100	vD	vA	vB		10000001100		
vsubuhm[1]	000100	vD	vA	vB		10001000000		
vavguh[1]	000100	vD	vA	vB		10001000010		0
vandc[1]	000100	vD	vA	vB		10001000100		0
vminfp[1]	000100	vD	vA	vB		10001001010		
vsro[1]	000100	vD	vA	vB		10001001100		
vsubuwm[1]	000100	vD	vA	vB		10010000000		
vavguw[1]	000100	vD	vA	vB		10010000010		0
vor[1]	000100	vD	vA	vB		10010000100		
vxor[1]	000100	vD	vA	vB		10011000100		

Table A-4. Instructions by Primary and Secondary Opcode (Bin) (continued)

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
vavgsb[1]	000100	**v**D			**v**A		**v**B	10100000010		0
vnor[1]	000100	**v**D			**v**A		**v**B	10100000100		
vavgsh[1]	000100	**v**D			**v**A		**v**B	10101000010		0
vsubcuw[1]	000100	**v**D			**v**A		**v**B	10110000000		
vavgsw[1]	000100	**v**D			**v**A		**v**B	10110000010		0
vsububs[1]	000100	**v**D			**v**A		**v**B	11000000000		
mfvscr[1]	000100	**v**D			00_000		0_0000	11000000100		0
vsum4ubs[1]	000100	**v**D			**v**A		**v**B	11000001000		
vsubuhs[1]	000100	**v**D			**v**A		**v**B	11001000000		
mtvscr[1]	000100	000_00			00_000		**v**B	11001000100		0
vsum4shs[1]	000100	**v**D			**v**A		**v**B	11001001000		
vsubuws[1]	000100	**v**D			**v**A		**v**B	11010000000		
vsum2sws[1]	000100	**v**D			**v**A		**v**B	11010001000		
vsubsbs[1]	000100	**v**D			**v**A		**v**B	11100000000		
vsum4sbs[1]	000100	**v**D			**v**A		**v**B	11100001000		
vsubshs[1]	000100	**v**D			**v**A		**v**B	11101000000		
vsubsws[1]	000100	**v**D			**v**A		**v**B	11110000000		
vsumsws[1]	000100	**v**D			**v**A		**v**B	11110001000		
mulli	000111	D			A		SIMM			
subfic	001000	D			A		SIMM			
cmpli	001010	crfD	0	L	A		UIMM			
cmpi	001011	crfD	0	L	A		SIMM			
addic	001100	D			A		SIMM			
addic.	001101	D			A		SIMM			
addi	001110	D			A		SIMM			
addis	001111	D			A		SIMM			
bc*x*	010000	BO			BI		BD		AA	LK
sc	010001	000_0000_0000_0000_0000_0000_00							1	0
b*x*	010010	LI							AA	LK
mcrf	010011	crfD	00		crfS	00	0_0000	000000000		0
bclr*x*	010011	BO			BI		0_0000	0000010000		LK
crnor	010011	crbD			crbA		crbB	0000100001		0
rfi[2]	010011	000_00			00_000		0_0000	0000110010		0

Table A-4. Instructions by Primary and Secondary Opcode (Bin) (continued)

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
crandc	010011	crbD			crbA	crbB	0010000001			0
isync	010011	000_00			00_000	0_0000	0010010110			0
crxor	010011	crbD			crbA	crbB	0011000001			0
crnand	010011	crbD			crbA	crbB	0011100001			0
crand	010011	crbD			crbA	crbB	0100000001			0
creqv	010011	crbD			crbA	crbB	0100100001			0
crorc	010011	crbD			crbA	crbB	0110100001			0
cror	010011	crbD			crbA	crbB	0111000001			0
bcctr*x*	010011	BO			BI	0_0000	1000010000			LK
rlwimi*x*	010100	S			A	SH	MB		ME	Rc
rlwinm*x*	010101	S			A	SH	MB		ME	Rc
rlwnm*x*	010111	S			A	B	MB		ME	Rc
ori	011000	S			A	UIMM				
oris	011001	S			A	UIMM				
xori	011010	S			A	UIMM				
xoris	011011	S			A	UIMM				
andi.	011100	S			A	UIMM				
andis.	011101	S			A	UIMM				
cmp	011111	crfD	0	L	A	B	0000000000			0
tw	011111	TO			A	B	0000000100			0
lvsl[1]	011111	**v**D			A	B	0000000110			0
lvebx[1]	011111	**v**D			A	B	0000000111			0
subfc*x*	011111	D			A	B	OE	000001000		Rc
addc*x*	011111	D			A	B	OE	000001010		Rc
mulhwu*x*	011111	D			A	B	0	000001011		Rc
mfcr	011111	D			00_000	0_0000	0000010011			0
lwarx	011111	D			A	B	0000010100			0
lwzx	011111	D			A	B	0000010111			0
slw*x*	011111	S			A	B	0000011000			Rc
cntlzw*x*	011111	S			A	0_0000	0000011010			Rc
and*x*	011111	S			A	B	000011100			Rc
cmpl	011111	crfD	0	L	A	B	0000100000			0
lvsr[1]	011111	**v**D			A	B	0000100110			0

Table A-4. Instructions by Primary and Secondary Opcode (Bin) (continued)

<table>
<tr><th>Name</th><th>0 5</th><th>6 7 8 9 10</th><th>11</th><th>12 13 14 15</th><th>16 17 18 19</th><th>20</th><th>21</th><th>22 23 24 25 26 27 28 29 30</th><th>31</th></tr>
<tr><td>lvehx[1]</td><td>011111</td><td>vD</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0000100111</td><td>0</td></tr>
<tr><td>subf<i>x</i></td><td>011111</td><td>D</td><td colspan="2">A</td><td colspan="2">B</td><td>OE</td><td>000101000</td><td>Rc</td></tr>
<tr><td>dcbst</td><td>011111</td><td>000_00</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0000110110</td><td>0</td></tr>
<tr><td>lwzux</td><td>011111</td><td>D</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0000110111</td><td>0</td></tr>
<tr><td>andc<i>x</i></td><td>011111</td><td>S</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">000111100</td><td>Rc</td></tr>
<tr><td>lvewx[1]</td><td>011111</td><td>vD</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0001000111</td><td>0</td></tr>
<tr><td>mulhw<i>x</i></td><td>011111</td><td>D</td><td colspan="2">A</td><td colspan="2">B</td><td>0</td><td>001001011</td><td>Rc</td></tr>
<tr><td>mfmsr [2]</td><td>011111</td><td>D</td><td colspan="2">00_000</td><td colspan="2">0_0000</td><td colspan="2">0001010011</td><td>0</td></tr>
<tr><td>dcbf</td><td>011111</td><td>000_00</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0001010110</td><td>0</td></tr>
<tr><td>lbzx</td><td>011111</td><td>D</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0001010111</td><td>0</td></tr>
<tr><td>lvx[1]</td><td>011111</td><td>vD</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0001100111</td><td>0</td></tr>
<tr><td>neg<i>x</i></td><td>011111</td><td>D</td><td colspan="2">A</td><td colspan="2">0_0000</td><td>OE</td><td>001101000</td><td>Rc</td></tr>
<tr><td>lbzux</td><td>011111</td><td>D</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0001110111</td><td>0</td></tr>
<tr><td>nor<i>x</i></td><td>011111</td><td>S</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0001111100</td><td>Rc</td></tr>
<tr><td>stvebx [1]</td><td>011111</td><td>vS</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0010000111</td><td>0</td></tr>
<tr><td>subfe<i>x</i></td><td>011111</td><td>D</td><td colspan="2">A</td><td colspan="2">B</td><td>OE</td><td>010001000</td><td>Rc</td></tr>
<tr><td>adde<i>x</i></td><td>011111</td><td>D</td><td colspan="2">A</td><td colspan="2">B</td><td>OE</td><td>010001010</td><td>Rc</td></tr>
<tr><td>mtcrf</td><td>011111</td><td>S</td><td>0</td><td colspan="2">CRM</td><td>0</td><td colspan="2">0010010000</td><td>0</td></tr>
<tr><td>mtmsr [2]</td><td>011111</td><td>S</td><td colspan="2">00_000</td><td colspan="2">0_0000</td><td colspan="2">0010010010</td><td>0</td></tr>
<tr><td>stwcx.</td><td>011111</td><td>S</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">10010110</td><td>1</td></tr>
<tr><td>stwx</td><td>011111</td><td>S</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">10010111</td><td>0</td></tr>
<tr><td>stvehx [1]</td><td>011111</td><td>vS</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0010100111</td><td>0</td></tr>
<tr><td>stwux</td><td>011111</td><td>S</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">10110111</td><td>0</td></tr>
<tr><td>stvewx [1]</td><td>011111</td><td>vS</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0011000111</td><td>0</td></tr>
<tr><td>subfze<i>x</i></td><td>011111</td><td>D</td><td colspan="2">A</td><td colspan="2">0_0000</td><td>OE</td><td>011001000</td><td>Rc</td></tr>
<tr><td>addze<i>x</i></td><td>011111</td><td>D</td><td colspan="2">A</td><td colspan="2">0_0000</td><td>OE</td><td>11001010</td><td>Rc</td></tr>
<tr><td>mtsr [2]</td><td>011111</td><td>S</td><td>0</td><td>SR</td><td colspan="2">0_0000</td><td colspan="2">0011010010</td><td>0</td></tr>
<tr><td>stbx</td><td>011111</td><td>S</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0011010111</td><td>0</td></tr>
<tr><td>stvx [1]</td><td>011111</td><td>vS</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0011100111</td><td>0</td></tr>
<tr><td>subfme<i>x</i></td><td>011111</td><td>D</td><td colspan="2">A</td><td colspan="2">0_0000</td><td>OE</td><td>011101000</td><td>Rc</td></tr>
<tr><td>addme<i>x</i></td><td>011111</td><td>D</td><td colspan="2">A</td><td colspan="2">0_0000</td><td>OE</td><td>11101010</td><td>Rc</td></tr>
<tr><td>mullw<i>x</i></td><td>011111</td><td>D</td><td colspan="2">A</td><td colspan="2">B</td><td>OE</td><td>011101011</td><td>Rc</td></tr>
<tr><td>mtsrin [2]</td><td>011111</td><td>S</td><td colspan="2">00_000</td><td colspan="2">B</td><td colspan="2">0011110010</td><td>0</td></tr>
</table>

Table A-4. Instructions by Primary and Secondary Opcode (Bin) (continued)

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
dcbtst	011111	000_00			A	B	0011110110		0
stbux	011111	S			A	B	0011110111		0
add*x*	011111	D			A	B	OE	100 001 010	Rc
dcbt	011111	000_00			A	B	0100010110		0
lhzx	011111	D			A	B	0100010111		0
eqv*x*	011111	S			A	B	0100011100		Rc
tlbie [2, 3]	011111	000_00			00_000	B	0100110010		0
eciwx [3]	011111	D			A	B	0100110110		0
lhzux	011111	D			A	B	0100110111		0
xor*x*	011111	S			A	B	0100111100		Rc
mfspr[4]	011111	D			spr		0101010011		0
dst [1]	011111	T	00	STRM	A	B	0101010110		0
dstt [1]	011111	1	00	STRM	A	B	0101010110		0
lhax	011111	D			A	B	0101010111		0
lvxl[1]	011111	**v**D			A	B	0101100111		0
tlbia[5]	011111	000_00			00_000	0_0000	0101110010		0
mftb	011111	D			tbr		0101110011		0
dstst[1]	011111	T	00	STRM	A	B	0101110110		0
dststt[1]	011111	1	00	STRM	A	B	0101110110		0
lhaux	011111	D			A	B	0101110111		0
sthx	011111	S			A	B	110010111		0
orc*x*	011111	S			A	B	0110011100		Rc
ecowx [3]	011111	S			A	B	0110110110		0
sthux	011111	S			A	B	110110111		0
or*x*	011111	S			A	B	0110111100		Rc
divwu*x*	011111	D			A	B	OE	1 1100 1011	Rc
mtspr[4]	011111	S			spr		0111010011		0
dcbi [2]	011111	000_00			A	B	0111010110		0
nand*x*	011111	S			A	B	0111011100		Rc
stvxl [1]	011111	**v**S			A	B	0111100111		0
divw*x*	011111	D			A	B	OE	1 1110 1011	Rc
mcrxr	011111	crfD		00	00_000	0_0000	1000000000		0
lswx [6]	011111	D			A	B	1000010101		0

Table A-4. Instructions by Primary and Secondary Opcode (Bin) (continued)

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
lwbrx	011111	D	A	B	1000010110	0
lfsx	011111	D	A	B	1000010111	0
srw*x*	011111	S	A	B	1000011000	Rc
tlbsync [2, 3]	011111	000_00	00_000	0_0000	1000110110	0
lfsux	011111	D	A	B	1000110111	0
mfsr[2]	011111	D	0 SR	0_0000	1001010011	0
lswi[6]	011111	D	A	NB	1001010101	0
sync	011111	000_00	00_000	0_0000	1001010110	0
lfdx	011111	D	A	B	1001010111	0
lfdux	011111	D	A	B	1001110111	0
mfsrin [2]	011111	D	00_000	B	1010010011	0
stswx [6]	011111	S	A	B	1010010101	0
stwbrx	011111	S	A	B	1010010110	0
stfsx	011111	S	A	B	1010010111	0
stfsux	011111	S	A	B	1010110111	0
stswi [6]	011111	S	A	NB	1011010101	0
stfdx	011111	S	A	B	1011010111	0
dcba [3]	011111	000_00	A	B	1011110110	0
stfdux	011111	S	A	B	1011110111	0
lhbrx	011111	D	A	B	1100010110	0
sraw*x*	011111	S	A	B	1100011000	Rc
dss [1]	011111	A 00 STRM	00_000	0_0000	1100110110	0
dssall [1]	011111	A 00 STRM	00_000	0_0000	1100110110	0
srawi*x*	011111	S	A	SH	1100011000	Rc
eieio	011111	000_00	00_000	0_0000	1101010110	0
sthbrx	011111	S	A	B	1110010110	0
extsh*x*	011111	S	A	0_0000	1110011010	Rc
extsb*x*	011111	S	A	0_0000	1110111010	Rc
tlbld [2, 3]	011111	000_00	00_000	B	1111010010	0
icbi	011111	000_00	A	B	1111010110	0
stfiwx [3]	011111	S	A	B	1111010111	0
tlbli [2, 3]	011111	000_00	00_000	B	1111110010	0
dcbz	011111	000_00	A	B	1111110110	0

Table A-4. Instructions by Primary and Secondary Opcode (Bin) (continued)

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
lwz	100000	D	A	d			
lwzu	100001	D	A	d			
lbz	100010	D	A	d			
lbzu	100011	D	A	d			
stw	100100	S	A	d			
stwu	100101	S	A	d			
stb	100110	S	A	d			
stbu	100111	S	A	d			
lhz	101000	D	A	d			
lhzu	101001	D	A	d			
lha	101010	D	A	d			
lhau	101011	D	A	d			
sth	101100	S	A	d			
sthu	101101	S	A	d			
lmw [6]	101110	D	A	d			
stmw [6]	101111	S	A	d			
lfs	110000	D	A	d			
lfsu	110001	D	A	d			
lfd	110010	D	A	d			
lfdu	110011	D	A	d			
stfs	110100	S	A	d			
stfsu	110101	S	A	d			
stfd	110110	S	A	d			
stfdu	110111	S	A	d			
fdivs_x_	111011	D	A	B	0000_0	1 0010	Rc
fsubs_x_	111011	D	A	B	0000_0	1 0100	Rc
fadds_x_	111011	D	A	B	0000_0	1 0101	Rc
fsqrts_x_ [5]	111011	D	00_000	B	0000_0	1 0110	Rc
fres_x_ [3]	111011	D	00_000	B	0000_0	1 1000	Rc
fmuls_x_	111011	D	A	0_0000	C	1 1001	Rc
fmsubs_x_	111011	D	A	B	C	1 1100	Rc
fmadds_x_	111011	D	A	B	C	1 1101	Rc
fnmsubs_x_	111011	D	A	B	C	1 1110	Rc

Table A-4. Instructions by Primary and Secondary Opcode (Bin) (continued)

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
fnmadds*x*	111011	D			A		B		C	1 1111	Rc
fcmpu	111111	crfD		00	A		B		0000000000		0
frsp*x*	111111	D			00_000		B		0000001100		Rc
fctiw*x*	111111	D			00_000		B		0000001110		Rc
fctiwz*x*	111111	D			00_000		B		0000001111		Rc
fdiv*x*	111111	D			A		B		0000_0	1 0010	Rc
fsub*x*	111111	D			A		B		0000_0	1 0100	Rc
fadd*x*	111111	D			A		B		0000_0	1 0101	Rc
fsqrt*x* [5]	111111	D			00_000		B		0000_0	1 0110	Rc
fsel*x* [3]	111111	D			A		B		C	1 0111	Rc
fmul*x*	111111	D			A		0_0000		C	1 1001	Rc
frsqrte*x* [3]	111111	D			00_000		B		0000_0	1 1010	Rc
fmsub*x*	111111	D			A		B		C	1 1100	Rc
fmadd*x*	111111	D			A		B		C	1 1101	Rc
fnmsub*x*	111111	D			A		B		C	1 1110	Rc
fnmadd*x*	111111	D			A		B		C	1 1111	Rc
fcmpo	111111	crfD		00	A		B		0000100000		0
mtfsb1*x*	111111	crbD			00_000		0_0000		0000100110		Rc
fneg*x*	111111	D			00_000		B		0000101000		Rc
mcrfs	111111	crfD		00	crfS	00	0_0000		001000000		0
mtfsb0*x*	111111	crbD			00_000		0_0000		0001000110		Rc
fmr*x*	111111	D			00_000		B		0001001000		Rc
mtfsfi*x*	111111	crfD		00	00_000		IMM	0	0010000110		Rc
fnabs*x*	111111	D			00_000		B		0010001000		Rc
fabs*x*	111111	D			00_000		B		0100001000		Rc
mffs*x*	111111	D			00_000		0_0000		1001000111		Rc
mtfsf*x*	111111	0	FM			0	B		1011000111		Rc

[1] AltiVec technology-specific instruction.

[2] Supervisor-level instruction.

[3] Optional to the PowerPC architecture but implemented by the MPC7450.

[4] Supervisor- and user-level instruction.

[5] Optional instruction not implemented by the MPC7450.

[6] Load/store string/multiple instruction.

A.5 Instructions Grouped by Functional Categories

Table A-5 through Table A-45 list the MPC7450 instructions grouped by function.

Key: ▨ Reserved bits

Table A-5. 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
divw_x_	31	D	A	B	OE	491	Rc
divwu_x_	31	D	A	B	OE	459	Rc
mulhw_x_	31	D	A	B	0	75	Rc
mulhwu_x_	31	D	A	B	0	11	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	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-6. Integer Compare 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
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-7. Integer 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
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		
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
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		
vand[1]	04	**v**D	**v**A	**v**B	1028	0
vandc [1]	04	**v**D	**v**A	**v**B	1092	0
vnor [1]	04	**v**D	**v**A	**v**B	1284	
vor [1]	04	**v**D	**v**A	**v**B	1156	
vxor [1]	04	D	A	B	1220	
xor_x_	31	S	A	B	316	Rc
xori	26	S	A	UIMM		
xoris	27	S	A	UIMM		

[1] AltiVec technology-specific instruction.

Table A-8. Integer Rotate 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_	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-9. Integer Shift 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
slw_x_	31	S	A	B	24		Rc
sraw_x_	31	S	A	B	792		Rc
srawi_x_	31	S	A	SH	824		Rc
srw_x_	31	S	A	B	536		Rc

Table A-10. Floating-Point 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
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_[1]	59	D	0 0 0 0 0	B	0 0 0 0 0	24	Rc
frsqrte_x_[1]	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_	63	D	A	B	C	23	Rc
fsqrt_x_[2]	63	D	0 0 0 0 0	B	0 0 0 0 0	22	Rc
fsqrts_x_[2]	59	D	0 0 0 0 0	B	0 0 0 0 0	22	Rc
vaddfp[3]	04	**v**D	**v**A	**v**B	10		0
vmaxfp[3]	04	**v**D	**v**A	**v**B	1034		
vminfp[3]	04	**v**D	**v**A	**v**B	1098		
vsubfp[3]	04	**v**D	**v**A	**v**B	74		

[1] Optional to the PowerPC architecture but implemented by the MPC7450.

[2] Optional instruction not implemented by the MPC7450.

[3] AltiVec technology-specific instruction.

Table A-11. Floating-Point Multiply-Add 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
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
vmaddfp[1]	04	**v**D	**v**A	**v**B	**v**C	46	
vnmsubfp[1]	04	**v**D	**v**A	**v**B	**v**C	47	

[1] AltiVec technology-specific instruction.

Table A-12. Floating-Point Rounding and Conversion 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
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
vcfs*x*[1]	04	**v**D	UIMM	**v**B	842	
vcfu*x*[1]	04	**v**D	UIMM	**v**B	778	0
vctsxs[1]	04	**v**D	UIMM	**v**B	970	
vctuxs[1]	04	**v**D	UIMM	**v**B	906	
vrfim[1]	04	**v**D	0 0 0 0 0	**v**B	714	
vrfin[1]	04	**v**D	0 0 0 0 0	**v**B	522	
vrfip[1]	04	**v**D	0 0 0 0 0	**v**B	650	
vrfiz[1]	04	**v**D	0 0 0 0 0	**v**B	586	

[1] AltiVec technology-specific instruction.

Table A-13. Floating-Point Compare 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
fcmpo	63	crfD	0 0	A	B	32		0
fcmpu	63	crfD	0 0	A	B	0		0
vcmpbfp*x*[1]	04	**v**D		**v**A	**v**B	Rc	966	
vcmpeqfp*x*[1]	04	**v**D		**v**A	**v**B	Rc	198	
vcmpgefp*x*[1]	04	**v**D		**v**A	**v**B	Rc	454	
vcmpgtfp*x*[1]	04	**v**D		**v**A	**v**B	Rc	710	

[1] AltiVec technology-specific instruction.

Table A-14. Floating-Point Status and Control Register 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
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-15. 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
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
lvebx[1]	31	**v**D	A	B	7	0
lvehx[1]	31	**v**D	A	B	39	0

Table A-15. Integer Load Instructions (continued)

lvewx [1]	31	vD	A	B	71	0
lvx [1]	31	vD	A	B	103	0
lvxl [1]	31	vD	A	B	359	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

[1] AltiVec technology-specific instruction.

Table A-16. 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
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-17. 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-18. 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[1]	46	D	A	d		
stmw[1]	47	S	A	d		

[1] Load/store string/multiple instruction.

Table A-19. 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[1]	31	D	A	NB	597	0
lswx[1]	31	D	A	B	533	0
stswi[1]	31	S	A	NB	725	0
stswx[1]	31	S	A	B	661	0

[1] Load/store string/multiple instruction.

Table A-20. 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
lwarx	31	D	A	B	20	0
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-21. Floating-Point 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
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-22. Floating-Point 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
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[1]	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

[1] Optional to the PowerPC architecture but implemented by the MPC7450.

Table A-23. Floating-Point Move 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
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-24. 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-25. 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
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-26. 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

[1] Supervisor-level instruction.

Table A-27. 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
tw	31	TO	A	B	4	0
twi	03	TO	A	SIMM		

Table A-28. 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

[1] Supervisor-level instruction

[2] Supervisor- and user-level instruction

Table A-29. 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
dcba[1]	31	0 0 0 0 0	A	B	758	0
dcbf	31	0 0 0 0 0	A	B	86	0
dcbi[2]	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

[1] Optional to the PowerPC but implemented by the MPC7450.

[2] Supervisor-level instruction.

Table A-30. 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

[1] Supervisor-level instruction

Table A-31. 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
tlbia[1]	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	370	0
tlbie[2,3]	31	0 0 0 0 0	0 0 0 0 0	B	306	0
tlbld [2, 3]	31	0 0 0 0 0	0 0 0 0 0	B	978	0
tlbli [2, 3]	31	0 0 0 0 0	0 0 0 0 0	B	1010	0
tlbsync [2, 3]	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	566	0

[1] Optional instruction not implemented by the MPC7450.

[2] Optional to the PowerPC architecture but implemented by the MPC7450.

[3] Supervisor-level instruction.

Table A-32. External Control Instructions

Name	0–5	6–10	11–15	16–20	21–30	31
eciwx[1]	31	D	A	B	310	0
ecowx[1]	31	S	A	B	438	0

[1] Optional to the PowerPC architecture but implemented by the MPC7450.

Table A-33. Vector Integer Arithmetic Instructions

Name	0–5	6–10	11–15	16–20	21–25	26–30	31
vaddcuw[1]	04	**v**D	**v**A	**v**B	384		0
vaddsbs[1]	04	**v**D	**v**A	**v**B	768		0
vaddshs[1]	04	**v**D	**v**A	**v**B	832		0
vaddsws[1]	04	**v**D	**v**A	**v**B	896		0
vaddubm[1]	04	**v**D	**v**A	**v**B	0		0
vaddubs[1]	04	**v**D	**v**A	**v**B	512		0
vadduhm[1]	04	**v**D	**v**A	**v**B	64		0
vadduhs[1]	04	**v**D	**v**A	**v**B	576		0
vadduwm[1]	04	**v**D	**v**A	**v**B	128		0
vadduws[1]	04	**v**D	**v**A	**v**B	640		0
vavgsb[1]	04	**v**D	**v**A	**v**B	1282		0
vavgsh[1]	04	**v**D	**v**A	**v**B	1346		0
vavgsw[1]	04	**v**D	**v**A	**v**B	1410		0
vavgub[1]	04	**v**D	**v**A	**v**B	1026		0
vavguh[1]	04	**v**D	**v**A	**v**B	1090		0
vavguw[1]	04	**v**D	**v**A	**v**B	1154		0
vmaxsb[1]	04	**v**D	**v**A	**v**B	258		
vmaxsh[1]	04	**v**D	**v**A	**v**B	322		
vmaxsw[1]	04	**v**D	**v**A	**v**B	386		
vmaxub[1]	04	**v**D	**v**A	**v**B	2		
vmaxuh[1]	04	**v**D	**v**A	**v**B	66		
vmaxuw[1]	04	**v**D	**v**A	**v**B	130		
vmhaddshs[1]	04	**v**D	**v**A	**v**B	**v**C	32	
vmhraddshs[1]	04	**v**D	**v**A	**v**B	**v**C	33	
vminsb[1]	04	**v**D	**v**A	**v**B	770		
vminsh[1]	04	**v**D	**v**A	**v**B	834		
vminsw[1]	04	**v**D	**v**A	**v**B	898		

Table A-33. Vector Integer Arithmetic Instructions (continued)

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
vminub[1]	04	vD	vA	vB	514	
vminuh[1]	04	vD	vA	vB	578	
vminuw[1]	04	vD	vA	vB	642	
vmladduhm[1]	04	vD	vA	vB	vC	34
vmsummbm[1]	04	vD	vA	vB	vC	37
vmsumshm[1]	04	vD	vA	vB	vC	40
vmsumshs[1]	04	vD	vA	vB	vC	41
vmsumubm[1]	04	vD	vA	vB	vC	36
vmsumuhm[1]	04	vD	vA	vB	vC	38
vmsumuhs[1]	04	vD	vA	vB	vC	39
vmulesb[1]	04	vD	vA	vB	776	
vmulesh[1]	04	vD	vA	vB	840	
vmuleub[1]	04	vD	vA	vB	520	
vmuleuh[1]	04	vD	vA	vB	584	
vmulosb[1]	04	vD	vA	vB	264	
vmulosh[1]	04	vD	vA	vB	328	
vmuloub[1]	04	vD	vA	vB	8	
vmulouh[1]	04	vD	vA	vB	72	
vsubcuw[1]	04	vD	vA	vB	1408	
vsubsbs[1]	04	vD	vA	vB	1792	
vsubshs[1]	04	vD	vA	vB	1856	
vsubsws[1]	04	vD	vA	vB	1920	
vsububm[1]	04	vD	vA	vB	1024	
vsububs[1]	04	vD	vA	vB	1536	
vsubuhm[1]	04	vD	vA	vB	1088	
vsubuhs[1]	04	vD	vA	vB	1600	
vsubuwm[1]	04	vD	vA	vB	1152	
vsubuws[1]	04	vD	vA	vB	1664	
vsumsws[1]	04	vD	vA	vB	1928	
vsum2sws[1]	04	D	A	B	1672	
vsum4sbs[1]	04	D	A	B	1800	
vsum4shs[1]	04	D	A	B	1608	
vsum4ubs[1]	04	D	A	B	1544	

[1] AltiVec technology-specific instruction

Table A-34. Floating-Point Compare 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
vcmpbfp*x*	04	vD	vA	vB	Rc	966
vcmpeqfp*x*	04	vD	vA	vB	Rc	198
vcmpgefp*x*	04	vD	vA	vB	Rc	454
vcmpgtfp*x*	04	vD	vA	vB	Rc	710

Table A-35. Floating-Point Estimate 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
vexptefp	04	vD	0 0 0 0 0	vB	394
vlogefp	04	vD	0 0 0 0 0	vB	458
vrefp	04	vD	0 0 0 0 0	vB	266
vrsqrtefp	04	vD	0 0 0 0 0	vB	330

Table A-36. Vector Load Instructions Supporting Alignment

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
lvsl	31	vD	A	B	6	0
lvsr	31	vD	A	B	38	0

Table A-37. 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
stvebx	31	S	A	B	135	0
stvehx	31	S	A	B	167	0
stvewx	31	S	A	B	199	0
stvx	31	S	A	B	231	0
stvxl	31	S	A	B	487	0

Table A-38. Vector Pack 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
vpkpx	04	vD	vA	vB	782
vpkshss	04	vD	vA	vB	398
vpkshus	04	vD	vA	vB	270
vpkswss	04	vD	vA	vB	462
vpkswus	04	vD	vA	vB	334
vpkuhum	04	vD	vA	vB	14
vpkuhus	04	vD	vA	vB	142

Table A-38. Vector Pack Instructions (continued)

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
vpkuwum	04	vD	vA	vB	78
vpkuwus	04	vD	vA	vB	206

Table A-39. Vector Unpack 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
vmrghb	04	vD	vA	vB	12
vmrghh	04	vD	vA	vB	76
vmrghw	04	vD	vA	vB	140
vmrglb	04	vD	vA	vB	268
vmrglh	04	vD	vA	vB	332
vupkhpx	04	D	0 0 0 0 0	B	846
vupkhsb	04	D	0 0 0 0 0	B	526
vupkhsh	04	D	0 0 0 0 0	B	590
vupklpx	04	D	0 0 0 0 0	B	974
vupklsb	04	D	0 0 0 0 0	B	654
vupklsh	04	D	0 0 0 0 0	B	718

Table A-40. Vector Splat 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
vspltb	04	vD	UIMM	vB	524
vsplth	04	vD	UIMM	vB	588
vspltisb	04	vD	SIMM	0 0 0 0 0	780
vspltish	04	vD	SIMM	0 0 0 0 0	844
vspltisw	04	vD	SIMM	0 0 0 0 0	908
vspltw	04	vD	UIMM	vB	652

Table A-41. Vector Permute 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
vperm	04	vD	vA	vB	vC	43

Table A-42. Vector Select 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
vsel	04	vD	vA	vB	vC	42

Table A-43. Vector Shift 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
vsl	04	**v**D	**v**A	**v**B	452		
vsldoi	04	**v**D	**v**A	**v**B	0	SH	44
vslo	04	**v**D	**v**A	**v**B	1036		
vsro	04	**v**D	**v**A	**v**B	1100		

Table A-44. Move To/From Condition Register 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
mfvscr	04	**v**D	0 0 0 0 0	0 0 0 0 0	1540	0
mtvscr	04	0 0 0 0 0	0 0 0 0 0	**v**B	1604	

Table A-45. User-Level Cache 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
dss	31	A	0 0	STRM	0 0 0 0 0	0 0 0 0 0	822	0
dssall	31	A	0 0	STRM	0 0 0 0 0	0 0 0 0 0	822	0
dst	31	T	0 0	STRM	A	B	342	0
dstst	31	T	0 0	STRM	A	B	374	0
dststt	31	1	0 0	STRM	A	B	374	0
dstt	31	1	0 0	STRM	A	B	342	0

A.6 Instructions Sorted by Form

Table A-46 through Table A-59 list the MPC7450 instructions grouped by form.

Key:

▨ Reserved bits

Table A-46. 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-47. 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-48. 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 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 0	1	0

Table A-49. 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

Table A-49. D-Form (continued)

Specific Instructions

Name	0–5	6–8	9	10	11–15	16–31
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	50	D			A	d
lfdu	51	D			A	d
lfs	48	D			A	d
lfsu	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[1]	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	54	S			A	d
stfdu	55	S			A	d
stfs	52	S			A	d
stfsu	53	S			A	d
sth	44	S			A	d

Table A-49. D-Form (continued)

Specific Instructions

Name	0–5	6–10	11–15	16–31
sthu	45	S	A	d
stmw[1]	47	S	A	d
stw	36	S	A	d
stwu	37	S	A	d
subfic	08	D	A	SIMM
twi	03	TO	A	SIMM
xori	26	S	A	UIMM
xoris	27	S	A	UIMM

[1] Load/store string/multiple instruction

Table A-50. 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

Table A-50. X-Form (continued)

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
OPCD	**v**D			**v**A	**v**B	XO	0
OPCD	**v**S			**v**A	**v**B	XO	0
OPCD	T	0 0	STRM	A	B	XO	0

Table A-50. X-Form

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
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
cntlzw*x*	31	S				A	0 0 0 0 0	26	Rc
dcba[1]	31	0 0 0 0 0				A	B	758	0
dcbf	31	0 0 0 0 0				A	B	86	0
dcbi[2]	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
dst	31	T	0 0	STRM		A	B	342	0
dstt[3]	31	1	0 0	STRM		A	B	342	0
dstst[3]	31	T	0 0	STRM		A	B	374	0
dststt[3]	31	1	0 0	STRM		A	B	374	0
dss[3]	31	A	0 0	STRM		0 0 0 0 0	0 0 0 0 0	822	0
dssall[3]	31	A	0 0	STRM		0 0 0 0 0	0 0 0 0 0	822	0
eciwx[1]	31	D				A	B	310	0
ecowx[1]	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

Table A-50. X-Form (continued)

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
extsb*x*	31	S		A	0 0 0 0 0	954	Rc
extsh*x*	31	S		A	0 0 0 0 0	922	Rc
fabs*x*	63	D		0 0 0 0 0	B	264	Rc
fcmpo	63	crfD	0 0	A	B	32	0
fcmpu	63	crfD	0 0	A	B	0	0
fctiw*x*	63	D		0 0 0 0 0	B	14	Rc
fctiwz*x*	63	D		0 0 0 0 0	B	15	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
frsp*x*	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
lfdux	31	D		A	B	631	0
lfdx	31	D		A	B	599	0
lfsux	31	D		A	B	567	0
lfsx	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[4]	31	D		A	NB	597	0
lswx[4]	31	D		A	B	533	0
lvebx[3]	31	**v**D		**v**A	**v**B	7	0
lvehx[3]	31	**v**D		A	B	39	0
lvewx[3]	31	**v**D		A	B	71	0
lvsl[3]	31	**v**D		A	B	6	0
lvsr[3]	31	**v**D		A	B	38	0

Table A-50. X-Form (continued)

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
lvx [3]	31	vD		A			B		103	0
lvxl [3]	31	vD		A			B		359	0
lwarx	31	D		A			B		20	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_	63	D		0 0 0 0 0			0 0 0 0 0		583	Rc
mfmsr [2]	31	D		0 0 0 0 0			0 0 0 0 0		83	0
mfsr [2]	31	D		0	SR		0 0 0 0 0		595	0
mfsrin [2]	31	D		0 0 0 0 0			B		659	0
mtfsb0_x_	63	crbD		0 0 0 0 0			0 0 0 0 0		70	Rc
mtfsb1_x_	63	crfD		0 0 0 0 0			0 0 0 0 0		38	Rc
mtfsfi_x_	63	crbD	0 0	0 0 0 0 0			IMM	0	134	Rc
mtmsr [2]	31	S		0 0 0 0 0			0 0 0 0 0		146	0
mtsr[2]	31	S		0	SR		0 0 0 0 0		210	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
slw_x_	31	S		A			B		24	Rc
sraw_x_	31	S		A			B		792	Rc
srawi_x_	31	S		A			SH		824	Rc
srw_x_	31	S		A			B		536	Rc
stbux	31	S		A			B		247	0
stbx	31	S		A			B		215	0
stfdux	31	S		A			B		759	0
stfdx	31	S		A			B		727	0
stfiwx [1]	31	S		A			B		983	0

Table A-50. X-Form (continued)

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
stfsux	31	S	A	B	695	0
stfsx	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[4]	31	S	A	NB	725	0
stswx [4]	31	S	A	B	661	0
stvebx [3]	31	vS	A	B	135	0
stvehx [3]	31	vS	A	B	167	0
stvewx [3]	31	vS	A	B	199	0
stvx [3]	31	vS	A	B	231	0
stvxl [3]	31	vS	A	B	487	0
stwbrx [3]	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
tlbia [5]	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	370	0
tlbie [2]	31	0 0 0 0 0	0 0 0 0 0	B	306	0
tlbld [1, 2]	31	0 0 0 0 0	0 0 0 0 0	B	978	0
tlbli [1, 2]	31	0 0 0 0 0	0 0 0 0 0	B	1010	0
tlbsync [2]	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

[1] Optional to the PowerPC architecture but implemented by the MPC7450

[2] Supervisor-level instruction

[3] Altivec technology-specific instruction

[4] Load/store string/multiple instruction

[5] Optional instruction not implemented by the MPC7450

Table A-51. 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
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

[1] Supervisor-level instruction

Table A-52. 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 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
mfspr[1]	31	D	spr			339	0
mftb	31	D	tbr			371	0
mtcrf	31	S	0	CRM	0	144	0
mtspr[1]	31	D	spr			467	0

[1] Supervisor- and user-level instruction

Table A-53. XFL-Form

OPCD	0	FM	0	B	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
mtfsf*x*	63	0	FM	0	B	711	Rc

Table A-54. 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 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
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
divw*x*	31	D	A	B	OE	491	Rc
divwu*x*	31	D	A	B	OE	459	Rc
mulhw*x*	31	D	A	B	0	75	Rc
mulhwu*x*	31	D	A	B	0	11	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-55. 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*	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
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
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
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
fres*x* [1]	59	D	0 0 0 0 0	B	0 0 0 0 0	24	Rc
frsqrte*x* [1]	63	D	0 0 0 0 0	B	0 0 0 0 0	26	Rc
fsel*x* [1]	63	D	A	B	C	23	Rc
fsqrt*x* [2]	63	D	0 0 0 0 0	B	0 0 0 0 0	22	Rc
fsqrts*x* [2]	59	D	0 0 0 0 0	B	0 0 0 0 0	22	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

[1] Optional to the PowerPC architecture but implemented by the MPC7450

[2] Optional instruction not implemented by the MPC7450

Table A-56. 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-57. VA-Form

OPCD	**v**D	**v**A	**v**B	**v**C		XO
OPCD	**v**D	**v**A	**v**B	0	SH	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
vmhaddshs[1]	04	**v**D	**v**A	**v**B	**v**C		32
vmhraddshs [1]	04	**v**D	**v**A	**v**B	**v**C		33
vmladduhm [1]	04	**v**D	**v**A	**v**B	**v**C		34
vmsumubm [1]	04	**v**D	**v**A	**v**B	**v**C		36
vmsummbm [1]	04	**v**D	**v**A	**v**B	**v**C		37
vmsumuhm [1]	04	**v**D	**v**A	**v**B	**v**C		38
vmsumuhs [1]	04	**v**D	**v**A	**v**B	**v**C		39
vmsumshm [1]	04	**v**D	**v**A	**v**B	**v**C		40
vmsumshs [1]	04	**v**D	**v**A	**v**B	**v**C		41
vsel [1]	04	**v**D	**v**A	**v**B	**v**C		42
vperm [1]	04	**v**D	**v**A	**v**B	**v**C		43
vsldoi [1]	04	**v**D	**v**A	**v**B	0	SH	44
vmaddfp [1]	04	**v**D	**v**A	**v**B	**v**C		46
vnmsubfp [1]	04	**v**D	**v**A	**v**B	**v**C		47

[1] AltiVec technology-specific instruction

Table A-58. VX-Form

OPCD	**v**D	**v**A	**v**B	XO	
OPCD	**v**D	0 0 0 0 0	0 0 0 0 0	XO	0
OPCD	0 0 0 0 0	0 0 0 0 0	**v**B	XO	0
OPCD	**v**D	0 0 0 0 0	**v**B	XO	
OPCD	**v**D	UIMM	**v**B	XO	
OPCD	**v**D	SIMM	0 0 0 0 0	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
vaddubm[1]	04	**v**D	**v**A	**v**B	0
vadduhm[1]	04	**v**D	**v**A	**v**B	64
vadduwm[1]	04	**v**D	**v**A	**v**B	128
vaddcuw[1]	04	**v**D	**v**A	**v**B	384
vaddubs[1]	04	**v**D	**v**A	**v**B	512
vadduhs[1]	04	**v**D	**v**A	**v**B	576
vadduws[1]	04	**v**D	**v**A	**v**B	640
vaddsbs[1]	04	**v**D	**v**A	**v**B	768
vaddshs[1]	04	**v**D	**v**A	**v**B	832
vaddsws[1]	04	**v**D	**v**A	**v**B	896
vsububm[1]	04	**v**D	**v**A	**v**B	1024
vsubuhm[1]	04	**v**D	**v**A	**v**B	1088
vsubuwm[1]	04	**v**D	**v**A	**v**B	1152
vsubcuw[1]	04	**v**D	**v**A	**v**B	1408
vsububs[1]	04	**v**D	**v**A	**v**B	1536
vsubuhs[1]	04	**v**D	**v**A	**v**B	1600
vsubuws[1]	04	**v**D	**v**A	**v**B	1664
vsubsbs[1]	04	**v**D	**v**A	**v**B	1792
vsubshs[1]	04	**v**D	**v**A	**v**B	1856
vsubsws[1]	04	**v**D	**v**A	**v**B	1920
vmaxub[1]	04	**v**D	**v**A	**v**B	2
vmaxuh[1]	04	**v**D	**v**A	**v**B	66
vmaxuw[1]	04	**v**D	**v**A	**v**B	130
vmaxsb[1]	04	**v**D	**v**A	**v**B	258
vmaxsh[1]	04	**v**D	**v**A	**v**B	322

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
vmaxsw [1]	04	vD	vA	vB	386	
vminub [1]	04	vD	vA	vB	514	
vminuh [1]	04	vD	vA	vB	578	
vminuw [1]	04	vD	vA	vB	642	
vminsb [1]	04	vD	vA	vB	770	
vminsh [1]	04	vD	vA	vB	834	
vminsw [1]	04	vD	vA	vB	898	
vavgub [1]	04	vD	vA	vB	1026	
vavguh [1]	04	vD	vA	vB	1090	
vavguw [1]	04	vD	vA	vB	1154	
vavgsb [1]	04	vD	vA	vB	1282	
vavgsh [1]	04	vD	vA	vB	1346	
vavgsw [1]	04	vD	vA	vB	1410	
vrlb [1]	04	vD	vA	vB	4	
vrlh [1]	04	vD	vA	vB	68	
vrlw [1]	04	vD	vA	vB	132	
vslb [1]	04	vD	vA	vB	260	
vslh [1]	04	vD	vA	vB	324	
vslw [1]	04	vD	vA	vB	388	
vsl [1]	04	vD	vA	vB	452	
vsrb [1]	04	vD	vA	vB	516	
vsrh [1]	04	vD	vA	vB	580	
vsrw [1]	04	vD	vA	vB	644	
vsr [1]	04	vD	vA	vB	708	
vsrab [1]	04	vD	vA	vB	772	
vsrah [1]	04	vD	vA	vB	836	
vsraw [1]	04	vD	vA	vB	900	
vand [1]	04	vD	vA	vB	1028	
vandc [1]	04	vD	vA	vB	1092	
vor [1]	04	vD	vA	vB	1156	
vnor [1]	04	vD	vA	vB	1284	
mfvscr [1]	04	vD	0 0 0 0 0	0 0 0 0 0	1540	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
mtvscr[1]	04	0 0 0 0 0	0 0 0 0 0	vB	1604	0
vmuloub[1]	04	vD	vA	vB	8	
vmulouh[1]	04	vD	vA	vB	72	
vmulosb[1]	04	vD	vA	vB	264	
vmulosh[1]	04	vD	vA	vB	328	
vmuleub[1]	04	vD	vA	vB	520	
vmuleuh[1]	04	vD	vA	vB	584	
vmulesb[1]	04	vD	vA	vB	776	
vmulesh[1]	04	vD	vA	vB	840	
vsum4ubs[1]	04	vD	vA	vB	1544	
vsum4sbs[1]	04	vD	vA	vB	1800	
vsum4shs[1]	04	vD	vA	vB	1608	
vsum2sws[1]	04	vD	vA	vB	1672	
vsumsws[1]	04	vD	vA	vB	1928	
vaddfp[1]	04	vD	vA	vB	10	
vsubfp[1]	04	vD	vA	vB	74	
vrefp[1]	04	vD	0 0 0 0 0	vB	266	
vrsqrtefp[1]	04	vD	0 0 0 0 0	vB	330	
vexptefp[1]	04	vD	0 0 0 0 0	vB	394	
vlogefp[1]	04	vD	0 0 0 0 0	vB	458	
vrfin[1]	04	vD	0 0 0 0 0	vB	522	
vrfiz[1]	04	vD	0 0 0 0 0	vB	586	
vrfip[1]	04	vD	0 0 0 0 0	vB	650	
vrfim[1]	04	vD	0 0 0 0 0	vB	714	
vcfux[1]	04	vD	UIMM	vB	778	
vcfsx[1]	04	vD	UIMM	vB	842	
vctuxs[1]	04	vD	UIMM	vB	906	
vctsxs[1]	04	vD	UIMM	vB	970	
vmaxf p[1]	04	vD	vA	vB	1034	
vminfp[1]	04	vD	vA	vB	1098	
vmrghb[1]	04	vD	vA	vB	12	
vmrghh[1]	04	vD	vA	vB	76	

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
vmrghw [1]	04	vD	vA	vB	140	
vmrglb [1]	04	vD	vA	vB	268	
vmrglh [1]	04	vD	vA	vB	332	
vmrglw [1]	04	vD	vA	vB	396	
vspltb [1]	04	vD	UIMM	vB	524	
vsplth [1]	04	vD	UIMM	vB	588	
vspltw [1]	04	vD	UIMM	vB	652	
vspltisb [1]	04	vD	SIMM	0 0 0 0 0	780	
vspltish [1]	04	vD	SIMM	0 0 0 0 0	844	
vspltisw [1]	04	vD	SIMM	0 0 0 0 0	908	
vslo [1]	04	vD	vA	vB	1036	
vsro [1]	04	vD	vA	vB	1100	
vpkuhum [1]	04	vD	vA	vB	14	
vpkuwum [1]	04	vD	vA	vB	78	
vpkuhus [1]	04	vD	vA	vB	142	
vpkuwus [1]	04	vD	vA	vB	206	
vpkshus [1]	04	vD	vA	vB	270	
vpkswus [1]	04	vD	vA	vB	334	
vpkshs s[1]	04	vD	vA	vB	398	
vpkswss [1]	04	vD	vA	vB	462	
vpkswus [1]	04	vD	vA	vB	334	
vupkhsb [1]	04	vD	0 0 0 0 0	vB	526	
vupkhsh [1]	04	vD	0 0 0 0 0	vB	590	
vupklsb [1]	04	vD	0 0 0 0 0	vB	654	
vupklsh [1]	04	vD	0 0 0 0 0	vB	718	
vpkpx [1]	04	vD	vA	vB	12	782
vupkhpx [1]	04	vD	0 0 0 0 0	vB	846	
vupklpx [1]	04	vD	0 0 0 0 0	vB	974	
vxor [1]	04	vD	vA	vB	1220	

[1] AltiVec technology-specific instruction

Table A-59. VXR-Form

OPCD	vD	vA	vB	Rc	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
vcmpbfp*x*[1]	04	vD	vA	vB	Rc	966
vcmpeqfp*x*[1]	04	vD	vA	vB	Rc	198
vcmpequb*x*[1]	04	vD	vA	vB	Rc	6
vcmpequh*x*[1]	04	vD	vA	vB	Rc	70
vcmpequw*x*[1]	04	vD	vA	vB	Rc	134
vcmpgefp*x*[1]	04	vD	vA	vB	Rc	454
vcmpgtfp*x*[1]	04	vD	vA	vB	Rc	710
vcmpgtsb*x*[1]	04	vD	vA	vB	Rc	774
vcmpgtsh*x*[1]	04	vD	vA	vB	Rc	838
vcmpgtsw*x*[1]	04	vD	vA	vB	Rc	902
vcmpgtub*x*[1]	04	vD	vA	vB	Rc	518
vcmpgtuh*x*[1]	04	vD	vA	vB	Rc	582
vcmpgtuw*x*[1]	04	vD	vA	vB	Rc	646

[1] AltiVec technology-specific instruction

A.7 Instruction Set Legend

Table A-60 provides general information on the PowerPC instruction set (such as the architectural level, privilege level, and form).

Table A-60. PowerPC Instruction Set Legend

Name	UISA	VEA	OEA	Supervisor Level	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

Table A-60. PowerPC Instruction Set Legend (continued)

Name	UISA	VEA	OEA	Supervisor Level	Optional	Form
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
cntlzw*x*	√					X
crand	√					XL
crandc	√					XL
creqv	√					XL
crnand	√					XL
crnor	√					XL
cror	√					XL
crorc	√					XL
crxor	√					XL
dcba		√			√	X
dcbf		√				X
dcbi			√	√		X
dcbst		√				X
dcbt		√				X
dcbtst		√				X
dcbz		√				X
divw*x*	√					XO
divwu*x*	√					XO
eciwx		√			√	X

Table A-60. PowerPC Instruction Set Legend (continued)

Name	UISA	VEA	OEA	Supervisor Level	Optional	Form
ecowx		√			√	X
eieio		√				X
eqv*x*	√					X
extsb*x*	√					X
extsh*x*	√					X
fabs*x*	√					X
fadd*x*	√					A
fadds*x*	√					A
fcmpo	√					X
fcmpu	√					X
fctiw*x*	√					X
fctiwz*x*	√					X
fdiv*x*	√					A
fdivs*x*	√					A
fmadd*x*	√					A
fmadds*x*	√					A
fmr*x*	√					X
fmsub*x*	√					A
fmsubs*x*	√					A
fmul*x*	√					A
fmuls*x*	√					A
fnabs*x*	√					X
fneg*x*	√					X
fnmadd*x*	√					A
fnmadds*x*	√					A
fnmsub*x*	√					A
fnmsubs*x*	√					A
fres*x*	√				√	A
frsp*x*	√					X
frsqrte*x*	√				√	A
fsel*x*	√				√	A

Table A-60. PowerPC Instruction Set Legend (continued)

Name	UISA	VEA	OEA	Supervisor Level	Optional	Form
fsqrt*x*	√				√	A
fsqrts*x*	√				√	A
fsub*x*	√					A
fsubs*x*	√					A
icbi		√				X
isync		√				XL
lbz	√					D
lbzu	√					D
lbzux	√					X
lbzx	√					X
lfd	√					D
lfdu	√					D
lfdux	√					X
lfdx	√					X
lfs	√					D
lfsu	√					D
lfsux	√					X
lfsx	√					X
lha	√					D
lhau	√					D
lhaux	√					X
lhax	√					X
lhbrx	√					X
lhz	√					D
lhzu	√					D
lhzux	√					X
lhzx	√					X
lmw [2]	√					D
lswi [2]	√					X
lswx [2]	√					X
lwarx	√					X

Table A-60. PowerPC Instruction Set Legend (continued)

Name	UISA	VEA	OEA	Supervisor Level	Optional	Form
lwbrx	√					X
lwz	√					D
lwzu	√					D
lwzux	√					X
lwzx	√					X
mcrf	√					XL
mcrfs	√					X
mcrxr	√					X
mfcr	√					X
mffs*x*	√					X
mfmsr			√	√		X
mfspr [1]	√		√	√		XFX
mfsr			√	√		X
mfsrin			√	√		X
mftb		√				XFX
mtcrf	√					XFX
mtfsb0*x*	√					X
mtfsb1*x*	√					X
mtfsf*x*	√					XFL
mtfsfi*x*	√					X
mtmsr			√	√		X
mtspr [1]	√		√	√		XFX
mtsr			√	√		X
mtsrin			√	√		X
mulhw*x*	√					XO
mulhwu*x*	√					XO
mulli	√					D
mullw*x*	√					XO
nand*x*	√					X
neg*x*	√					XO
nor*x*	√					X

Table A-60. PowerPC Instruction Set Legend (continued)

Name	UISA	VEA	OEA	Supervisor Level	Optional	Form
or*x*	√					X
orc*x*	√					X
ori	√					D
oris	√					D
rfi			√	√		XL
rlwimi*x*	√					M
rlwinm*x*	√					M
rlwnm*x*	√					M
sc	√		√			SC
slw*x*	√					X
sraw*x*	√					X
srawi*x*	√					X
srw*x*	√					X
stb	√					D
stbu	√					D
stbux	√					X
stbx	√					X
stfd	√					D
stfdu	√					D
stfdux	√					X
stfdx	√					X
stfiwx	√					X
stfs	√					D
stfsu	√					D
stfsux	√					X
stfsx	√					X
sth	√					D
sthbrx	√					X
sthu	√					D
sthux	√					X
sthx	√					X

Table A-60. PowerPC Instruction Set Legend (continued)

Name	UISA	VEA	OEA	Supervisor Level	Optional	Form
stmw [2]	√					D
stswi [2]	√					X
stswx [2]	√					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
tlbia			√	√	√	X
tlbie			√	√	√	X
tlbsync			√	√		X
tw	√					X
twi	√					D
xor*x*	√					X
xori	√					D
xoris	√					D

Notes:

[1] Supervisor- and user-level instruction.

[2] Load/store string or multiple instruction.

[3] Optional instruction provided to support temporary 64-bit bridge.

[4] Defined for the 32-bit architecture and by the temporary 64-bit bridge.

Appendix B
Instructions Not Implemented

This appendix provides a list of the 32-bit instructions that are not implemented in the MPC7450 microprocessor. Note that an attempt to execute instructions that are not implemented on the MPC7450 generates 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 and not implemented by the MPC7450.

Table B-1. 32-Bit Instructions Not Implemented by the MPC7450

Mnemonic	Instruction
fsqrt*x*	Floating Square Root (Double-Precision)
fsqrts*x*	Floating Square Root Single
tlbia	Translation Lookaside Buffer Invalidate All

Appendix C
Special-Purpose Registers

Table C-1. PowerPC SPR Encodings Ordered by Decimal Value

Register Name	SPR[1]			Access	mfspr/mtspr
	Decimal	spr[5–9]	spr[0–4]		
XER	1	00000	00001	User (UISA)	Both
LR	8	00000	01000	User (UISA)	Both
CTR	9	00000	01001	User (UISA)	Both
DSISR	18	00000	10010	Supervisor (OEA)	Both
DAR	19	00000	10011	Supervisor (OEA)	Both
DEC	22	00000	10110	Supervisor (OEA)	Both
SDR1	25	00000	11001	Supervisor (OEA)	Both
SRR0	26	00000	11010	Supervisor (OEA)	Both
SRR1	27	00000	11011	Supervisor (OEA)	Both
VRSAVE [2]	256	01000	00000	User (AltiVec/UISA)	Both
TBL [3]	268	01000	01100	User (VEA)	**mfspr**, **mftb**
TBU [3]	269	01000	01101	User (VEA)	**mfspr**, **mftb**
SPRG0	272	01000	10000	Supervisor (OEA)	Both
SPRG1	273	01000	10001	Supervisor (OEA)	Both
SPRG2	274	01000	10010	Supervisor (OEA)	Both
SPRG3	275	01000	10011	Supervisor (OEA)	Both
SPRG4 [4]	276	01000	10100	Supervisor (OEA)	Both
SPRG5 [4]	277	01000	10101	Supervisor (OEA)	Both
SPRG6 [4]	278	01000	100110	Supervisor (OEA)	Both
SPRG7 [4]	279	01000	10111	Supervisor (OEA)	Both
EAR [5]	282	01000	11010	Supervisor (OEA)	Both
TBL [3]	284	01000	11100	Supervisor (OEA)	**mtspr**
TBU [3]	285	01000	11101	Supervisor (OEA)	**mtspr**
SVR [8]	286	01000	11110	Supervisor (OEA)	**mfspr**
PVR	287	01000	11111	Supervisor (OEA)	**mfspr**

Table C-1. PowerPC SPR Encodings Ordered by Decimal Value (continued)

Register Name	SPR[1]			Access	mfspr/mtspr
	Decimal	spr[5–9]	spr[0–4]		
IBAT0U	528	10000	10000	Supervisor (OEA)	Both
IBAT0L	529	10000	10001	Supervisor (OEA)	Both
IBAT1U	530	10000	10010	Supervisor (OEA)	Both
IBAT1L	531	10000	10011	Supervisor (OEA)	Both
IBAT2U	532	10000	10100	Supervisor (OEA)	Both
IBAT2L	533	10000	10101	Supervisor (OEA)	Both
IBAT3U	534	10000	10110	Supervisor (OEA)	Both
IBAT3L	535	10000	10111	Supervisor (OEA)	Both
DBAT0U	536	10000	11000	Supervisor (OEA)	Both
DBAT0L	537	10000	11001	Supervisor (OEA)	Both
DBAT1U	538	10000	11010	Supervisor (OEA)	Both
DBAT1L	539	10000	11011	Supervisor (OEA)	Both
DBAT2U	540	10000	11100	Supervisor (OEA)	Both
DBAT2L	541	10000	11101	Supervisor (OEA)	Both
DBAT3U	542	10000	11110	Supervisor (OEA)	Both
DBAT3L	543	10000	11111	Supervisor (OEA)	Both
IBAT4U [4]	560	10001	10000	Supervisor (OEA)	Both
IBAT4L [4]	561	10001	10001	Supervisor (OEA)	Both
IBAT5U [4]	562	10001	10010	Supervisor (OEA)	Both
IBAT5L [4]	563	10001	10011	Supervisor (OEA)	Both
IBAT6U [4]	564	10001	10100	Supervisor (OEA)	Both
IBAT6L [4]	565	10001	10101	Supervisor (OEA)	Both
IBAT7U [4]	566	10001	10110	Supervisor (OEA)	Both
IBAT7L [4]	567	10001	10111	Supervisor (OEA)	Both
DBAT4U [4]	568	10001	11000	Supervisor (OEA)	Both
DBAT4L [4]	569	10001	11001	Supervisor (OEA)	Both
DBAT5U [4]	570	10001	11010	Supervisor (OEA)	Both
DBAT5L [4]	571	10001	11011	Supervisor (OEA)	Both
DBAT6U [4]	572	10001	11100	Supervisor (OEA)	Both
DBAT6L [4]	573	10001	11101	Supervisor (OEA)	Both
DBAT7U [4]	574	10001	11110	Supervisor (OEA)	Both

Table C-1. PowerPC SPR Encodings Ordered by Decimal Value (continued)

Register Name	SPR[1]			Access	mfspr/mtspr
	Decimal	spr[5–9]	spr[0–4]		
DBAT7L[4]	575	10001	11111	Supervisor (OEA)	Both
UMMCR2[6]	928	11101	00000	User (UISA)	**mfspr**
UPMC5[6]	929	11101	00001	User (UISA)	**mfspr**
UPMC6[6]	930	11101	00010	User (UISA)	**mfspr**
UMMCR0[6]	936	11101	01000	User (UISA)	**mfspr**
UPMC1[6]	937	11101	01001	User (UISA)	**mfspr**
UPMC2[6]	938	11101	01010	User (UISA)	**mfspr**
USIAR[6]	939	11101	01011	User (UISA)	**mfspr**
UMMCR1[6]	940	11101	01100	User (UISA)	**mfspr**
UPMC3[6]	941	11101	01101	User (UISA)	**mfspr**
UPMC4[6]	942	11101	01110	User (UISA)	**mfspr**
MMCR2[6]	944	11101	10000	Supervisor (OEA)	Both
PMC5[5]	945	11101	10001	Supervisor (OEA)	Both
PMC6 [5]	946	11101	10010	Supervisor (OEA)	Both
BAMR[6]	951	11101	10111	Supervisor (OEA)	Both
MMCR0[5]	952	11101	11000	Supervisor (OEA)	Both
PMC1[5]	953	11101	11001	Supervisor (OEA)	Both
PMC2 [5]	954	11101	11010	Supervisor (OEA)	Both
SIAR [5]	955	11101	11011	Supervisor (OEA)	Both
MMCR1 [5]	956	11101	11100	Supervisor (OEA)	Both
PMC3 [5]	957	11101	11101	Supervisor (OEA)	Both
PMC4 [5]	958	11101	11110	Supervisor (OEA)	Both
TLBMISS [3]	980	11110	10100	Supervisor (OEA)	Both
PTEHI [6]	981	11110	10101	Supervisor (OEA)	Both
PTELO [6]	982	11110	10110	Supervisor (OEA)	Both
L3PM [7]	983	11110	10111	Supervisor (OEA)	Both
L3ITCR0 [7]	984	11110	11000	Supervisor (OEA)	Both
L2ERRINJHI [8]	985	11110	11001	Supervisor (OEA)	Both
L2ERRINJLO [8]	986	11110	11010	Supervisor (OEA)	Both
L2ERRINJCTL [8]	987	11110	11011	Supervisor (OEA)	Both
L2CAPTDATAHI [8]	988	11110	11100	Supervisor (OEA)	**mfspr**

Table C-1. PowerPC SPR Encodings Ordered by Decimal Value (continued)

Register Name	SPR[1]			Access	mfspr/mtspr
	Decimal	spr[5–9]	spr[0–4]		
L2CAPTDATALO [8]	989	11110	11101	Supervisor (OEA)	**mfspr**
L2CAPTECC [8]	990	11110	11110	Supervisor (OEA)	**mfspr**
L2ERRDET [8]	991	11110	11111	Supervisor (OEA)	Special [8]
L2ERRDIS [8]	992	11111	00000	Supervisor (OEA)	Both
L2ERRINTEN [8]	993	11111	00001	Supervisor (OEA)	Both
L2ERRATTR [8]	994	11111	00010	Supervisor (OEA)	Both
L2ERRADDR [8]	995	11111	00011	Supervisor (OEA)	**mfspr**
L2ERREADDR [8]	996	11111	00100	Supervisor (OEA)	**mfspr**
L2ERRCTL [8]	997	11111	00101	Supervisor (OEA)	Both
L3OHCR [9]	1000	11111	01000	Supervisor (OEA)	Both
L3ITCR1 [9]	1001	11111	01001	Supervisor (OEA)	Both
L3ITCR2 [9]	1002	11111	01010	Supervisor (OEA)	Both
L3ITCR3 [9]	1003	11111	01011	Supervisor (OEA)	Both
HID0 [6]	1008	11111	10000	Supervisor (OEA)	Both
HID1	1009	11111	10001	Supervisor (OEA)	Both
IABR [6]	1010	11111	10010	Supervisor (OEA)	Both
ICTRL [6]	1011	11111	10011	Supervisor (OEA)	Both
DABR [5]	1013	11111	10101	Supervisor (OEA)	Both
MSSCR0 [6]	1014	11111	10110	Supervisor (OEA)	Both
MSSSR0	1015	11111	10111	Supervisor (OEA)	Both
LDSTCR [6]	1016	11111	11000	Supervisor (OEA)	Both
L2CR	1017	11111	11001	Supervisor (OEA)	Both
L3CR [7]	1018	11111	11010	Supervisor (OEA)	Both
ICTC [6]	1019	11111	11011	Supervisor (OEA)	Both
PIR [5]	1023	11111	11111	Supervisor (OEA)	Both

[1] 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.

[2] Register defined by the AltiVec technology.

[3] The TB registers are referred to as TBRs rather than SPRs and can be written to using the **mtspr** instruction in supervisor mode and the TBR numbers here. The TB registers can be read in user mode using either the **mftb** or **mtspr** instruction and specifying TBR 268 for TBL and TBR 269 for TBU**.**

[4] MPC7445-, MPC7447-, MPC7455- and MPC7457-specific only, register may not be supported on other processors that implement the PowerPC architecture.

[5] Register defined as optional in the PowerPC architecture.

[6] MPC7441-, MPC7445-, MPC7447-, MPC7451-, MPC7455-, and MPC7457-specific register may not be supported on other processors that implement the PowerPC architecture.

[7] MPC7451-, MPC7455-, and MPC7457-specific register, not supported on the MPC7441, MPC7445, and MPC7457.

[8] Most bits are bit reset/write 1 clear. A write of 0 to a bit does not change it. A write of 1 to a bit clears it. Reads act normally.

[9] MPC7457-specific register, not supported on the MPC7441, MPC7445, MPC7447, MPC7451, and MPC7455

Table C-2. PowerPC SPR Encodings Ordered by Register Name

Register Name	SPR[1]			Access	mfspr/mtspr
	Decimal	spr[5–9]	spr[0–4]		
BAMR[2]	951	11101	10111	Supervisor (OEA)	Both
CTR	9	00000	01001	User (UISA)	Both
DABR [3]	1013	11111	10101	Supervisor (OEA)	Both
DAR	19	00000	10011	Supervisor (OEA)	Both
DBAT0L	537	10000	11001	Supervisor (OEA)	Both
DBAT0U	536	10000	11000	Supervisor (OEA)	Both
DBAT1L	539	10000	11011	Supervisor (OEA)	Both
DBAT1U	538	10000	11010	Supervisor (OEA)	Both
DBAT2L	541	10000	11101	Supervisor (OEA)	Both
DBAT2U	540	10000	11100	Supervisor (OEA)	Both
DBAT3L	543	10000	11111	Supervisor (OEA)	Both
DBAT3U	542	10000	11110	Supervisor (OEA)	Both
DBAT4L[4]	569	10001	11001	Supervisor (OEA)	Both
DBAT4U [4]	568	10001	11000	Supervisor (OEA)	Both
DBAT5L [4]	571	10001	11011	Supervisor (OEA)	Both
DBAT5U [4]	570	10001	11010	Supervisor (OEA)	Both
DBAT6L [4]	573	10001	11101	Supervisor (OEA)	Both
DBAT6U [4]	572	10001	11100	Supervisor (OEA)	Both
DBAT7L [4]	575	10001	11111	Supervisor (OEA)	Both
DBAT7U [4]	574	10001	11110	Supervisor (OEA)	Both
DEC	22	00000	10110	Supervisor (OEA)	Both
DSISR	18	00000	10010	Supervisor (OEA)	Both
EAR [3]	282	01000	11010	Supervisor (OEA)	Both
HID0 [2]	1008	11111	10000	Supervisor (OEA)	Both

Table C-2. PowerPC SPR Encodings Ordered by Register Name (continued)

Register Name	SPR[1]			Access	mfspr/mtspr
	Decimal	spr[5–9]	spr[0–4]		
HID1	1009	11111	10001	Supervisor (OEA)	Both
IABR [2]	1010	11111	10010	Supervisor (OEA)	Both
IBAT0L	529	10000	10001	Supervisor (OEA)	Both
IBAT0U	528	10000	10000	Supervisor (OEA)	Both
IBAT1L	531	10000	10011	Supervisor (OEA)	Both
IBAT1U	530	10000	10010	Supervisor (OEA)	Both
IBAT2L	533	10000	10101	Supervisor (OEA)	Both
IBAT2U	532	10000	10100	Supervisor (OEA)	Both
IBAT3L	535	10000	10111	Supervisor (OEA)	Both
IBAT3U	534	10000	10110	Supervisor (OEA)	Both
IBAT4L [4]	561	10001	10001	Supervisor (OEA)	Both
IBAT4U [4]	560	10001	10000	Supervisor (OEA)	Both
IBAT5L [4]	563	10001	10011	Supervisor (OEA)	Both
IBAT5U [v]	562	10001	10010	Supervisor (OEA)	Both
IBAT6L [4]	565	10001	10101	Supervisor (OEA)	Both
IBAT6U [4]	564	10001	10100	Supervisor (OEA)	Both
IBAT7L [4]	567	10001	10111	Supervisor (OEA)	Both
IBAT7U [4]	566	10001	10110	Supervisor (OEA)	Both
ICTC [2]	1019	11111	11011	Supervisor (OEA)	Both
ICTRL [2]	1011	11111	10011	Supervisor (OEA)	Both
L2CAPTECC [1]	990	11110	11110	Supervisor (OEA)	**mfspr**
L2CAPTDATAHI [1]	988	11110	11100	Supervisor (OEA)	**mfspr**
L2CAPTDATALO [1]	989	11110	11101	Supervisor (OEA)	**mfspr**
L2CR	1017	11111	11001	Supervisor (OEA)	Both
L2ERRADDR [1]	995	11111	00011	Supervisor (OEA)	**mfspr**
L2ERRATTR [1]	994	11111	00010	Supervisor (OEA)	Both
L2ERRCTL [1]	997	11111	00101	Supervisor (OEA)	Both
L2ERRDET [1]	991	11110	11111	Supervisor (OEA)	Special [5]
L2ERRDIS [1]	992	11111	00000	Supervisor (OEA)	Both
L2ERREADDR [1]	996	11111	00100	Supervisor (OEA)	**mfspr**
L2ERRINJCTL [1]	987	11110	11011	Supervisor (OEA)	Both

Table C-2. PowerPC SPR Encodings Ordered by Register Name (continued)

Register Name	SPR[1]			Access	mfspr/mtspr
	Decimal	spr[5–9]	spr[0–4]		
L2ERRINJH [1]I	985	11110	11001	Supervisor (OEA)	Both
L2ERRINJLO [1]	986	11110	11010	Supervisor (OEA)	Both
L2ERRINTEN [1]	993	11111	00001	Supervisor (OEA)	Both
L3CR[6]	1018	11111	11010	Supervisor (OEA)	Both
L3ITCR0 [6]	1000	11111	01000	Supervisor (OEA)	Both
L3ITCR1 [6]	1001	11111	01001	Supervisor (OEA)	Both
L3ITCR2 [6]	1002	11111	01010	Supervisor (OEA)	Both
L3ITCR3 [6]	1003	11111	01011	Supervisor (OEA)	Both
L3OHCR [7]	984	11110	11000	Supervisor (OEA)	Both
L3PM [6]	983	11110	10111	Supervisor (OEA)	Both
LDSTCR [2]	1016	11111	11000	Supervisor (OEA)	Both
LR	8	00000	01000	User (UISA)	Both
MMCR0 [3]	952	11101	11000	Supervisor (OEA)	Both
MMCR1 [3]	956	11101	11100	Supervisor (OEA)	Both
MMCR2 [2]	944	11101	10000	Supervisor (OEA)	Both
MSSCR0 [2]	1014	11111	10110	Supervisor (OEA)	Both
MSSSR0	1015	11111	10111	Supervisor (OEA)	Both
PIR [3]	1023	11111	11111	Supervisor (OEA)	Both
PMC1 [3]	953	11101	11001	Supervisor (OEA)	Both
PMC2 [3]	954	11101	11010	Supervisor (OEA)	Both
PMC3 [3]	957	11101	11101	Supervisor (OEA)	Both
PMC4 [3]	958	11101	11110	Supervisor (OEA)	Both
PMC5 [3]	945	11101	10001	Supervisor (OEA)	Both
PMC6 [3]	946	11101	10010	Supervisor (OEA)	Both
PTEHI [2]	981	11110	10101	Supervisor (OEA)	Both
PTELO [2]	982	11110	10110	Supervisor (OEA)	Both
PVR	287	01000	11111	Supervisor (OEA)	**mfspr**
SDR1	25	00000	11001	Supervisor (OEA)	Both
SIAR [3]	955	11101	11011	Supervisor (OEA)	Both
SPRG0	272	01000	10000	Supervisor (OEA)	Both
SPRG1	273	01000	10001	Supervisor (OEA)	Both

Table C-2. PowerPC SPR Encodings Ordered by Register Name (continued)

Register Name	SPR[1]			Access	mfspr/mtspr
	Decimal	spr[5–9]	spr[0–4]		
SPRG2	274	01000	10010	Supervisor (OEA)	Both
SPRG3	275	01000	10011	Supervisor (OEA)	Both
SPRG4 [4]	276	01000	10100	Supervisor (OEA)	Both
SPRG5 [4]	277	01000	10101	Supervisor (OEA)	Both
SPRG6 [4]	278	01000	100110	Supervisor (OEA)	Both
SPRG7 [4]	279	01000	10111	Supervisor (OEA)	Both
SRR0	26	00000	11010	Supervisor (OEA)	Both
SRR1	27	00000	11011	Supervisor (OEA)	Both
SVR [1]	286	01000	11110	Supervisor (OEA)	**mfspr**
TBL[8]	284	01000	11100	Supervisor (OEA)	**mtspr**
TBL [8]	268	01000	01100	User (VEA)	**mfspr**, **mftb**
TBU [8]	285	01000	11101	Supervisor (OEA)	**mtspr**
TBU [8]	269	01000	01101	User (VEA)	**mfspr**, **mftb**
TLBMISS [2]	980	11110	10100	Supervisor (OEA)	Both
UMMCR0 [2]	936	11101	01000	User (UISA)	**mfspr**
UMMCR1 [2]	940	11101	01100	User (UISA)	**mfspr**
UMMCR2 [2]	928	11101	00000	User (UISA)	**mfspr**
UPMC1 [2]	937	11101	01001	User (UISA)	**mfspr**
UPMC2 [2]	938	11101	01010	User (UISA)	**mfspr**
UPMC3 [2]	941	11101	01101	User (UISA)	**mfspr**
UPMC4 [2]	942	11101	01110	User (UISA)	**mfspr**
UPMC5 [2]	929	11101	00001	User (UISA)	**mfspr**
UPMC6 [2]	930	11101	00010	User (UISA)	**mfspr**
USIAR [2]	939	11101	01011	User (UISA)	**mfspr**
VRSAVE [9]	256	01000	00000	User (AltiVec/UISA)	Both
XER	1	00000	00001	User (UISA)	Both

[1] 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.

[2] MPC7441-, MPC7445-, MPC7447-, MPC7451-, MPC7455-, and MPC7457-specific register may not be supported on other processors that implement the PowerPC architecture.

[3] Register defined as optional in the PowerPC architecture.

[4] MPC7445-, MPC7447-, MPC7455-, and MPC7457-specific only, register may not be supported on other processors that implement the PowerPC architecture.

[5] Most bits are bit reset/write 1 clear. A write of 0 to a bit does not change it. A write of 1 to a bit clears it. Reads act normally.

[6] MPC7451-, MPC7455-, and MPC7457-specific register, not supported on the MPC7441, MPC7445, and MPC7457.

[7] MPC7457-specific register, not supported on the MPC7441, MPC7445, MPC7447, MPC7451, and MPC7455

[8] The TB registers are referred to as TBRs rather than SPRs and can be written to using the **mtspr** instruction in supervisor mode and the TBR numbers here. The TB registers can be read in user mode using either the **mftb** or **mtspr** instruction and specifying TBR 268 for TBL and TBR 269 for TBU**.**

[9] Register defined by the AltiVec technology.

Appendix D
Revision History

This appendix provides a list of the major differences between revisions. Note that the list only covers the major changes to the Reference manual.

Changes to the *MPC7450 RISC Microprocessor Family Reference Manual* from Revision 4.2 to Revision 5 are as follows:

Section, Page	Change
1.1, 1-4	Replaced Figure 1-1 with revised MPC7450 Microprocessor block diagram.
1.1, 1-5	Replaced Figure 1-2 with revised MPC7448 Microprocessor block diagram.
1.5, 1-62	Footnote added to Table 1-5.
2.1.6, 2-65	Replaced the L2CR resource setting for the MPC7448 in Table 2-45 with: For the MPC7448, 0x3000_0000
3.1.3.2, 3-10	Replaced the second sentence in the second paragraph with: The L2 services accesses from the LLQ and LSQ with a 3-cycle total latency (in the MPC7448, 5 cycles with ECC disabled, 6 cycles with ECC enabled) and a maximum throughput of one L2 access per clock cycle (one access every other cycle in the MPC7448)
3.3.2.1.2, 3-18	Replaced the first paragraph with: In the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448 the allowance of different cache states between the L1 and L2 caches eliminates the need to allocate or update the state in the L2 when a transient (**dststt** or **stvxl**) store occurs to a block that is marked as shared in the L2. In this case, the LLQ treats the L2 block as invalid for stores.
3.4.4.2, 3-35	Replaced the first sentence of the second paragraphs with: From a programming point of view, it can be advantageous to **dcbtst** instructions on the e600 core if multiple line misses may occur that would otherwise be caused by store instructions.
4.6.1, 4-16	Corrected cross-reference in the third paragraph.
5.3.3, 5-31	Replaced the first sentence of the second paragraph with: In the MPC7445, MPC7447, MPC7447A, MPC7448, MPC7455, and MPC7457, Figure 5-15 shows translation of address bits when the extended block size is enabled (HID0[XBBSEN]=1) and extended addressing is enabled (HID0[XAEN] = 1).
5.5.5.1, 5-64	Replaced the footnote on Table 5-15 with:

Specific only to the MPC7445, MPC7447, MPC7447A, MPC7448, MPC7455, and MPC7457 registers.

6.7.3.2, 6-67	Replaced the first paragraph with: The four-entry vector issue queue (VIQ) handles all AltiVec computational instructions. Two instructions can dispatch to it per cycle, and it can issue two instructions in-order per cycle from its bottom two entries (VIQ1–VIQ0) if reservation stations are available. The MPC7448 can issue two instructions out-of-order from its bottom two entries, preventing an instruction in VIQ1 destined for VIU1 from waiting for an instruction in VIQ0 that is stalled behind an instruction waiting for operand availability. Moreover, the VIU2, VFPU, and VPU are pipelined, so they can operate on multiple instructions.
6.7.7, 6-82	Replaced the first paragraph with: The three-level cache implementation (two-level in the MPC7441, MPC7445, MPC7447, MPC7447A, and MPC7448) affects instruction fetching, the loading and storing source, and destination operands, as described in the following sections
6.7.7.1, 6-82	Replaced the first two sentences of the first paragraph with: For the MPC7450, the unified 256-Kbyte on-board L2 cache (512-Kbyte for MPC7457 and 1 Mbyte for the MPC7448) has 8-way set associativity and 64-byte lines (with two sectors/line). This implies 4096 lines (8192 for the MPC7447, MPC7457, and MPC7447A, 16384 for the MPC7448) and 512 sets (1024 for the MPC7447, MPC7457, and MPC7447A, 2048 for the MPC7448)
8.2.11.1, 8-26	Replaced the first sentence of the Timing Comments with: Assertion—Must not occur before the last cycle of the snoop response window (cycle after AACK).
8.2.11.2, 8-27	Added the following sentence to the Timing Comments: $\overline{\text{TEA}}$ must not be asserted before the last cycle of the snoop response window.
8.3.8.2	Replaced the Timing Comments with the following statement: Same as MPX bus interface.
8.4.1.1, 8-46	Add the following after the fourth sentence: The L3 address signals reflect the real address for various L3 cache sizes as shown in Table 8.7. Note that the L3 address does not correspond bit-for-bit with the real address.

Table D-1. L3 Address to Physical Address Bit Mapping

L3 Address Bit	Private Memory Size and Corresponding Physical Address Bit[1]		
	1M	2M	4M[2]
L3_ADDR[18][3]	—	PM[4]	17
L3_ADDR[17]	PM[5]	18	18

Table D-1. L3 Address to Physical Address Bit Mapping (continued)

L3 Address Bit	Private Memory Size and Corresponding Physical Address Bit[1]		
	1M	2M	4M[2]
L3_ADDR[16]	19	19	19
L3_ADDR[15]	20	20	20
L3_ADDR[14]	21	21	21
L3_ADDR[13]	22	22	22
L3_ADDR[12]	23	23	23
L3_ADDR[11]	24	24	24
L3_ADDR[10]	25	25	25
L3_ADDR[9]	26	26	26
L3_ADDR[8]	27	27	27
L3_ADDR[7]	28	28	28
L3_ADDR[6]	29	29	29
L3_ADDR[5]	30	30	30
L3_ADDR[4]	16 (Way 2)	15 (Way 2)	14
L3_ADDR[3]	17 (Way 1)	16 (Way 1)	15
L3_ADDR[2]	18 (Way 0)	17 (Way 0)	16
L3_ADDR[1]	31	31	31
L3_ADDR[0]	32	32	32

[1] Entries in parentheses indicate the function of these bits when the L3 is operated in cache mode. These bits reflect which of the eight ways is being accessed. When operating in private memory mode, the bits reflect the stated bits of the physical address.

[2] MPC7457-specific; not supported on other devices. Note that because the MPC7457 does not support 4M of L3 as cache, 4M of SRAM must be configured as either 4M of private memory or 2M of private memory and 2M of cache; see the *MPC7450 RISC Microprocessor Family Reference Manual* for more information.

[3] MPC7457-specific; not implemented on other devices.

[4] For MPC7457 only, the value of this bit depends on the size of the SRAM and the private memory space. For 4M of SRAM configured as 2M of cache and 2M of private memory, this bit will be driven low (0b0) for cache accesses and high (0b1) for private memory accesses. For 2M of SRAM configured as private memory (only), this bit is always driven low (0b0).

[5] The value of this bit depends on the size of the SRAM and the private memory space. For 2M of SRAM configured as 1M of cache and 1M of private memory, this bit will be driven low (0b0) for cache accesses and high (0b1) for private memory accesses. For 1M of SRAM configured as private memory (only), this bit is always driven low (0b0).

9.1.2, 9-4	Replaced Figure 9-1 with revised MPC7450 Microprocessor block diagram.
9.1.2, 9-5	Replaced Figure 9-2 with revised MPC7448 Microprocessor block diagram.
9.4.2.2, 9-35	Replace the first paragraph with the following:

	Because the MPC7450 has no means to retry a data tenure, data must never be transferred before the last cycle of the snoop response window. Do not issue a $\overline{\text{TA}}$ before an $\overline{\text{AACK}}$.
10.2.5, 10-5	Reorganized the Dynamic Frequency Switching section.

Major changes to the *MPC7450 RISC Microprocessor Family User's Manual* from Revision 3.1 to Revision 4.2 were to include the MPC7448 microprocessor in the MPC7450 family. For a summary of the changes, see Section 1.1.8, "MPC7448 Microprocessor Overview."

Other changes to the *MPC7450 RISC Microprocessor Family User's Manual* from Revision 3.1 to Revision 4.2 are as follows:

Section, Page	**Change**
1.5, 1-62	Footnote added to Table 1-5.
2.1.5.1, 2-18	Figure 2-8. modified to include the MPC7445, MPC7447, MPC7455, and MPC7457.
3.7.9.2, 3-90	Figure 3-24, "L3 Cache Configuration for Late-Write or PB2 SRAMs," has been updated.
8.2.7.2.1, 8-16	Transfer type RCLAIM; TT[0:4] = 0b0111 (0x7) changed to RCLAIM; TT[0:4] = 0b01111 (0xF).

Major changes from Revision 3 to Revision 3.1 were to include the MPC7447A microprocessor in the MPC7450 family. There are no microarchitectural differences between the MPC7447A and the MPC7447. The MPC7447A provides new functionality to reduce the power consumption on the microprocessor that includes:

- An additional bit to the HID1 register for Dynamic Frequency Switching (DFS),
- An internal temperature diode.

Other changes to the *MPC7450 RISC Microprocessor Family User's Manual* from Revision 3 to Revision 3.1 are as follows:

Section, Page	**Change**
5.3.1, 5-26	The last paragraph describing the extended addressing for BAT registers has been reworded. Also, the figures were reversed and are updates as follows:

The format and bit definitions of the upper and lower BAT registers for extended addressing are shown in Figure 11-8 and Figure 11-9, respectively. The upper BAT register format is the same as that for 32-bit addressing as shown in Figure 11-7. When using the MPC7445, MPC7447, MPC7455, or the MPC7457, the extended block length (XBL) for the BATs replaces BATU[15–18] reserved field, as shown in Figure 11-8. When extended addressing is used, the lower BAT contains the new BXPN and BX fields that comprise the extended physical page number.

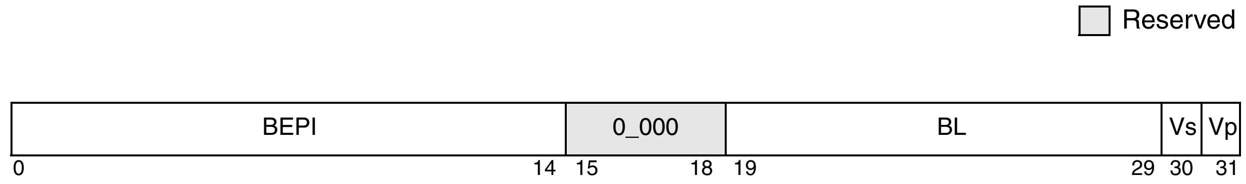


Figure 11-7. Format of Upper BAT Register (BATU)—Extended Addressing for the MPC7441 and the MPC7451

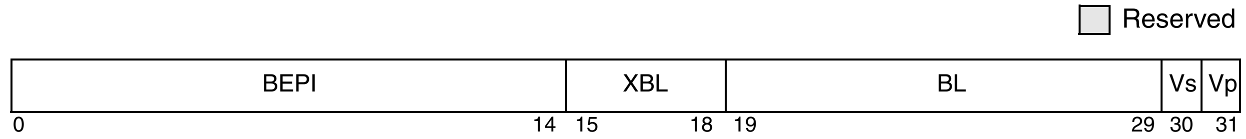
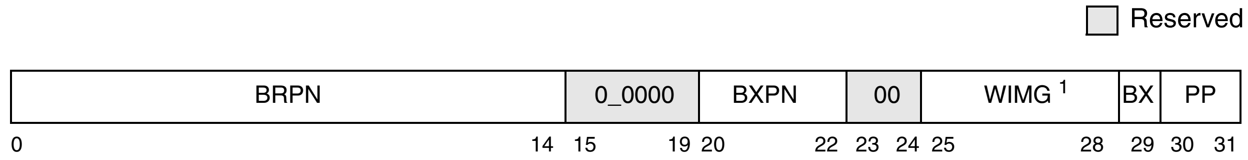


Figure 11-8. Format of Upper BAT Register (BATU)—Extended Block Size for the MPC7445, MPC7447, MPC7455, or the MPC7457



[1] W and G bits are not defined for IBAT registers. Attempting to write to these bits causes boundedly undefined results.

Figure 11-9. Format of Lower BAT Register (BATL)—Extended Addressing

Major changes to the *MPC7450 RISC Microprocessor Family User's Manual* from Revision 2 to Revision 3 are as follows:

Section, Page	**Change**
Through out the UM	Two new processors are described in Revision 3 of the MPC7450 RISC Microprocessor Family User's Manual, they are the MPC7447 and the MPC7457. Any differences from the MPC7450 are noted through-out the user's manual. Section 1.6, "Differences Between MPC7441/MPC7451 and MPC7447/MPC7457," describes the major differences.
1.1, 1-4	The MPC7450 block diagram was updated.
1.4, 1-56	In Table 1-4, MPC7450 and and MPC7400/MPC7410 Feature Comparison, the following entry under Execution Unit Timings (Latency Throughput) was updated to:

L1 miss, L2 hit latency	9—data access 13—instruction access	9 (11)[1]

[1] Numbers in parentheses are for 2:1 SRAM.

2.1.5.3, 2-25	MSSCR0[ABD] bit description has been added. Table 2-12 shows the updated bit description:

11	ABD	Address bus driven mode 0 Address bus driven mode disabled 1 Address bus driven mode enabled The read-only bit reflects the state of the $\overline{\text{BMODE0}}$ signal after $\overline{\text{HRSET}}$ negation and indicates whether the processor is address bus driven mode. See Section 9.3.2.1, "Address Bus Driven Mode," for more information.

2.1.5.5.2, 2-32 The L3CR[L3NIRCA] bit description has been reworded. Table 2-9 shows the updated bit description:

24	L3NIRCA	L3 non-integer ratios clock adjustment for the SRAM. When this bit is set, the AC timing of L3_CLK[0:1] is changed. 0 L3 SRAM clock timing is unchanged (default). 1 The L3_CLK[0:1] signals occur earlier relative to the MPC7450 driving the L3 address, control and data buses in non-integer L3 clock ratios. Because of the way that the L3_CLK[0:1] signals are internally derived, these signals may be driven slightly later (one-eight of a core clock) with non-integer clock ratios than they would normally be with an integer L3 clock ratio. This can potentially cause AC timing problems on the L3 interface if the timing margins are very small. This signal corrects for this phemomenon by causing the MPC7450 to drive the L3_CLK[0:1] signals one-quarter of a core clock earlier. See the hardware specifications for further clarification.

2.1.5.5.2, 2-29 Two new sections were added following the L3CR regisier description that describes the L3OHCR and L3ITCR registers.

2.3.2.4.1, 2-59 In Table 2-26, MSR[BE} and MSR[SE] bit synchronization requirements were added to the table.

3.1.2.3, 3-8 The following line is added to the end of the section:

If a cache line is full, it does not load, it just issues a kill block type transfer.

3.3.3.1, 3-28 The following table was added at the end of the end of the "Performed Loads and Store," section:

Table D-2. Load and Store Ordering with WIMG Bit Settings

W	I	M	G	Order[1], [2]
n	1	*n*	1	Stores are ordered with respect to other stores. Loads are ordered with respect to other loads. A store followed by a load requires an **eieio**.instruction in between the store and load.
1	0	*n*	1	Stores are ordered with respect to other stores. Loads are ordered with respect to other loads. A store followed by a load requires a **sync**.instruction in between the store and load.
1	*n*	*n*	0	Stores are ordered with respect to other stores. A load followed by a load requires a **sync**.instruction in betweenthe loads. A store followed by a load requires a **sync**.instruction in between the store and load.
0	0	1	*n*	A store followed by a store requires an **eieio**.instruction in between the stores. A load followed by a load requires a **sync**.instruction in betweenthe loads. A store followed by a load requires a **sync**.instruction in between the store and load.

Table D-2. Load and Store Ordering with WIMG Bit Settings

W	I	M	G	Order[1], [2]
0	0	0	*n*	A store followed by a store requires an **eieio**.instruction in between the stores. A load followed by a load requires a **sync**.instruction in betweenthe loads. A store followed by a load requires a **sync**.instruction in between the store and load.
0	1	*n*	0	A store followed by a store requires an **eieio**.instruction in between the stores. A load followed by a load requires a **sync**.instruction in betweenthe loads. A store followed by a load requires a **sync**.instruction in between the store and load.

[1] Any load followed by any store is always ordered for the MPC7450.

[2] A **sync** instruction will cover the synchronization cases that require an **eieio** instruction. However, an **eieio** instruction will not cover all the synchronization cases that require a **sync** instruction.

3.7.1., 3-66 Figure 3-19 has been updated and moved under the section, “MSUG2 DDR Interface Timing.”

3.7.3.5, 3-69 Table 3-24 was updated.

3.7.8, 3-78 A section on how to initialize the L3 cache for private memory has been added.

3.7.8, 3-79 In the second paragraph , L2PM[PMBA] should actually be L3PM[PMBA].

3.7.9.2, 3-83 Figure 3-23, “L3 Cache Configuration for Late-Write or PB2 SRAMs,” has been updated.

4.6.1, 4-18 The first line of the second paragraph has been changed to:

$\overline{\text{SRESET}}$ is an edge-sensitive signal that can be asserted and negated asynchronously, provided there are two bus cycles in between, see Section 8.4.3.4.1, “Soft Reset (SRESET)—Input,” for more details.

4.6.15 The TLBMISS entry in Table 4-14 was corrected to the following:

TLBMISS	0–30	Effective page address for the access that caused the TLB miss exception
	31	LRU Way

8.2.6.1, 8-12 The following sentence was added after the second sentence.

When driving A[0:3] as outputs they are driven as zero.

8.2.10.2.1, 8-25 State Meaning has changed to the following:

State Meaning Asserted/Negated—Represents odd parity for each of the eight bytes during data write transactions. Odd parity means that an odd number of bits, including the parity bit, are driven high. All eight parity bits are driven with valid parity on all bus operations. HID1[EBA] and HID1[EBD] control whether control whether the processor will check address and data parity respectively. The MPC7450 always generates parity regardless of whether checking is enable or disabled. The signal assignments are listed in Table 8-6.

Changes to the *MPC7450 RISC Microprocessor Family User's Manual* from Revision 2.0 to Revision 2.2 are as follows:

Section, Page	**Change**
2.1.5.2, 2-22	PLL_CFG[0:3] signals were changed to reflect the the additional PLL_CFG[4] signal. The name of the PLL signal was changed from PLL_EXT to PLL_CFG[4].
2.1.5.5.1, 2-27	L2CR[12] is updated from reserved to the L2CR[L3OH0] description.
2.1.5.5.2, 2-29	L3CR[12] is updated from reserved to the L3CR[L3OH1] description.
3.7.3.5,3-69	Table 3-25 signal assignments were updated.
Chapter 8, Signal Descriptions	
	The PLL_EXT signal was renamed to PLL_CFG[4]. Section 8.4.5.3 "PLL Extension (PLL_EXT)—Input," was deleted. The information was migrated to Section 8.4.5.2 "PLL Configuration (PLL_CFG[0:4])—Input."

Major changes to the *MPC7450 RISC Microprocessor Family User's Manual* from Revision 1 to Revision 2 are as follows:

Section, Page	**Change**
Through out the UM	Two new processor are described in Revision 2 of the MPC7450 RISC Microprocessor Family User's Manual, they are the MPC7445 and the MPC7455. Any differences from the MPC7450 are noted through-out the user's manual. Section 1.5, "Differences Between MPC7441/MPC7451 and MPC7445/MPC7455," describes the major differences.
Listed in Table D-3	The thermal assist unit (TAU) is no longer supported on the MPC7441, MPC7450, or MPC7451. The TAU is not supported on the MPC7445 and MPC7455 either. All references in the *MPC7450 RISC Microprocessor Family User's Manual* Revision 1 for the TAU are listed in Table D-3 and should be ignored.

Table D-3. TAU References

Section	Page No.
1.1	1-3
1.2.1	1-10—1-11
1.2.10	1-27
1.2.11	1-28
1.3.1	1-32 (Figure 1-6), 1-38 (Table 1-1)
1.3.4.2	1-44, 1-46 (Table 1-3)
2.1.1	2-3 (Figure 2-1)
2.1.2	2-7 (Table 2-1), 2-9 (Table 2-1)
2.1.3.3	2-13

Table D-3. TAU References (continued)

Section	Page No.
2.1.5.8	2-38—2-42
4.1	4-3
4.6.19	4-34—4-35
10.2	10-1—10-5
10.3	10-5—10-10
10.4	10-10
Appendix D	D-5

2.1.5.5.2, 2-28 & 2.1.5.5.2, 2-31 &	The name for bit 24 of L3CR has changed from L3CYA to L3NIRCA.

The major changes to the *MPC7450 RISC Microprocessor Family User's Manual*, from Revision 0 to Revision 1, are as follows:

Section, Page	**Change**
1.2.1, 1-5	Under the bullet: Separate on-chip L1 instruction and data caches (Harvard architecture), deleted the following: "Parity support on cache and tags" Feature lists for L1, L2, L3 cache were updated.
2.1.2, 2-7	In Table 2-1, replaced the SDA register description with the following:

SDAR, USDAR	—	Sampled data address register. The MPC7450 does not implement the optional registers (SDAR or the user-level, read-only USDAR register) defined by the PowerPC architecture. Note that in previous processors the SDA and USDA registers could be written to by boot code without causing an exception, this is not the case in the MPC7450. A **mtspr** or **mfspr** SDAR or USDAR instruction causes a program exception**.**	—

Also, deleted Section 2.1.5.9.12, "User-Sampled Instruction Address Register (USIAR)" on Page 2-40 as this is not relevant to the MPC7450.

2.1.5.1, 2-11	In Table 2-4, modified the following entries to:

16	ICE [6]	Instruction cache enable 0 The instruction cache is neither accessed nor updated. All pages are accessed as if they were marked cache-inhibited (WIM = x1x). Potential cache accesses from the bus (snoop and cache operations) are ignored. In the disabled state for the L1 caches, the cache tag state bits are ignored and all accesses are propagated to the L2 cache, L3 cache, or bus as burst transactions. For those transactions, $\overline{\text{CI}}$ is asserted regardless of address translation. ICE is zero at power-up. 1 The instruction cache is enabled. Note that HID0[ICFI] must be set at the same time that this bit is set.
17	DCE [2]	Data cache enable 0 The data cache is neither accessed nor updated. All pages are accessed as if they were marked cache-inhibited (WIM = x1x). Potential cache accesses from the bus (snoop and cache operations) are ignored. In the disabled state for the L1 caches, the cache tag state bits are ignored and all accesses are propagated to the L2 cache, L3 cache, or bus as cache-inhibited. For those transactions, $\overline{\text{CI}}$ is asserted regardless of address translation.DCE is zero at power-up. 1 The data cache is enabled.Note that HID0[DCFI] must be set at the same time that this bit is set.

[6] A context synchronizing instruction must immediately follow a mtspr. A mtspr instruction for HID0 should not modify either of these bits at the same time it modifies another bit that requires additional synchronization.

[2] A dssall and sync must precede a mtspr and then a sync and context synchronizing instruction must follow. Note that if a user is not using the AltiVec data streaming instructions, then a dssall is not necessary prior to accessing the HID0{DCE] or HID0[DCFI] bit.

2.1.5.2, 2-14	In Table 2-5, the last sentence of bit 7’s description should be modified. That is, delete the sentence: “In some bus modes this bit is ignored.”
2.1.5.3, 2-16	In Table 2-7, replaced the DTQ bit value for “7 entries” to “111.”
2.1.5.5.2, 2-21	In Table 2-10, replaced the L3CR[5] description with the following:
2.1.5.5.5, 2-26	In Table 2-13, replaced the L3PMADDR field description with the following:

5	—	Reserved. Must be set by software during initialization. See Section 3.7.3.1, “Enabling the L3 Cache and L3 Initialization,” for details on when to set this bit.

0–15	L3PMADDR	L3 base address of L3 private memory L3PMADDR contain the base address of the range of addresses used in the L3 private memory. Specific bits of the L3PM[L3PMADDR] field are used based on the memory size as follows: 1MB L3PM[0–15] 2MB L3PM[0–14]

2.1.5.7.1, 2-27	In Table 2-15, and in Section 5.5.5.1.1, Page 5-64, Table 5-16, replaced the PAGE field description with the following:

0–30	PAGE	Effective page address Stores EA[0–30] of the access that caused the TLB Miss exception.

2.3.2.4.1, 2-49 In Table 2-26, note that if a user is not using the AltiVec data streaming instructions, a **dssall** is not necessary prior to accessing the register.

3.3.3.3, 3-28 Added more information on load and store ordering in new User’s Manual, the new sections are:

– Section 3.3.3.3, “Load Ordering with Respect to Other Loads,”

– Section 3.3.3.4, “Store Ordering with Respect to Other Stores

– Section 3.3.3.5, “Enforcing Store Ordering with Respect to Loads

Section, Page **Change**

3.8.2, 3-84 Added Table 3-30:

Table D-4. Bus Operations Caused by Cache Control Instructions (WIM = xx0)

Instruction	Current State		Next Cache State	Bus Operation	Comment
	Cache Coherency	HID1 Setting			
dcbt	M, E, S	—	No change	None	—
dcbt	I	—	E, S	Read	Fetched cache block is stored in the cache
dcbtst	M, E, S	—	No change	None	—
dcbtst	I	—	E, S	Read (60x mode) RCLAIM (MPX mode)	Fetched cache block is stored in the cache
dcbz	M, E, S, I	—	M	None	
dcba	M, E, S, I	—	M	None	
dcbf, dcbi	M	—	I	Write with kill	Block is pushed
dcbf, dcbi	E, S, I	—	I	None	—
dcbst	M	—	E, S, I	Write with kill	Block is pushed
dcbst	E, S, I	—	I	None	—
icbi	V, I	—	I	None	Instruction cache only

3.8.3, 3-86 Updated Table 3-29, and deleted the following entry:

Double-beat read (caching-inhibited or cache disabled)	PA[0:32] || 0b000	0 1 0 1 0	0	0 0 1	1	0	¬ M

3.8.4.3, 3-92 Following the fourth bullet item:

Because the MPC7450 only snoops global accesses ($\overline{\text{GBL}}$ asserted), that is assumed for all of the tables.

Added the following sentence:

“The MPC7450 will not issue a snoop response ($\overline{\text{ARTRY}}$ and $\overline{\text{HIT}}$) for transactions in which $\overline{\text{GBL}}$ is not asserted.”8.2.7.6, 8-17In the “State Meaning: Asserted” section it states:

Note that on the MPC750, $\overline{\text{WT}}$ assertion during a read operation indicates an instruction fetch. The MPC7450 also uses $\overline{\text{WT}}$ to indicate instruction fetches.

This is incorrect; replaced the sentence with the following:

“The MPC7450 negates $\overline{\text{WT}}$ for instruction fetches. Note that this is different from previous processors.”

Section, Page **Change**

8.4.4.4, 8-46 Modified entry in Table 8-7 to:

10	N/A[1]

[1] Not applicable; 1.5V is not supported for the system bus

9.3.2.4.1, 9-18 In Table 9-1, the description of **dcba** and **dcbz** states:

“store miss merge to 32 bytes”

This is incorrect and should be deleted.

In Table 9-1, added the following row:

Burst	Store miss	0	1	1	1	0	Read-with-intent-to-modify	Burst

10.3.1, 10-7 In Table 10-3, replaced the description of SITV with the following:

“Sample interval timer value. Number of elapsed system bus clock cycles before a junction temperature vs. threshold comparison result is sampled in order for TIN to be set and an interrupt to be generated. The value should be greater than 20 µs. This is necessary due to the thermal sensor, DAC, and the analog comparator settling time being greater than the bus cycle time.” 11.1, 11–2Deleted the following text:

Note that the MPC7450 does not implement the sampled data address register (SDAR) or the user-level, read-only USDAR defined by the architecture. However, for compatibility with processors that do, those registers can be written to by boot code without causing an exception.

Replaced the text with the following:

Note that in previous processors the optional SDAR and USDAR registers could be written to by boot code without causing an exception, this is not the case in the MPC7450. A **mtspr** or **mfspr** SDAR or USDAR instruction causes a program exception

11.5.1, 11-18 In Table 11-9 replaced the PMC1 events 65 and 68 with the following respectively:

65 (100_0001)	Floating-point store instructions completed in LSU	Counts aligned floating-point store instructions completed. All misaligned floating-point store instructions completed are counted under PMC1, event number 88 (0x101_1000).
68 (100_0100)	Floating-Point store causes stall in LSU	Counts cycles a floating-point store in the FSQ results in a store not being able to complete.

Section, Page **Change**

11.5.2, 11-18 In Table 11-10 replaced the PMC2 event 62 with the following:

62 (011_1110)	LSU completes floating-point store single	Counts aligned floating-point store single instructions completed. All misaligned floating-point store instructions completed are counted under PMC1, event number 88 (0x101_1000).

11.5.4, 11-28 In Table 11-12 replaced the PMC4 events 24 and 30 with the following respectively:

11.5.5, 11-30 In Table 11-13 replaced the PMC5 event 15 with the following:

24 (1_1000)	Snoop retries	Counts the number of load-store snoops that are retried by the load-store. This includes external snoops which are retried because of a load-store collision, as well as internal load-store self-snoop retries. It does not include snoops which are retried because of an MSS collision or busy condition. An example of an internal self-snoop collision is a load L1 miss which collides with a castout in the L1 castout queue. This type of collision is handled through internal snoop retry instead of load-store pipeline stall.
30 (1_1110)	Floating-point store double completes in LSU	Counts aligned floating-point store double instructions completed. All misaligned floating-point store instructions completed are counted under PMC1, event number 88 (0x101_1000).

15 (0_1111)	Snoop retries	Counts counts the number of internal requests that are internally retried. This includes load-store retries as well as some MSS collision cases (that would prevent an L2 hit from being considered good).

11.5.6, 11-31 In Table 11-14 replaced the PMC6 event 24 with the following:

24 (01_1000)	External snoop retry	Counts the number of external snoops that get a retry response.

Appendix, A-35 In Table A-9, the instructions **fsqrt***x* and **fsqrts***x* have footnotes 1 and 2. They now only have footnote 2 “Optional instruction not implemented by the MPC7450.”

Glossary

The glossary contains an alphabetical list of terms, phrases, and abbreviations used in this reference manual.

A

Architecture. A detailed specification of requirements for a processor or computer system. It does not specify details of how the processor or computer system must be implemented; instead it provides a template for a family of compatible *implementations*.

Atomic access. A bus access that attempts to be part of a read-write operation to the same address uninterrupted by any other access to that address (the term refers to the fact that the transactions are indivisible). The PowerPC architecture implements atomic accesses through the **lwarx**/**stwcx.** instruction pair.

Autobaud. The process of determining a serial data rate by timing the width of a single bit.

B

Beat. A single state on the bus interface that may extend across multiple bus cycles. A transaction can be composed of multiple address or data *beats*.

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*. See *Little endian*.

Boundedly undefined. A characteristic of certain operation results that are not rigidly prescribed by the PowerPC architecture. Boundedly-undefined results for a given operation may vary among implementations and between execution attempts in the same implementation.

Although the architecture does not prescribe the exact behavior for when results are allowed to be boundedly undefined, the results of executing instructions in contexts where results are allowed to be boundedly undefined are constrained to ones that could have been achieved by executing an arbitrary sequence of defined instructions, in valid form, starting in the state the machine was in before attempting to execute the given instruction.

Breakpoint. A programmable event that forces the core to take a breakpoint exception.

Burst. A multiple-beat data transfer whose total size is typically equal to a cache block.

Bus clock. Clock that causes the bus state transitions.

Bus master. The owner of the address or data bus; the device that initiates or requests the transaction.

C

Cache. High-speed memory containing recently accessed data or instructions (subset of main memory).

Cache block. A small region of contiguous memory that is copied from memory into a *cache*. The size of a cache block may vary among processors; the maximum block size is one *page*. In PowerPC processors, *cache coherency* is maintained on a cache-block basis. Note that the term ‘cache block’ is often used interchangeably with ‘cache line.’

Cache coherency. An attribute wherein an accurate and common view of memory is provided to all devices that share the same memory system. Caches are coherent if a processor performing a read from its cache is supplied with data corresponding to the most recent value written to memory or to another processor’s cache.

Cache flush. An operation that removes from a cache any data from a specified address range. This operation ensures that any modified data within the specified address range is written back to main memory. This operation is generated typically by a Data Cache Block Flush (**dcbf**) instruction.

Caching-inhibited. A memory update policy in which the *cache* is bypassed and the load or store is performed to or from main memory.

Cast out. A *cache block* that must be written to memory when a cache miss causes a cache block to be replaced.

Changed bit. One of two *page history bits* found in each *page table entry* (PTE). The processor sets the changed bit if any store is performed into the *page*. See also *Page access history bits* and *Referenced*.

Clean. An operation that causes a cache block to be written to memory, if modified, and then left in a valid, unmodified state in the cache.

Clear. To cause a bit or bit field to register a value of zero. See also *Set*.

Completer. In PCI-X, a completer is the device addressed by a transaction (other than a split completion transaction). If a target terminates a transaction with a split response, the completer becomes the initiator of the subsequent split completion.

Context synchronization. An operation that ensures that all instructions in execution complete past the point where they can produce an *exception*, that all instructions in execution complete in the context in which they began execution, and that all subsequent instructions are *fetched* and executed in the new context. Context synchronization may result from executing specific instructions (such as **isync** or **rfi**) or when certain events occur (such as an exception).

Copy-back 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

Direct-mapped cache. A cache in which each main memory address can appear in only one location within the cache; operates more quickly when the memory request is a cache hit.

Double data rate. Memory that allows data transfers at the start and end of a clock cycle. thereby doubling the data rate.

E

Effective address (EA). The 32-bit address specified for a load, store, or an instruction fetch. This address is then submitted to the MMU for translation to either a *physical memory* or an I/O address.

Exclusive state. MEI state (E) in which only one caching device contains data that is also in system memory.

F

Frame-check sequence (FCS). Specifies the standard 32-bit cyclic redundancy check (CRC) obtained using the standard CCITT-CRC polynomial on all fields except the preamble, SFD, and CRC.

Fetch. Retrieving instructions from either the cache or main memory and placing them into the instruction queue.

Flush. An operation that causes a cache block to be invalidated and the data, if modified, to be written to memory.

G

General-purpose register (GPR). Any of the 32 registers in the general-purpose register file. These registers provide the source operands and destination results for all integer data manipulation instructions. Integer load instructions move data from memory to GPRs and store instructions move data from GPRs to memory.

Gigabit media-independent interface (GMII) sublayer. Sublayer that provides a standard interface between the MAC layer and the physical layer for 1000-Mbps operation. It isolates the MAC layer and the physical layer, enabling the MAC layer to be used with various implementations of the physical layer.

Guarded. The guarded attribute pertains to out-of-order execution. When a page is designated as guarded, instructions and data cannot be accessed out-of-order.

H

Harvard architecture. An architectural model featuring separate caches and other memory management resources for instructions and data.

I

IEEE 754. A standard written by the Institute of Electrical and Electronics Engineers that defines operations and representations of binary floating-point numbers.

Illegal instructions. A class of instructions that are not implemented for a particular PowerPC processor. These include instructions not defined by the PowerPC architecture. In addition, for 32-bit implementations, instructions that are defined only for 64-bit implementations are considered to be illegal instructions. For 64-bit implementations instructions that are defined only for 32-bit implementations are considered to be illegal instructions.

Implementation. A particular processor that conforms to the PowerPC architecture, but may differ from other architecture-compliant implementations for example in design, feature set, and implementation of *optional* features. The PowerPC architecture has many different implementations.

Imprecise exception. A type of *synchronous exception* that is allowed not to adhere to the precise exception model (see *Precise exception*). The PowerPC architecture allows only floating-point exceptions to be handled imprecisely.

Inbound ATMU windows. Mappings that perform address translation from the external address space to the local address space, attach attributes and transaction types to the transaction, and map the transaction to its target interface.

Inter-packet gap. The gap between the end of one Ethernet packet and the beginning of the next transmitted packet.

Integer unit. An execution unit in the core responsible for executing integer instructions.

In-order. An aspect of an operation that adheres to a sequential model. An operation is said to be performed in-order if, at the time that it is performed, it is known to be required by the sequential execution model.

Instruction latency. The total number of clock cycles necessary to execute an instruction and make ready the results of that instruction.

K

Kill. An operation that causes a *cache block* to be invalidated without writing any modified data to memory.

L

Latency. The number of clock cycles necessary to execute an instruction and make ready the results of that execution for a subsequent instruction.

L2 cache. Level-2 cache. See *Secondary cache*.

Least significant bit (lsb). The bit of least value in an address, register, field, data element, or instruction encoding.

Least significant byte (LSB). The byte of least value in an address, register, data element, or instruction encoding.

Little endian. A byte-ordering method in memory where the address *n* of a word corresponds to the *least significant byte*. In an addressed memory word, the bytes are ordered (left to right) 3, 2, 1, 0, with 3 being the *most significant byte*. See *Big endian*.

Local access window. Mapping used to translate a region of memory to a particular target interface, such as the DDR SDRAM controller or the PCI controller. The local memory map is defined by a set of eight local access windows. The size of each window can be configured from 4 Kbytes to 2 Gbytes.

M

Media access control (MAC) sublayer. Sublayer that provides a logical connection between the MAC and its peer station. Its primary responsibility is to initialize, control, and manage the connection with the peer station.

Medium-dependent interface (MDI) sublayer. Sublayer that defines different connector types for different physical media and PMD devices.

Media-independent interface (MII) sublayer. Sublayer that provides a standard interface between the MAC layer and the physical layer for 10/100-Mbps operations. It isolates the MAC layer and the physical layer, enabling the MAC layer to be used with various implementations of the physical layer.

MEI (modified/exclusive/invalid). *Cache coherency* protocol used to manage caches on different devices that share a memory system. Note that the PowerPC architecture does not specify the implementation of a MEI protocol to ensure cache coherency.

Memory access ordering. The specific order in which the processor performs load and store memory accesses and the order in which those accesses complete.

Memory-mapped accesses. Accesses whose addresses use the page or block address translation mechanisms provided by the MMU and that occur externally with the bus protocol defined for memory.

Memory coherency. An aspect of caching in which it is ensured that an accurate view of memory is provided to all devices that share system memory.

Memory consistency. Refers to agreement of levels of memory with respect to a single processor and system memory (for example, on-chip cache, secondary cache, and system memory).

Memory management unit (MMU). The functional unit that is capable of translating an *effective* (logical) *address* to a physical address, providing protection mechanisms, and defining caching methods.

Modified state. MEI 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.

Most significant bit (msb). The highest-order bit in an address, registers, data element, or instruction encoding.

Most significant byte (MSB). The highest-order byte in an address, registers, data element, or instruction encoding.

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

OCeaN (on-chip network). Non-blocking crossbar switch fabric. Enables full duplex port connections at 128 Gb/s concurrent throughput and independent per port transaction queuing and flow control. Permits high bandwidth, high performance, as well as the execution of multiple data transactions.

Outbound ATMU windows. Mappings that perform address translations from local 32-bit address space to the address spaces of RapidIO or PCI/PCI-X RapidIO or PCI/PCI-X PCI PCI PCI PCI, which may be much larger than the local space. Outbound ATMU windows also map attributes such as transaction type or priority level.

P

Packet. A unit of binary data that can be routed through a network. Sometimes packet is used to refer to the frame plus the preamble and start frame delimiter (SFD).

Page. A region in memory. The OEA defines a page as a 4-Kbyte area of memory aligned on a 4-Kbyte boundary.

Page access history bits. The *changed* and *referenced* bits in the PTE keep track of the access history within the page. The referenced bit is set by the MMU whenever the page is accessed for a read or write operation. The changed bit is set when the page is stored into. See *Changed bit* and *Referenced bit*.

Page fault. A page fault is a condition that occurs when the processor attempts to access a memory location that does not reside within a *page* not currently resident in *physical memory*. On PowerPC processors, a page fault exception condition occurs when a matching, valid *page table entry* (PTE[V] = 1) cannot be located.

Page table. A table in memory is comprised of *page table entries*, or PTEs. It is further organized into eight PTEs per PTEG (page table entry group). The number of PTEGs in the page table depends on the size of the page table (as specified in the SDR1 register).

Page table entry (PTE). Data structures containing information used to translate *effective address* to physical address on a 4-Kbyte page basis. A PTE consists of 8 bytes of information in a 32-bit processor and 16 bytes of information in a 64-bit processor.

Physical coding sublayer (PCS). Sublayer responsible for encoding and decoding data stream to and from the MAC sublayer. Medium (1000BASEX) 8B/10B coding is used for fiber. Medium (1000BASET) 8B1Q coding is used for unshielded twisted pair (UTP).

Physical medium attachment (PMA) sublayer. Sublayer responsible for serializing code groups into a bit stream suitable for serial bit-oriented physical devices (SerDes) and vice versa. Synchronization is also performed for proper data decoding in this sublayer. The PMA sits between the PCS and the PMD sublayers. For fiber medium (1000BASEX) the interface on the PMD side of the PMA is a one-bit 1250-MHz signal, while on the PMA PCS side, the interface is a ten-bit interface (TBI) at 125 MHz. The TBI is an alternative to the GMII interface. If the TBI is used, the gigabit Ethernet controller must be capable of performing the PCS function. For UTP medium, the PMD interface side of the PMA consists of four pair of 62.5-MHz PAM5 encoded signals, while the PCS side provides the 1250-Mbps input to a 8B1Q4 PCS. For fiber medium (1000BASEX), the interface on the PMD side of the PMA is a one-bit 1250-MHz signal, while on the PMA PCS side, the interface is a ten-bit interface (TBI) at 125 MHz. The TBI is an alternative to the GMII interface. If the TBI is used the gigabit Ethernet controller must be capable of performing the PCS function. For UTP medium, the PMD interface side of the PMA consists of four pair of 62.5-MHz PAM5 encoded signals, while the PCS side provides the 1250-Mbps input to a 8B1Q4 PCS.

Physical medium dependent (PMD) sublayer. Sublayer responsible for signal transmission. The typical PMD functionality includes amplifier, modulation, and wave shaping. Different PMD devices may support different media.

Physical memory. The actual memory that can be accessed through the system's memory bus.

Pipelining. A technique that breaks operations, such as instruction processing or bus transactions, into smaller distinct stages or tenures (respectively) so that a subsequent operation can begin before the previous one has completed.

Precise exceptions. A category of exception for which the pipeline can be stopped so instructions that preceded the faulting instruction can complete and subsequent instructions can be flushed and redispatched after exception handling has completed. See *Imprecise exceptions*.

Primary opcode. The most-significant 6 bits (bits 0–5) of the instruction encoding that identifies the type of instruction.

Program order. The order of instructions in an executing program. More specifically, this term is used to refer to the original order in which program instructions are fetched into the instruction queue from the cache.

Protection boundary. A boundary between *protection domains*.

Protection domain. A protection domain is a segment, a virtual page, a BAT area, or a range of unmapped effective addresses. It is defined only when the appropriate relocate bit in the MSR (IR or DR) is 1.

Q

Quad word. A group of 16 contiguous locations starting at an address divisible by 16.

Quiesce. To come to rest. The processor is said to quiesce when an exception is taken or a **sync** instruction is executed. The instruction stream is stopped at the decode stage and executing instructions are allowed to complete to create a controlled context for instructions that may be affected by out-of-order, parallel execution. See *Context synchronization*.

R

rA. The **r**A instruction field is used to specify a GPR to be used as a source or destination.

rB. The **r**B instruction field is used to specify a GPR to be used as a source.

rD. The **r**D instruction field is used to specify a GPR to be used as a destination.

rS. The **r**S instruction field is used to specify a GPR to be used as a source.

RapidIO. High-performance, packet-switched, interconnect architecture that provides reliability, increased bandwidth, and faster bus speeds in an intra-system interconnect. Designed to be compatible with integrated communications processors, host processors, and networking digital signal processors,

Record bit. Bit 31 (or the Rc bit) in the instruction encoding. When it is set, updates the condition register (CR) to reflect the result of the operation.

Reconciliation sublayer. Sublayer that maps the terminology and commands used in the MAC layer into electrical formats appropriate for the physical layer entities.

Referenced bit. One of two *page history bits* found in each *page table entry*. The processor sets the *referenced bit* whenever the page is accessed for a read or write. See also *Page access history bits*.

Requester. In PCI-X, a requester is an initiator that first introduces a transaction into the PCI-X domain. If a transaction is terminated with a split response, the requester becomes the target of the subsequent split completion.

Reservation. The processor establishes a reservation on a *cache block* of memory space when it executes an **lwarx** instruction to read a memory semaphore into a GPR.

Reservation station. A buffer between the dispatch and execute stages that allows instructions to be dispatched even though the results of instructions on which the dispatched instruction may depend are not available.

RISC (reduced instruction set computing). An *architecture* characterized by fixed-length instructions with nonoverlapping functionality and by a separate set of load and store instructions that perform memory accesses.

S

Secondary cache. A cache memory that is typically larger and has a longer access time than the primary cache. A secondary cache may be shared by multiple devices. Also referred to as L2, or level-2, cache.

Sequence. In PCI-X, a sequence is one or more transactions associated with carrying out a single logical transfer by a requester. Each transaction in the same sequence carries the same unique sequence ID.

Set (*v*). To write a nonzero value to a bit or bit field; the opposite of *clear*. The term ‘set’ may also be used to generally describe the updating of a bit or bit field.

Set (*n*). A subdivision of a *cache*. Cacheable data can be stored in a given location in one of the sets, typically corresponding to its lower-order address bits. Because several memory locations can map to the same location, cached data is typically placed in the set whose *cache block* corresponding to that address was used least recently. See *Set associative*.

Set associative. Aspect of cache organization in which the cache space is divided into sections, called *sets*. The cache controller associates a particular main memory address with the contents of a particular set, or region, within the cache.

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. Response to a snooped transaction that hits a modified cache block. The cache block is written to memory and made available to the snooping device.

Stall. An occurrence when an instruction cannot proceed to the next stage.

Sticky bit. A bit that when *set* must be cleared explicitly.

Superscalar machine. A machine that can issue multiple instructions concurrently from a conventional linear instruction stream.

Supervisor mode. The privileged operation state of a processor. In supervisor mode, software, typically the operating system, can access all control registers and can access the supervisor memory space, among other privileged operations.

Synchronization. A process to ensure that operations occur strictly *in order*. See *Context synchronization*.

Synchronous exception. An *exception* that is generated by the execution of a particular instruction or instruction sequence. There are two types of synchronous exceptions, *precise* and *imprecise*.

System memory. The physical memory available to a processor.

T

Time-division multiplex (TDM). A single serial channel used by several channels taking turns.

Tenure. The period of bus mastership. There can be separate address bus tenures and data bus tenures.

TLB (translation lookaside buffer). A cache that holds recently-used *page table entries*.

Throughput. The measure of the number of instructions that are processed per clock cycle.

Transaction. A complete exchange between two bus devices. A transaction is typically comprised of an address tenure and one or more data tenures, which may overlap or occur separately from the address tenure. A transaction may be minimally comprised of an address tenure 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

User mode. The operating state of a processor used typically by application software. In user mode, software can access only certain control registers and can access only user memory space. No privileged operations can be performed. Also referred to as problem state.

V

Virtual address. An intermediate address used in the translation of an *effective address* to a physical address.

Virtual memory. The address space created using the memory management facilities of the processor. Program access to *virtual memory* is possible only when it coincides with *physical memory*.

W

Way. A location in the cache that holds a cache block, its tags, and status bits.

Word. A 32-bit data element.

Write-back. A cache memory update policy in which processor write cycles are directly written only to the cache. External memory is updated only indirectly, for example, when a modified cache block is *cast out* to make room for newer data.

Write-through. A cache memory update policy in which all processor write cycles are written to both the cache and memory.

Index