MOTOROLA
intelligence everywhere™

digital dna™

MPC7450UM/D
2/2003
Rev. 3

MPC7450 RISC Microprocessor Family User's Manual

Devices Supported: MPC7457
MPC7455
MPC7451
MPC7450
MPC7447
MPC7445
MPC7441

HOW TO REACH US:

USA/EUROPE/LOCATIONS NOT LISTED:

Motorola Literature Distribution;
P.O. Box 5405, Denver, Colorado 80217
1-303-675-2140 or 1-800-441-2447

JAPAN:

Motorola Japan Ltd.;
SPS, Technical Information Center,
3-20-1, Minami-Azabu Minato-ku,
Tokyo 106-8573 Japan
81-3-3440-3569

ASIA/PACIFIC:

Motorola Semiconductors H.K. Ltd.;
Silicon Harbour Centre, 2 Dai King Street,
Tai Po Industrial Estate, Tai Po, N.T., Hong Kong
852-26668334

TECHNICAL INFORMATION CENTER:

1-800-521-6274

HOME PAGE:

http://www.motorola.com/semiconductors

DOCUMENT COMMENTS:

FAX (512) 933-2625,
Attn: RISC Applications Engineering

MOTOROLA

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

Paragraph Number	Title	Page Number
2.3.4.3.5	Integer Store Gathering	2-79
2.3.4.3.6	Integer Load and Store with Byte-Reverse Instructions	2-79
2.3.4.3.7	Integer Load and Store Multiple Instructions	2-80
2.3.4.3.8	Integer Load and Store String Instructions	2-80
2.3.4.3.9	Floating-Point Load and Store Address Generation	2-81
2.3.4.3.10	Floating-Point Store Instructions	2-82
2.3.4.4	Branch and Flow Control Instructions	2-83
2.3.4.4.1	Branch Instruction Address Calculation	2-84
2.3.4.4.2	Branch Instructions	2-84
2.3.4.4.3	Condition Register Logical Instructions	2-85
2.3.4.4.4	Trap Instructions	2-85
2.3.4.5	System Linkage Instruction—UISA	2-85
2.3.4.6	Processor Control Instructions—UISA	2-86
2.3.4.6.1	Move to/from Condition Register Instructions	2-86
2.3.4.6.2	Move to/from Special-Purpose Register Instructions (UISA)	2-86
2.3.4.7	Memory Synchronization Instructions—UISA	2-88
2.3.5	PowerPC VEA Instructions	2-88
2.3.5.1	Processor Control Instructions—VEA	2-89
2.3.5.2	Memory Synchronization Instructions—VEA	2-89
2.3.5.3	Memory Control Instructions—VEA	2-90
2.3.5.3.1	User-Level Cache Instructions—VEA	2-90
2.3.5.4	Optional External Control Instructions	2-93
2.3.6	PowerPC OEA Instructions	2-94
2.3.6.1	System Linkage Instructions—OEA	2-94
2.3.6.2	Processor Control Instructions—OEA	2-94
2.3.6.3	Memory Control Instructions—OEA	2-98
2.3.6.3.1	Supervisor-Level Cache Management Instruction—(OEA)	2-99
2.3.6.3.2	Translation Lookaside Buffer Management Instructions—OEA	2-99
2.3.7	Recommended Simplified Mnemonics	2-100
2.3.8	Implementation-Specific Instructions	2-100
2.4	AltiVec Instructions	2-103
2.5	AltiVec UISA Instructions	2-104
2.5.1	Vector Integer Instructions	2-104
2.5.1.1	Vector Integer Arithmetic Instructions	2-104
2.5.1.2	Vector Integer Compare Instructions	2-106
2.5.1.3	Vector Integer Logical Instructions	2-107
2.5.1.4	Vector Integer Rotate and Shift Instructions	2-107
2.5.2	Vector Floating-Point Instructions	2-108
2.5.2.1	Vector Floating-Point Arithmetic Instructions	2-108
2.5.2.2	Vector Floating-Point Multiply-Add Instructions	2-109
2.5.2.3	Vector Floating-Point Rounding and Conversion Instructions	2-109
2.5.2.4	Vector Floating-Point Compare Instructions	2-110
2.5.2.5	Vector Floating-Point Estimate Instructions	2-110

CONTENTS

CONTENTS

Paragraph Number	Title	Page Number

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

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

Chapter 8 Signal Descriptions

CONTENTS

CONTENTS

Paragraph Number | Title | Page Number

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

FIGURES

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

ILLUSTRATIONS

ILLUSTRATIONS

ILLUSTRATIONS

ILLUSTRATIONS

ILLUSTRATIONS

TABLES

TABLES

TABLES

TABLES

TABLES

TABLES

TABLES

TABLES

TABLES

TABLES

Table Number	Title	Page Number

About This Book

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

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*

The *MPC7450 RISC Microprocessor Family User's Manual*, which describes MPC7451 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 features of the PowerPC architecture only for 32-bit implementations.

Contact your sales representative for a copy of the *Programming Environments Manual.*

This document and the *Programming Environments Manual* distinguish between the three levels, or programming environments, of the PowerPC architecture, which are as follows:

- PowerPC user instruction set architecture (UISA)—The UISA defines the level of the architecture to which user-level software should conform. The UISA defines the base user-level instruction set, user-level registers, data types, memory conventions, and the memory and programming models seen by application programmers.
- PowerPC virtual environment architecture (VEA)—The VEA, which is the smallest component of the PowerPC architecture, defines additional user-level functionality that falls outside typical user-level software requirements. The VEA describes the memory model for an environment in which multiple processors or other devices can access external memory 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 upon one another, beginning with a description and complete summary of the MPC7451 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 MPC7451 implements the AltiVec technology resources. There are two books that 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 Motorola documentation, see “Related Documentation,” on page xlvii.

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.motorola.com/semiconductors.

A list of the major differences between the *MPC7450 RISC Microprocessor Family User’s Manual* Revision 1 and Revision 2 is provided in Appendix D, “User’s Manual 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, MPC7447, MPC7450, MPC7451, MPC7455, and the MPC7457. It is assumed that the reader understands operating systems, microprocessor system design, basic principles of RISC processing, and details of the PowerPC architecture.

Organization

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

- Chapter 1, “Overview,” is useful for readers who want a general understanding of the features and functions of the PowerPC architecture and the MPC7451. 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. The major differences between the MPC7451 and the MPC7455 are listed in Section 1.5, “Differences Between MPC7441/MPC7451 and MPC7445/MPC7455.”

- Chapter 2, “Programming Model,” is useful for software engineers who need to understand the MPC7451-specific registers, operand conventions, and details regarding how PowerPC instructions are implemented on the MPC7451. Instructions are organized by function.
- Chapter 3, “L1, L2, and L3 Cache Operation,” discusses the cache and memory model as implemented on the MPC7451.
- Chapter 4, “Exceptions,” describes the exception model defined in the OEA and the specific exception model implemented on the MPC7451.
- Chapter 5, “Memory Management,” describes the MPC7451’s implementation of the memory management unit specified by the OEA.
- Chapter 6, “Instruction Timing,” provides information about latencies, interlocks, special situations, and various conditions to help make programming more efficient. This chapter is of special interest to software engineers and system designers.
- Chapter 7, “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 MPC7451.
- Chapter 9, “System Interface Operation,” describes signal timings for various operations. It also provides information for interfacing to the MPC7451.
- Chapter 10, “Power and Thermal Management,” provides information about power saving and thermal management for the MPC7451.
- Chapter 11, “Performance Monitor,” describes the operation of the performance monitor diagnostic tool incorporated in the MPC7451.
- Appendix A, “MPC7451 Instruction Set Listings,” lists all PowerPC instructions while indicating those instructions that are not implemented by the MPC7451; it also includes the instructions that are specific to the MPC7451. 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 MPC7451.
- Appendix C, “Special-Purpose Registers,” lists all MPC7451 SPRs.
- Appendix D, “User’s Manual Revision History,” lists the major differences between Revision 0, Revision 1, and Revision 2 of the *MPC7450 RISC Microprocessor User’s 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*, Second 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

Motorola 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.
- User's manuals—These books provide details about individual implementations and are intended for use with the *Programming Environments Manual.*
- Addenda/errata to user's manuals—Because some processors have follow-on parts an addendum is provided that describes the additional features and functionality changes. These addenda are intended for use with the corresponding user's 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 (MPC7441, MPC7445, MPC7447, MPC7450, MPC7451, MPC7455, and MPC7457) described in this book (*MPC7450 RISC Microprocessor Family User's Manual*). Note that when referring to the *MPC7451 RISC Microprocessor 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 user's manual.
- *The Programmer's Reference Guide for the PowerPC Architecture*: MPCPRG/D—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/D—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 Motorola processors.

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

Conventions

This document 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, $\overline{\textit{qual BG}}$
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
0000	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.
ⓐ	Indicates functionality defined by the AltiVec technology.

Acronyms and Abbreviations

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

Table i. . Acronyms and Abbreviated Terms

Term	Meaning
ALU	Arithmetic logic unit
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
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

Table i. . Acronyms and Abbreviated Terms (continued)

Term	Meaning
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 register 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
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

Table i. . Acronyms and Abbreviated Terms (continued)

Term	Meaning
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	The Programming Environments Manual
PID	Processor identification tag
PIM	The 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
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

Table i. . Acronyms and Abbreviated Terms (continued)

Term	Meaning
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
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 MPC7451 microprocessor features, including a block diagram showing the major functional components. It also provides information about how the MPC7451 implementation complies with the PowerPC and AltiVec™ architecture definitions. In addition, this manual supports the MPC7441, MPC7445, MPC7447, MPC7455, and the MPC7457. Any differences between the other microprocessors, including the MPC7450, are noted in the user’s manual. The MPC7451 has the same functionality as the MPC7450 and any differences in data regarding bus timing, signal behavior, and AC, DC, and thermal characteristics are detailed in the hardware specifications.

1.1 MPC7451 Microprocessor Overview

This section describes the features and general operation of the MPC7451 and provides a block diagram showing major functional units. The MPC7451 implements the PowerPC architecture and is a reduced instruction set computer (RISC) microprocessor. The MPC7451 consists of a processor core, 32-Kbyte separate L1 instruction and data caches, a 256-Kbyte L2 cache for the MPC7451 (512-Kbyte for MPC7457), and an internal L3 controller with tags that support a glueless backside L3 cache through a dedicated high-bandwidth interface. The core is a high-performance superscalar design supporting multiple execution units, including four independent units that execute AltiVec instructions.

The MPC7451 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 MPC7451 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 MPC7451 also implements the AltiVec instruction set architectural extension. The MPC7451 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 MPC7451-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 the 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 10 AltiVec instructions can be executing concurrently.

Note that for the MPC7451, double- and single-precision versions of floating-point instructions have the same latency. For example, a floating-point multiply-add instruction takes five cycles to execute, regardless of whether it is single- (**fmadds**) or double-precision (**fmadd**).

The MPC7451 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 four BAT entries, as described in Chapter 5, “Memory Management.” For information about the L1 caches, see Chapter 3, “L1, L2, and L3 Cache Operation.”

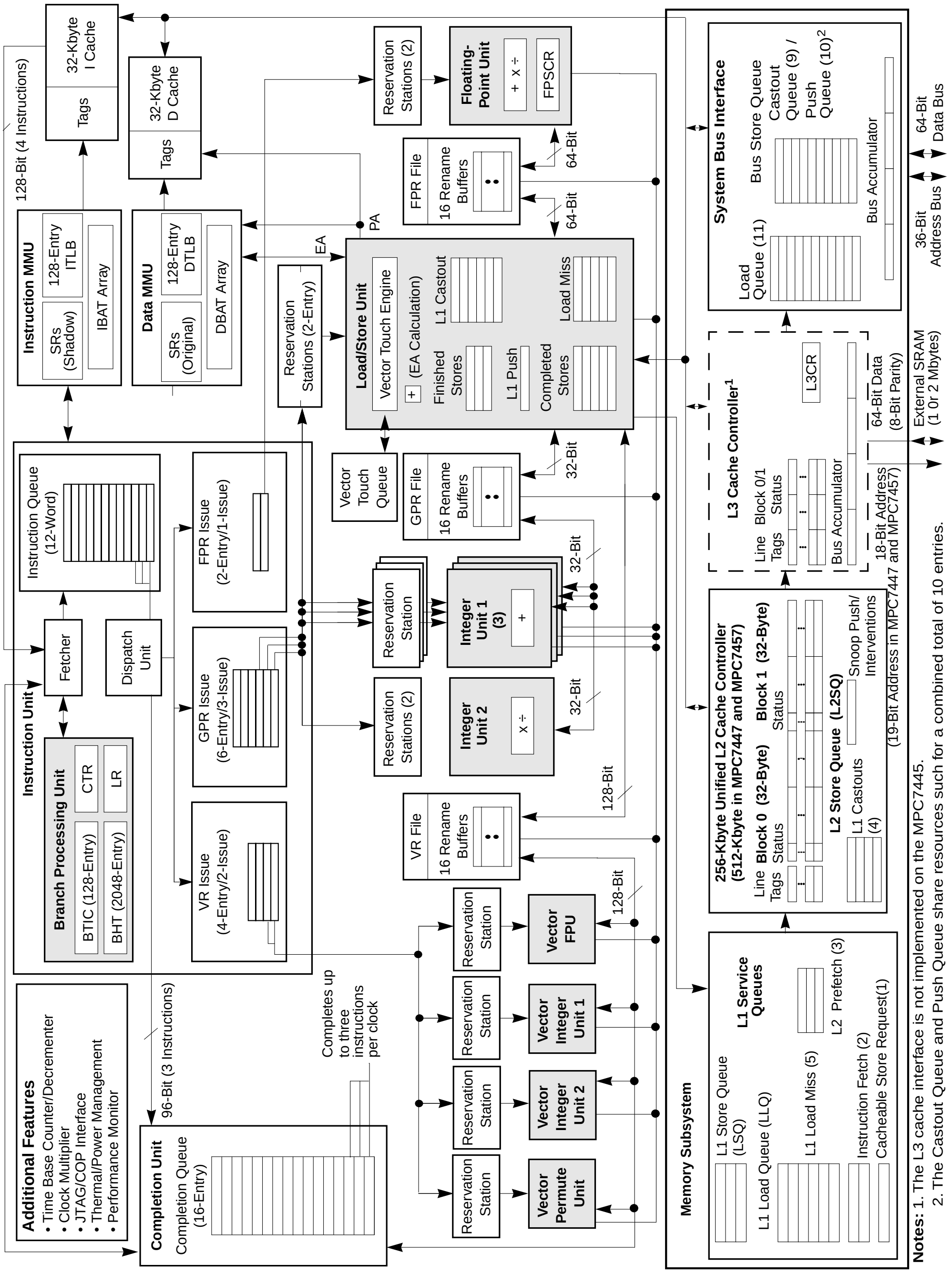
The MPC7451’s L2 cache is implemented with an on-chip, 256-Kbyte, eight-way set-associative physically addressed memory available for storing data, instructions, or both. For the MPC7447 and MPC7457 the L2 cache is 512-Kbyte. The L2 cache supports parity generation and checking for both tags and data. It responds with a nine-cycle load latency for an L1 miss that hits in L2. The L2 cache is fully pipelined for single-cycle throughput. 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.”

The MPC7451 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 Management,” describes how the power management can be used to reduce power consumption when the processor, or portions of it, are idle. Section 1.2.11, “Thermal Management,” describes how the instruction cache throttling mechanism reduces the instruction dispatch rate.The information in this section is 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 (that may be an approximation) can be used to trigger the performance monitor exception. Section 1.2.12, “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 attempting to show how features are implemented physically.



Notes: 1. The L3 cache interface is not implemented on the MPC7445.
2. The Castout Queue and Push Queue share resources such for a combined total of 10 entries.

Figure 1-1. MPC7451 Microprocessor Block Diagram

1.1.1 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. In the same way that the *MPC7450 RISC Microprocessor Family User's Manual* describes the functionality of the MPC7451, this document 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).

1.1.2 MPC7450 Microprocessor Overview

The functionality between the MPC7450 and the MPC7451 is the same. This document (*MPC7450 RISC Microprocessor Family User's Manual*) describes the functionality of the MPC7450 and any differences in data regarding bus timing, signal behavior, and AC, DC, and thermal characteristics are in the *MPC7450 RISC Microprocessor Hardware Specification.*

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:

- 4 IBAT and 4 DBAT additional registers
- Additional HID0 bits (HID0[HIGH_BAT_EN] and HID0[XBSEN]
- 4 more SPRG registers

The additional IBATs and DBATs provide mapping for more regions of memory. For more information on the 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. In the same way that the *MPC7450 RISC Microprocessor Family User's Manual* describes the functionality of the MPC7455, this document 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 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. In the same way that the *MPC7450 RISC Microprocessor Family User's Manual* describes the functionality of the MPC7457, this document 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).

1.1.6 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 Kbyte)
- Additional support for L3 Private Memory Size (4 Mbyte)
- An additional PLL Configuration Signal (PLL_CFG[4])
- 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 for the MPC7457, except where otherwise noted (in particular, the increased L2 cache and the additional L3 cache support is new for the MPC7457).

1.2 MPC7451 Microprocessor Features

This section describes the features of the MPC7451. The interrelationships of these features are shown in Figure 1-1.

1.2.1 Overview of the MPC7451 Microprocessor Features

Major features of the MPC7451 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 per clock cycle throughput 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 four instructions in the target stream.
  - 2048-entry branch history table (BHT) with two 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.
  - 8-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 IU1c) 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 (**vaddsbs**, **vaddshs**, and **vaddsws**, for example)
  - Vector integer unit 2 (VIU2) handles longer-latency AltiVec integer instructions, such as vector multiply add instructions (**vmhaddshs**, **vmhraddshs**, and **vmladduhm**, for example).
  - 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 1 cycle throughput
    - Four-cycle FPR load latency (single, double) with 1 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, 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).
- Rename buffers
  - 16 GPR rename buffers
  - 16 FPR rename buffers
  - 16 VR rename buffers
- Dispatch unit
  - The decode/dispatch stage fully decodes each instruction.
- Completion unit
  - The completion unit retires an instruction from the 16-entry completion queue (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 had 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—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 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 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.
- On-chip Level 2 (L2) cache has the following features:
  - Integrated 256-Kbyte, eight-way set-associative unified instruction and data cache for the MPC7451 (512-Kbyte for the MPC7447 and MPC7457).
  - Fully pipelined to provide 32 bytes per clock cycle to the L1 caches.
  - Total latency of nine processor cycles for L1 data cache miss that hits in the L2.
  - 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).
  - Supports parity generation and checking for both tags and data (enabled through L2CR).
- Level 3 (L3) cache interface (not supported on the MPC7441, MPC7445, and MPC7447)
  - Provides critical double-word forwarding to the requesting unit
  - On-chip tags support 1Mbyte 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 Mbyte, 2 Mbytes, or 4Mbytes (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
  - 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 256 bits per processor 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).
  - 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.
  - — 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 MPC7451 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 MPC7451-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 MPC7451-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 MPC7451 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 IQ two 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 one-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 MPC7451 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 MPC7451 provides the ten execution units described in the following sections.

1.2.2.4.1 AltiVec Vector Permute Unit (VPU)

The VPU execute 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. 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 five 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 MPC7451 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 MPC7451 writes the contents of the rename registers to the appropriate FPR when floating-point instructions are retired by the completion unit.

The MPC7451 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 the 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 four-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 MPC7451 executes store instructions with a maximum throughput of one per clock cycle and a three-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 MPC7451'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 MPC7451 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, MPC7451) or 4 GBytes (MPC7445, MPC7447, MPC7455, MPC7457).

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, which accesses 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 MPC7451 instruction and data TLBs are 128-entry, two-way set-associative caches that contain address translations. The MPC7451 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 MPC7451 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 MPC7451 L1 cache organization is shown in Figure 1-2.

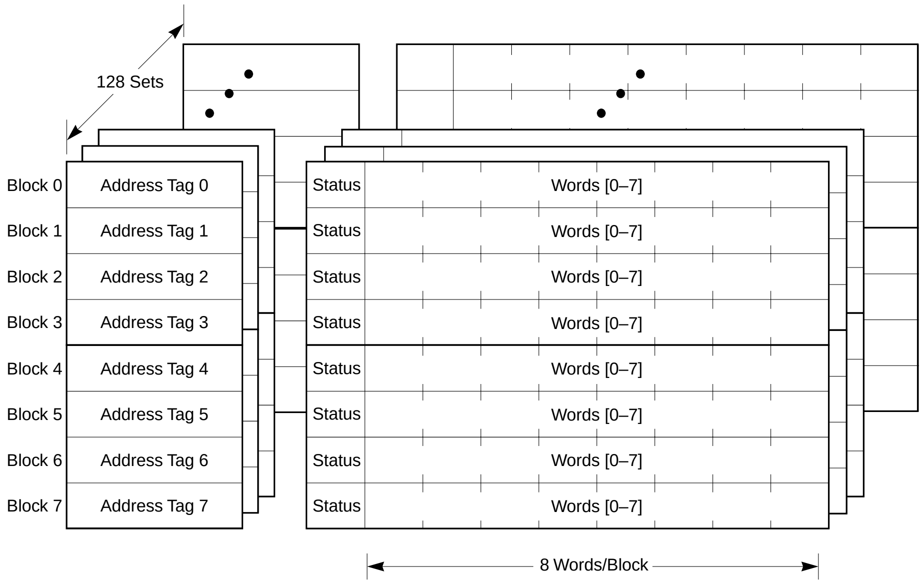


Figure 1-2. L1 Cache Organization

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 MPC7451 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 MPC7451 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 1-3.

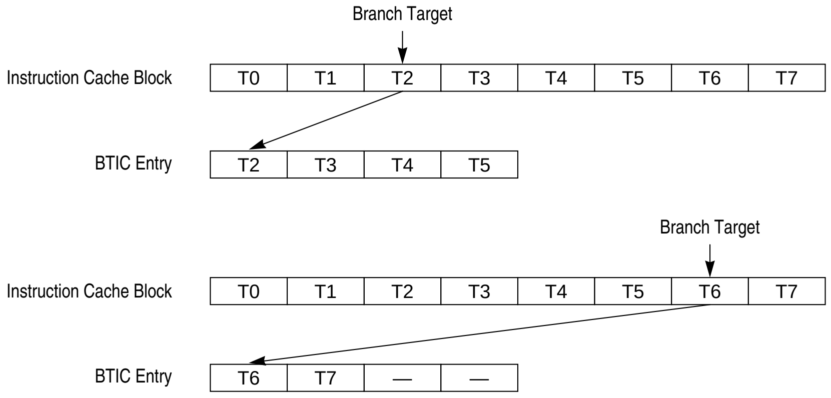


Figure 1-3. Alignment of Target Instructions in the BTIC

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 MPC7451 is a unified (containing both instructions and data) 256 Kbyte on-chip cache. For the MPC7447 and the MPC7457, the L2 cache has been increased to 512-Kbyte on-chip cache. 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 MPC7451 integrated L2 cache organization is shown in Figure 1-4.

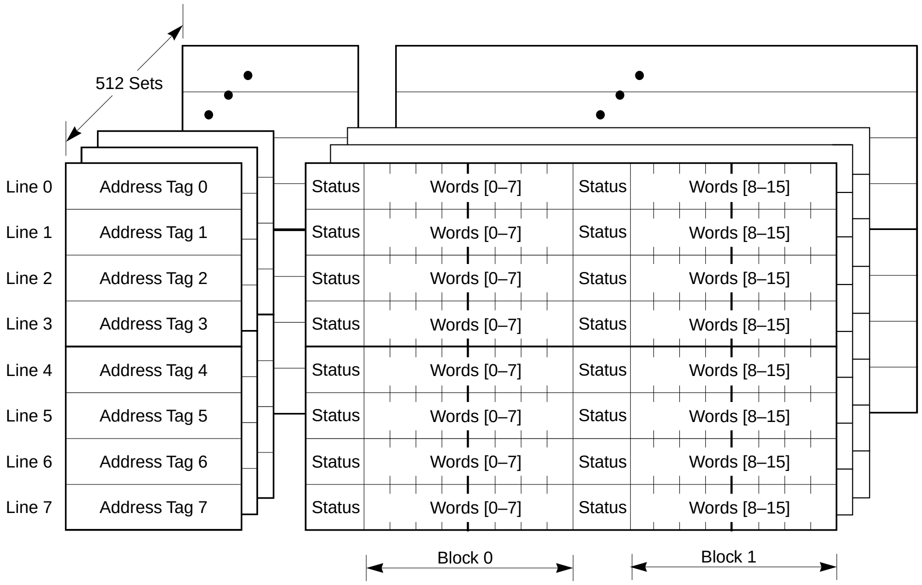


Figure 1-4. L2 Cache Organization for MPC7451

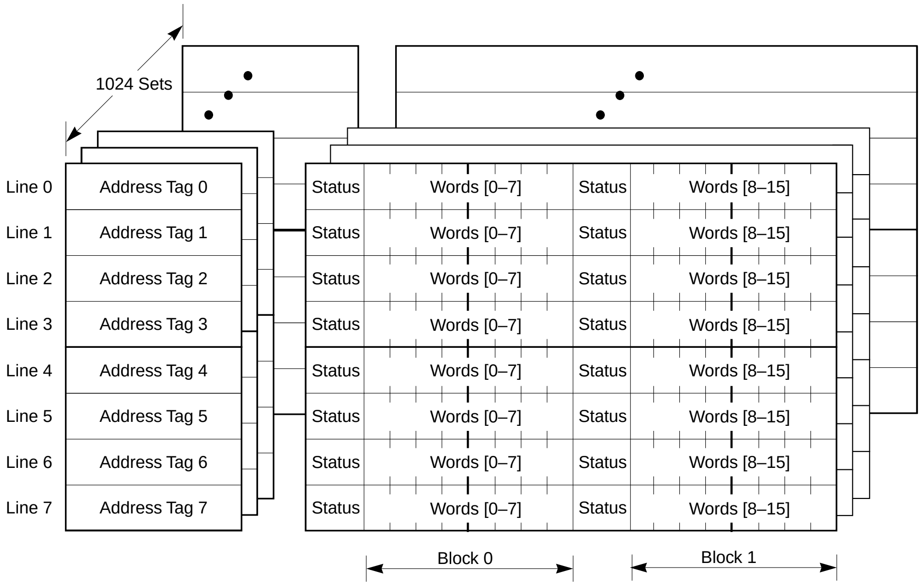


Figure 1-5. L2 Cache Organization for the MPC7447 and MPC7457

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 one 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 Mbyte of external synchronous SRAMs. Note that the L3 cache is not supported on the MPC7441 and the MPC7445.

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 can support 1,2, or 4 Mbytes of private memory. Accesses to private memory does not propagate to the system bus. The MPC7451 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 to 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.

1.2.7 System Interface

The MPC7451 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 MPC7451's performance is optimized for the MPX bus, its use is recommended over the 60x bus.

The MPC7451 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 MPC7451 uses the MPX bus protocol; if $\overline{\text{BMODE0}}$ is negated during the negation of $\overline{\text{HRESET}}$, the MPC7451 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 MPC7451 Bus Operation Features

The MPC7451 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 MPC7451 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 and the MPC7445.

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 MPC7451 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 and 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 loads, stores, and instruction fetches, the MPC7451 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 MPC7451 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 MPC7451 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 MPC7451 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 MPC7451 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 MPC7451 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 MPC7451 inputs are sampled and all outputs are driven on the rising edge of the bus clock cycle. The *MPC7451 RISC Microprocessor*

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 MPC7451 MPX and 60x bus interface protocol signals are grouped as follows:

- Address arbitration—The MPC7451 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 MPC7451 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 MPC7451 signals control and affect other aspects of the device, aside from the bus protocol. They are as follows:

- L3 cache address/data—The MPC7451 has separate address and data buses for accessing the L3 cache. Note that the L3 cache interface is not supported by the MPC7441 and the MPC7445.
- 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 and the MPC7445.

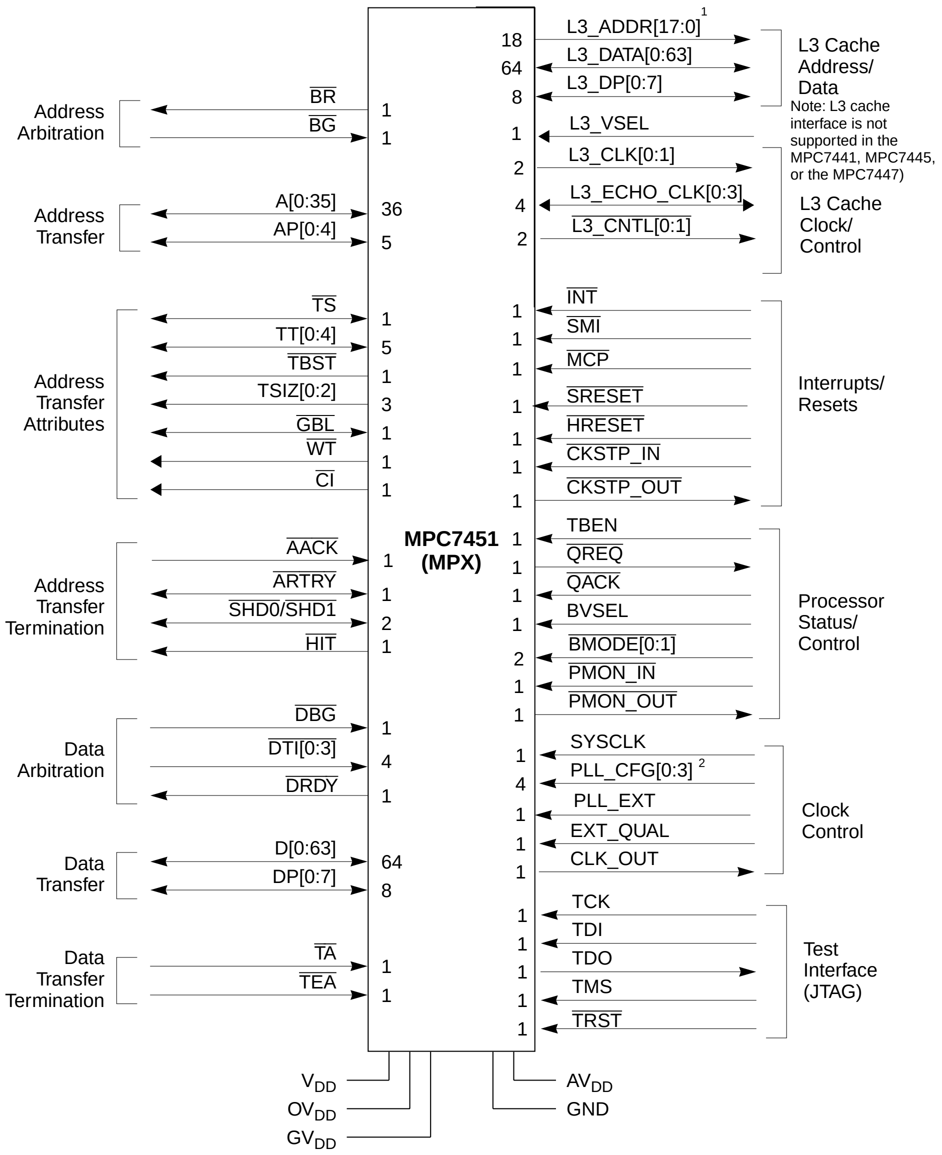
- Interrupts/resets—These signals include the external interrupt signal, checkstop signals, and both soft reset and hard reset signals. They are used to interrupt and, under various conditions, to reset the processor.
- Processor status and control—These signals enable the time-base facility and are 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.

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 1-6 illustrates the MPC7451's signal configuration in MPX bus mode, showing how the signals are grouped. A pinout diagram and tables showing pin numbers are included in the *MPC7451 RISC Microprocessor Hardware Specifications*. Note that the left side of the figure depicts the signals that implement the MPX bus protocol and the right side of the figure shows the remaining signals on the MPC7451 (not part of the bus protocol).



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

[2] For the MPC7447 and MPC7457, there are 5 PLL_CFG signals, (PLL_CFG[0:4])

Figure 1-6. MPX Bus Signal Groups

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 MPC7451 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 MPC7451’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[0:3] signals (for the MPC7447 and MPC7457, PLL_CFG[0:4]) are used to program the internal clock rate to a multiple of SYSCLK as defined in the *MPC7451 RISC Microprocessor 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 MPC7451 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 MPC7451 core clock frequency using a divisor selected through L3CR[L3CLK].

1.2.10 Power Management

The MPC7451 is designed for low-power operation. It provides both automatic and program-controlled power reduction modes. If an MPC7451 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 MPC7451 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 MPC7451 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.

Chapter 10, “Power and Thermal Management,” describes power saving modes for the MPC7451.

1.2.11 Thermal Management

The MPC7451 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 MPC7451 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 MPC7451.

1.2.12 Performance Monitor

The MPC7451 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 MPC7451.

1.3 MPC7451 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 MPC7451 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 MPC7451 features are listed in Section 1.2, “MPC7451 Microprocessor Features.”

This section describes the PowerPC architecture in general, and specific details about the implementation of the MPC7451 as a low-power, 32-bit device that implements this architecture. The structure of this section follows the user’s 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 MPC7451.

  Instruction set and addressing modes—Section 1.3.2, “Instruction Set,” describes the PowerPC instruction set and addressing modes for the PowerPC operating environment architecture, and defines and describes the PowerPC instructions implemented in the MPC7451. 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 MPC7451 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 MPC7451 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 MPC7451’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 MPC7451. 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 MPC7451 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.

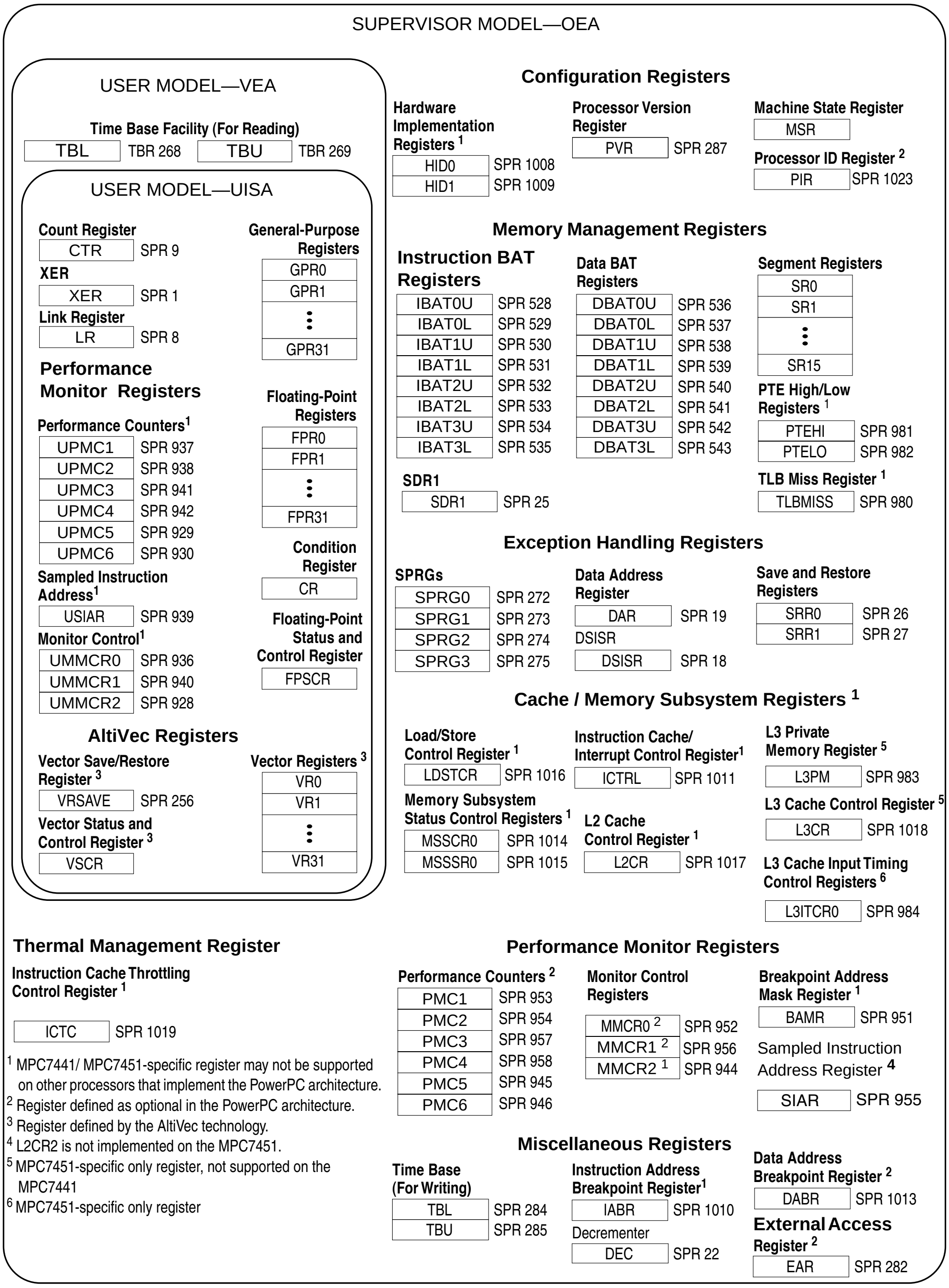
The PowerPC architecture also defines two levels of privilege—supervisor mode of operation (typically used by the operating system) and user mode of operation (used by the application software). The programming models incorporate 32 GPRs, 32 FPRs, SPRs, and several miscellaneous registers. The AltiVec extensions to the PowerPC architecture augment the programming model with 32 VRs, one status and control register, and one 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 1-7 shows all the MPC7451 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 1-7, 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.

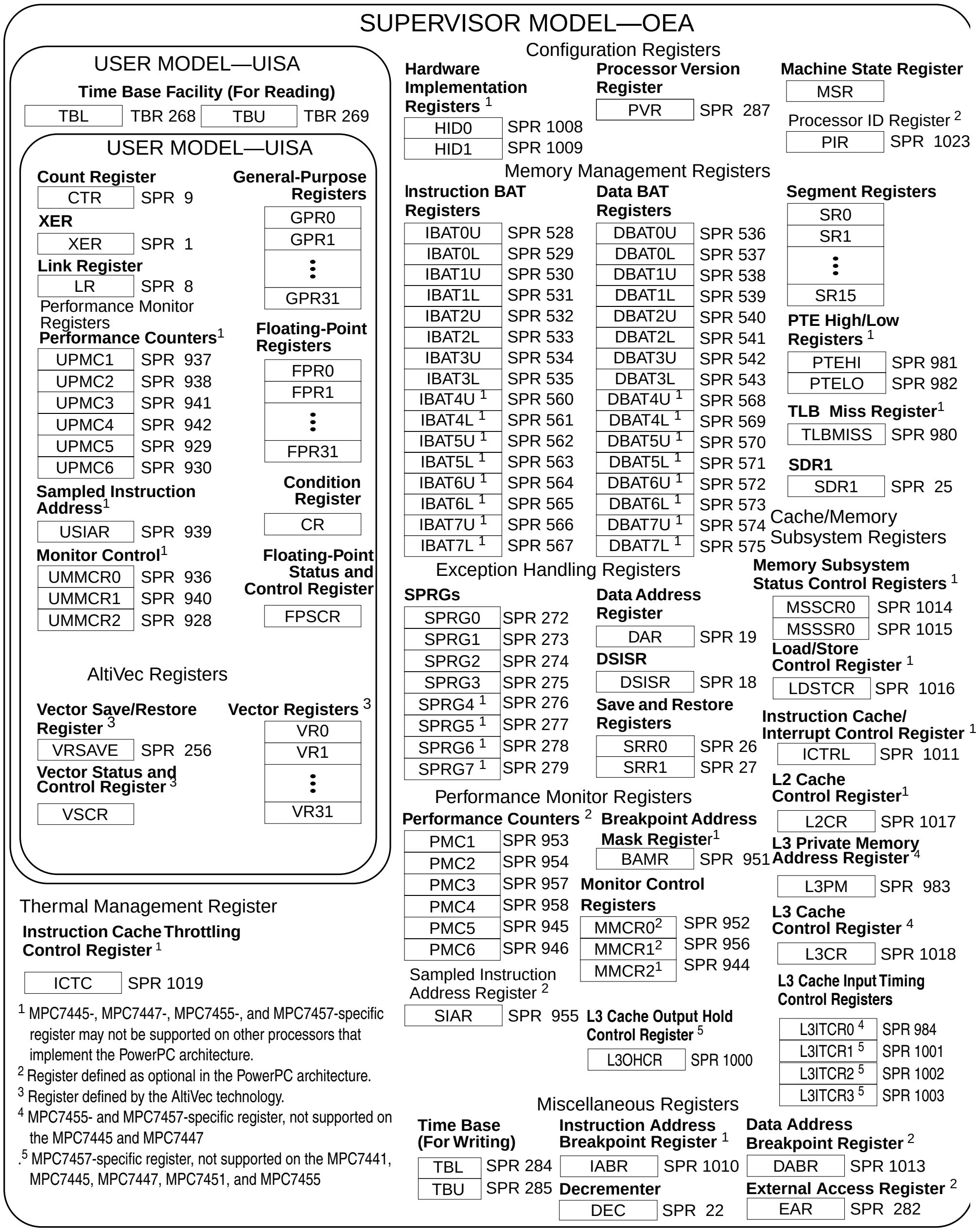
Figure 1-7 shows the MPC7441 and MPC7451 register set.



[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] L2CR2 is not implemented on the MPC7451.
[5] MPC7451-specific only register, not supported on the MPC7441
[6] MPC7451-specific only register

Figure 1-7. Programming Model—MPC7441/MPC7451 Microprocessor Registers

Figure 1-8 shows the MPC7445, MPC7455, MPC7447, 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, not supported on the MPC7445 and MPC7447
.[5] MPC7457-specific register, not supported on the MPC7441, MPC7445, MPC7447, MPC7451, and MPC7455

Figure 1-8. Programming Model—MPC7445, MPC7447, MPC7455, and MPC7457 Microprocessor Registers

Some registers can be accessed both explicitly and implicitly. In the MPC7451, all SPRs are 32 bits wide. Table 1-1 describes registers implemented by the MPC7451.

Table 1-1. Register Summary for MPC7451

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.	2.1.5.9 & 11.3.2.1, 2.1.5.9.4 & 11.3.3.1, 2.1.5.9.6 & 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.	2.1.5.9.9, 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.	2.1.5.9.11, 11.3.7.1
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

Table 1-1. Register Summary for MPC7451 (continued)

Name	SPR	Description	Reference / Section
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 2.1.4.1 2.3.5.1
		OEA	
BAMR [1, 3]	951	Breakpoint address mask register. Used in conjunction with the events that monitor IABR hits.	2.1.5.9.7, 11.3.5
DABR [4, 5]	1013	Data address breakpoint register. Optional register implemented in the MPC7451 and is used to cause a breakpoint exception if a specified data address is encountered.	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 MPC7451, 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
EAR [6, 7]	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.	PEM
HID0 [1, 7] HID1 [1, 8]	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[0:3] (for the MPC7447 and MPC7457, PLL_CFG[0:4]) clock signals and control other bus-related functions.	2.1.5.1, 2.1.5.2
IABR [1, 9]	1010	Instruction address breakpoint register. Used to cause a breakpoint exception if a specified instruction address is encountered.	2.1.5.6

Table 1-1. Register Summary for MPC7451 (continued)

Name	SPR	Description	Reference / Section
IBAT0U/L [10] IBAT1U/L [10] IBAT2U/L [10] IBAT3U/L [10] IBAT4U/L [10, 11] IBAT5U/L [10, 11] IBAT6U/L [10, 11] IBAT7U/L [10, 11] DBAT0U/L [12] DBAT1U/L [12] DBAT2U/L [12] DBAT3U/L [12] DBAT4U/L [11, 12] DBAT5U/L [11, 12] DBAT6U/L [11, 12] DBAT7U/L [11, 12]	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 MPC7451 implements IBAT[G]; however, attempting to execute code from an IBAT area with G = 1 causes an ISI exception.	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.	2.1.5.8, 10.3
ICTRL [1, 7]	1011	Instruction cache and interrupt control register. Used in configuring interrupts and error reporting for the instruction and data caches.	2.1.5.5.8
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.	2.1.5.5.1
L3CR [13]	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.	2.1.5.5.2
L3ITCR0 [13] L3ITCR1 [14] L3ITCR2 [14] L3ITCR3 [14]	984 1001 1002 1003	L3 cache input timing control register. Includes bits for controlling the input AC timing of the L3 cache interface.	2.1.5.5.4 2.1.5.5.5 2.1.5.5.6 2.1.5.5.7
L3OHCR [14]	1000	L3 cache output hold control register. Includes bits for controlling the output AC timing of the L3 cache interface of the MPC7457.	2.1.5.5.3
L3PM [13, 15]	983	The L3 private memory register. Configures the base address of the range of addresses that the L3 uses as private memory (not cache).	2.1.5.5.10
LDSTCR [1, 16]	1016	Load/store control register. Controls data L1 cache way-locking.	2.1.5.5.9
MMCR0 [4], MMCR1 [4], 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.	2.1.5.9.1, 11.3.2 2.1.5.9.3, 11.3.3 2.1.5.9.5, 11.3.4

Table 1-1. Register Summary for MPC7451 (continued)

Name	SPR	Description	Reference / Section
MSR [7]	—	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 4.2, “MPC7451 Exception Recognition and Priorities.” The following bits are optional in the PowerPC architecture. Note that setting MSR[EE] masks decrementer and external interrupt exceptions and MPC7451-specific system management, and performance monitor exceptions.	PEM, 2.1.3.3, 4.3

Bit	Name	Description
6	VEC	AltiVec available. MPC7451 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.
13	POW	Power management enable. MPC7451-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 2-6.
29	PMM	Performance monitor marked mode. MPC7451-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.

Name	SPR	Description	Reference / Section
MSSCR0 [1, 17]	1014	Memory subsystem control register. Used to configure and operate many aspects of the memory subsystem.	2.1.5.3
MSSSR0 [1]	1015	Memory subsystem status register. Used to configure and operate the parity functions in the L2 and L3 caches for the MPC7451.	2.1.5.4
PIR	1023	Processor identification register. Provided for system use. MPC7451 does not change PIR contents.	PEM
PMC1–PMC6 [4]	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.	2.1.5.9.8, 11.3.6

Table 1-1. Register Summary for MPC7451 (continued)

Name	SPR	Description	Reference / Section
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.	2.1.5.7.2, 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, 2.1.4.1
SDAR, USDAR	—	Sampled data address register. The MPC7451 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 MPC7451. A **mtspr** or **mfspr** SDAR or USDAR instruction causes a program exception**.**	2.1.5.9.12
SDR1 [18]	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 MPC7451 to support the extended 36-bit physical address (when HID0[XAEN] = 1]).	PEM, 2.1.3.5, 5.5.1
SIAR [4]	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.	2.1.5.9.11 11.3.7
SPRG0–SPRG3 SPRG4–SPRG7 [11]	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 [19]	—	Segment registers (SR*n*). Note that the MPC7451 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.	PEM
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 MPC7451 sets one or more error bits in SRR1. Refer to the individual exceptions for individual SRR1 bit settings.	PEM, 2.1.3.4 4.3

Table 1-1. Register Summary for MPC7451 (continued)

Name	SPR	Description	Reference / Section
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 2.1.4.1 2.3.4.7
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.	2.1.5.7.1 5.5.5.1.1

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

[2] Register is defined by the AltiVec technology.

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

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

[5] 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.

[6] 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.

[7] For specific synchronization requirements on the register see Table 2-32.

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

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

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

[11] MPC7445- and MPC7455-specific register.

[12] 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.

[13] MPC7451- and MPC7455-specific, not supported on the MPC7441 and MPC7445

[14] MPC7457-specific, not supported on the MPC7441, MPC7445, MPC7447, MPC7451, and MPC7455

[15] A 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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
  - 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 2.5.5, “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 2.3.4.6, “Processor Control Instructions—UISA.”
- Memory control instructions—These instructions are used for managing of caches (user level and supervisor level), described in Section 2.3.5.3, “Memory Control Instructions—VEA.”

1.3.2.3 MPC7451 Microprocessor Instruction Set

The MPC7451 instruction set is defined as follows:

- The MPC7451 provides hardware support for all 32-bit PowerPC instructions.
- The MPC7451 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 MPC7451-specific implementation, respectively. A detailed description of the MPC7451 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 MPC7451 Microprocessor Cache Implementation

The MPC7451 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 MPC7451 implementation. A detailed description of the MPC7451 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 MPC7451 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 1-2. 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 1-2. MPC7451 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, “MPC7451 Exception Recognition and Priorities.” For a better understanding of how the MPC7451 implements precise exceptions, see Chapter 6, “Instruction Timing.” Table 1-3 lists the exceptions implemented in the MPC7451, and conditions that cause them. Table 1-3 also notes the MPC7451-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 1-3. 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 cache tag parity error • L3 SRAM error • L3 tag parity errors. MSR[ME] must be set.
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 MPC7451 operates as specified in the OEA by taking this exception on an **isync**.

Table 1-3. Exceptions and Conditions (continued)

Exception Type	Vector Offset	Causing Conditions
Reserved	0x00E00	The MPC7451 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 (MPC7451-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**). (MPC7451-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 (MPC7451-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 (MPC7451-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 (MPC7451-specific).
Instruction address breakpoint	0x01300	IABR[0–29] matches EA[0–29] of the next instruction to complete and IABR[BE] = 1 (MPC7451-specific).
System management interrupt	0x01400	MSR[EE] = 1 and $\overline{\text{SMI}}$ is asserted (MPC7451-specific).
Reserved	0x01500–015FF	—
AltiVec assist	0x01600	This MPC7451-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 MPC7451 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 MPC7451 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 MPC7451). In addition, two translation lookaside buffers (TLBs) are implemented on the MPC7451 to keep recently used page address translations on-chip. Although the PowerPC OEA describes one MMU (conceptually), the MPC7451 hardware maintains separate TLBs and table search resources for instruction and data accesses that can be performed independently (and simultaneously). Therefore, the MPC7451 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 MPC7451, 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 MPC7451 Microprocessor Memory Management Implementation

The MPC7451 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 MPC7451 MMU is described in Section 1.2.3, “Memory Management Units (MMUs).”

The MPC7451 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 MPC7451 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 MPC7451 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 MPC7451; however, if HID0[STEN] = 1, a software table search is performed.

The MPC7451 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, an DTLB miss-on-load exception.
- For a data store, an DTLB miss-on-store exception.

The MPC7451 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 MPC7451 microprocessor performs operations defined by instructions and how it reports the results of instruction execution. The MPC7451 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 MPC7451 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 1-9 represents a generic four-stage pipelined execution unit, which when filled has a throughput of one instruction per clock cycle.

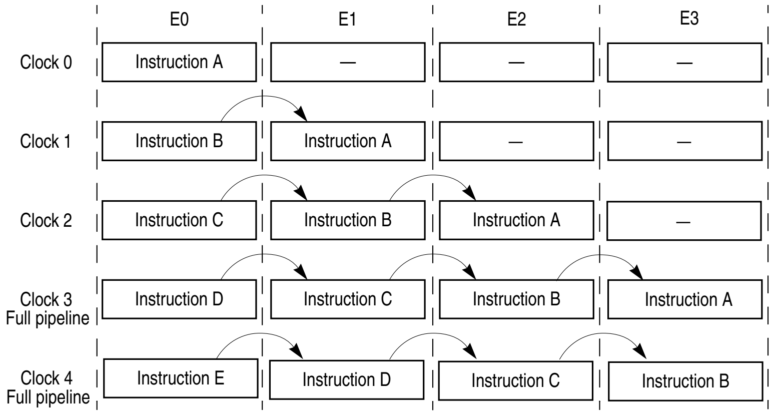


Figure 1-9. Pipelined Execution Unit

Figure 1-10 shows the entire path that instructions take through the fetch1, fetch2, decode/dispatch, execute, issue, complete, and write-back stages, which is considered the MPC7451's master pipeline. The FPU, LSU, IU2, VIU2, VFPU, and VPU are multiple-stage pipelines.

The MPC7451 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. Moreover, the VIU2, VFPU, and VPU are pipelined, so they can operate on multiple instructions.

The MPC7451 can complete as many as three instructions on each clock cycle. In general, the MPC7451 processes instructions in seven stages—fetch1, fetch2, decode/dispatch, issue, execute, complete, and writeback as shown in Figure 1-10. Note that the pipeline example in Figure 6-1 is similar to the four-stage VFPU pipeline in Figure 1-10.

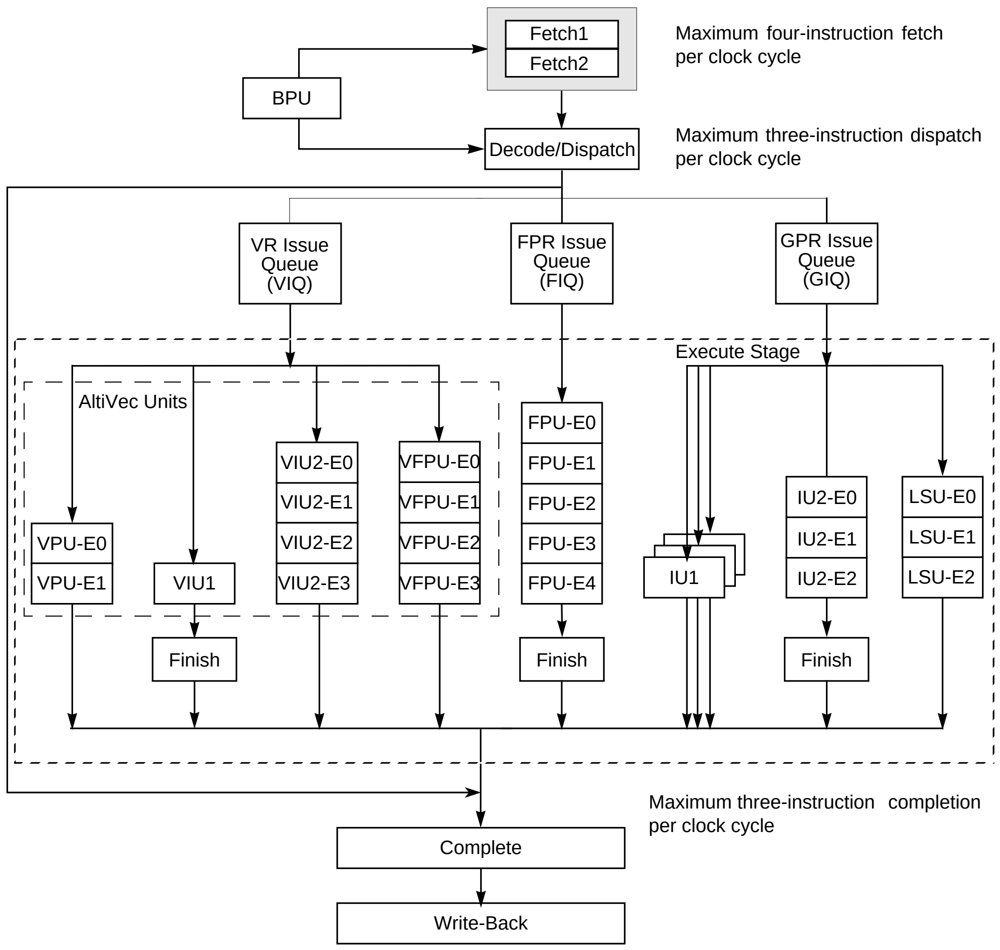


Figure 1-10. Superscalar/Pipeline Diagram

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 MPC7451 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 MPC7451. The AltiVec implementation is described fully in Chapter 7, "AltiVec Technology Implementation."

1.4 Differences between MPC7451 and MPC7400/ MPC7410

Table 1-4 compares the key features of the MPC7451 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 MPC7451 is extended (compared to the MPC7400), while maintaining the same level of performance (in terms of number of instructions executed per clock cycle. Table 1-4 shows these differences.

Table 1-4. MPC7451 and MPC7400/MPC7410 Feature Comparison

Microarchitectural Feature	MPC7451	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
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	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	In order

Table 1-4. MPC7451 and MPC7400/MPC7410 Feature Comparison (continued)

Microarchitectural Feature	MPC7451	MPC7400/MPC7410
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
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	32K/32K
Associativity	8-way	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	4 streams

Table 1-4. MPC7451 and MPC7400/MPC7410 Feature Comparison (continued)

Microarchitectural Feature	MPC7451	MPC7400/MPC7410
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	64
Private memory SRAM sizes	1 Mbyte, 2 Mbytes	512 Kbyte, 1 Mbyte, 2 Mbytes
Parity	Byte	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 1-5. 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 —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

1.6 Differences Between MPC7441/MPC7451 and MPC7447/MPC7457

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 of the differences. Table 1-4 shows these differences.

Table 1-6. MPC7451 and MPC7457 Differences

Microarchitectural Feature	MPC7441/MPC7451	MPC7447/MPC7457	Section
L2 Cache			
Cache level	L2	L2	3.6
Size/associativity	256-Kbyte/8-way	512-Kbyte/8-way	3.6.1
Access width	256 bits	256 bits	3.6
Number of 32-byte sectors/ line	2	2	3.6
Parity	Byte	Byte	3.6.3.1.2
Off-Chip Cache Support [1]			
Cache level	L3	L3	3.7
On-chip tag logical size	1 Mbyte, 2 Mbytes	1 Mbyte, 2 Mbytes, 4Mbytes	3.7.3.2
Associativity	8-way	8-way	3.7
Number of 32 byte sectors/line	2	2	3.7
Off-chip data SRAM support	MSUG2 DDR, LW, PB2	MSUG2 DDR, LW, PB2	3.7.3.9
Data path width	64 bits	64 bits	
Private memory SRAM sizes	1 Mbyte, 2 Mbyte	1 Mbyte, 2 Mbyte, 4 Mbyte	3.7.3.2
Parity	Byte	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	2.1.5.5.2
Signals			
L3 Address Signals	L3_ADDR[0:17]	L3_ADDR[0:18]	8.4.1.1
PLL Configuration Signals	PLL_CFG[0:3]	PLL_CFG[0:4]	2.1.5.2

Table 1-6. MPC7451 and MPC7457 Differences

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	2.1.5.2

[1] L3 cache interface is not supported on the MPC7441 and MPC7447

1.7 User’s Manual Revision History

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

Chapter 2
Programming Model

This chapter describes the MPC7451 programming model, emphasizing those features specific to the MPC7451 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 MPC7451
- Operand conventions
- The MPC7451 instruction set

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

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."

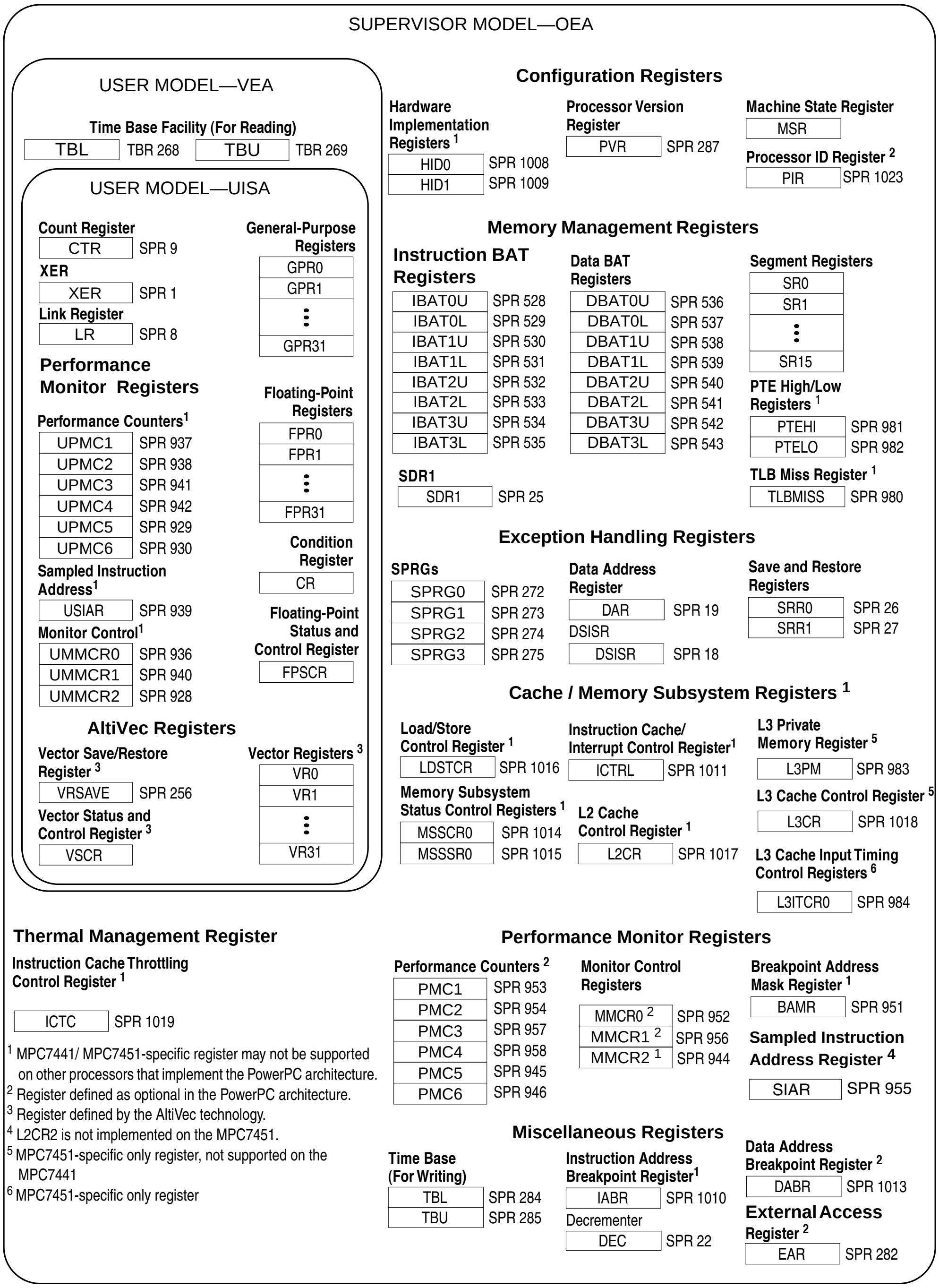
2.1 MPC7451 Processor Register Set

This section describes the registers implemented in the MPC7451. 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 MPC7451, and a detailed description of MPC7451-specific registers. Full descriptions of the architecture-defined register set are provided in Chapter 2, "PowerPC 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 is 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.

2.1.1 Register Set Overview

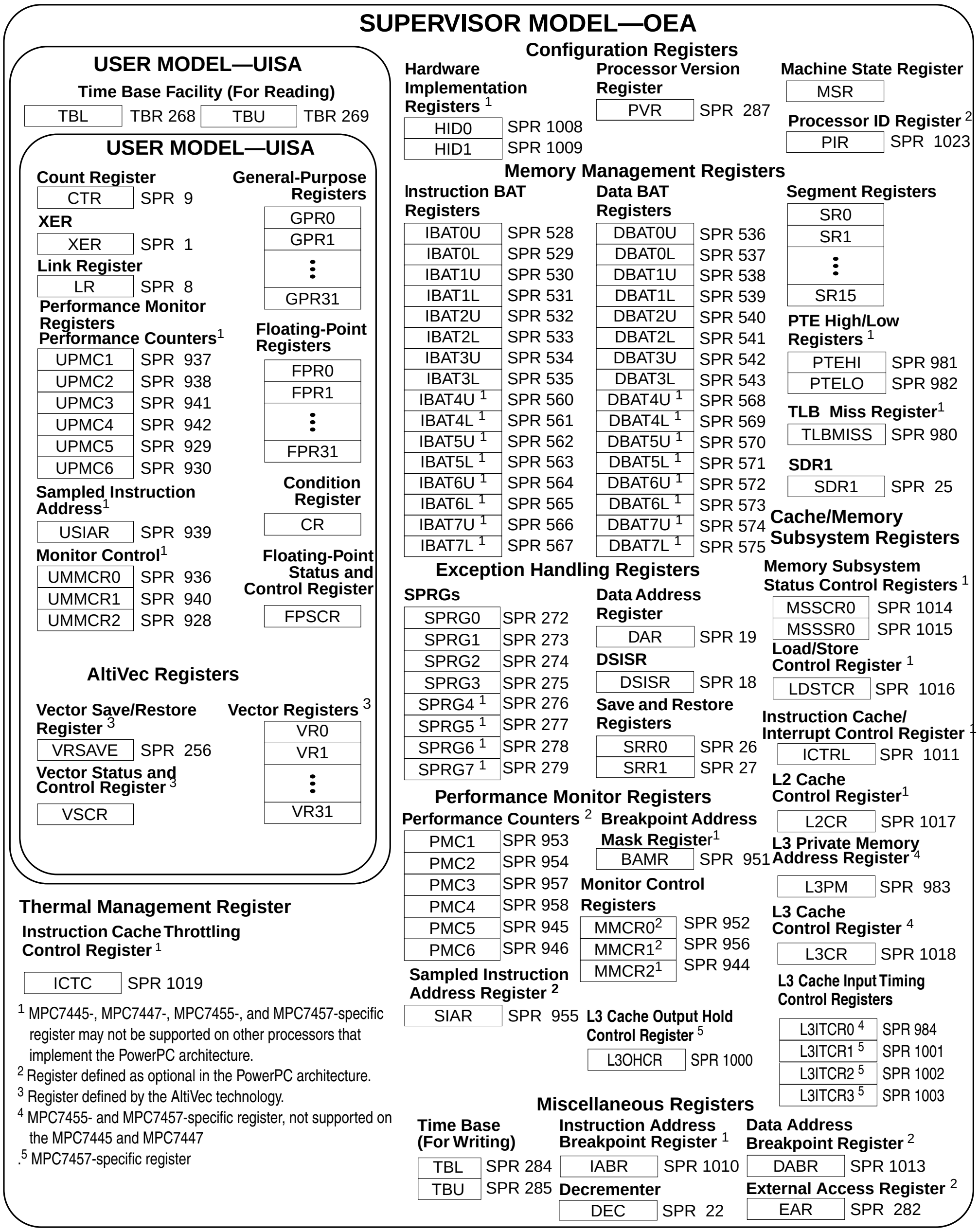
Figure 2-1 shows the MPC7441 and MPC7451 register set.



[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] L2CR2 is not implemented on the MPC7451.
[5] MPC7451-specific only register, not supported on the MPC7441
[6] MPC7451-specific only register

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

Figure 2-6 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, not supported on the MPC7445 and MPC7447

.[5] MPC7457-specific register

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

2.1.2 MPC7451 Register Set

Table 2-1 summarizes the registers implemented in the MPC7451.

Table 2-1. Register Summary for the MPC7451

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.	2.1.5.9 & 11.3.2.1, 2.1.5.9.4 & 11.3.3.1, 2.1.5.9.6 & 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.	2.1.5.9.9, 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.	2.1.5.9.11, 11.3.7.1

Table 2-1. Register Summary for the MPC7451 (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 2.1.4.1 2.3.5.1
OEA			
BAMR [1, 3]	951	Breakpoint address mask register. Used in conjunction with the events that monitor IABR hits.	2.1.5.9.7, 11.3.5
DABR [4, 5]	1013	Data address breakpoint register. Optional register implemented in the MPC7451 and is used to cause a breakpoint exception if a specified data address is encountered.	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 MPC7451, 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
EAR [6, 7]	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.	PEM
HID0 [1, 7] HID1 [1, 8]	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] clock signals and control other bus-related functions.	2.1.5.1, 2.1.5.2

Table 2-1. Register Summary for the MPC7451 (continued)

Name	SPR	Description	Reference / Section
IABR [1, 9]	1010	Instruction address breakpoint register. Used to cause a breakpoint exception if a specified instruction address is encountered.	2.1.5.6
IBAT0U/L [10] IBAT1U/L [10] IBAT2U/L [10] IBAT3U/L [10] IBAT4U/L [10, 11] IBAT5U/L [10, 11] IBAT6U/L [10, 11] IBAT7U/L [10, 11] DBAT0U/L [12] DBAT1U/L [12] DBAT2U/L [12] DBAT3U/L [12] DBAT4U/L [11, 12] DBAT5U/L [11, 12] DBAT6U/L [11, 12] DBAT7U/L [11, 12]	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 MPC7451 implements IBAT[G]; however, attempting to execute code from an IBAT area with G = 1 causes an ISI exception.	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.	2.1.5.8, 10.3
ICTRL [1, 7]	1011	Instruction cache and interrupt control register. Used in configuring interrupts and error reporting for the instruction and data caches.	2.1.5.5.8
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.	2.1.5.5.1
L3CR [13]	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.	2.1.5.5.2
L3ITCR0 [13] L3ITCR1 [14] L3ITCR2 [14] L3ITCR3 [14]	984 1001 1002 1003	L3 cache input timing control register. Includes bits for controlling the input AC timing of the L3 cache interface.	2.1.5.5.4 2.1.5.5.5 2.1.5.5.6 2.1.5.5.7
L3OHCR [14]	1000	L3 cache output hold control register. Includes bits for controlling the output AC timing of the L3 cache interface of the MPC7457.	2.1.5.5.3
L3PM [13, 15]	983	The L3 private memory register. Configures the base address of the range of addresses that the L3 uses as private memory (not cache).	2.1.5.5.10
LDSTCR [1, 16]	1016	Load/store control register. Controls data L1 cache way-locking.	2.1.5.5.9

Table 2-1. Register Summary for the MPC7451 (continued)

Name	SPR	Description	Reference / Section
MMCR0[4] MMCR1[4] 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.	2.1.5.9.1, 11.3.2 2.1.5.9.3, 11.3.3 2.1.5.9.5, 11.3.4
MSR[7]	—	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 MPC7451-specific system management, and performance monitor exceptions.	PEM, 2.1.3.3, 4.3

Bit	Name	Description
6	VEC	AltiVec available. MPC7451 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.
13	POW	Power management enable. MPC7451-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 2-6.
29	PMM	Performance monitor marked mode. MPC7451-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.

Name	SPR	Description	Reference / Section
MSSCR0[1, 17]	1014	Memory subsystem control register. Used to configure and operate many aspects of the memory subsystem.	2.1.5.3
MSSSR0[1]	1015	Memory subsystem status register. Used to configure and operate the parity functions in the L2 and L3 caches for the MPC7451.	2.1.5.4
PIR	1023	Processor identification register. Provided for system use. All 32 bits of the PIR can be written to with the **mtspr** instruction.	PEM 2.1.3.2

Table 2-1. Register Summary for the MPC7451 (continued)

Name	SPR	Description	Reference / Section
PMC1–PMC6 [4]	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.	2.1.5.9.8, 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.	2.1.5.7.2, 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, 2.1.3.1
SDAR, USDAR	—	Sampled data address register. The MPC7451 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 MPC7451. A **mtspr** or **mfspr** SDAR or USDAR instruction causes a program exception**.**	2.1.5.9.12
SDR1 [18]	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 MPC7451 to support the extended 36-bit physical address (when HID0[XAEN] = 1]).	PEM, 2.1.3.5, 5.5.1
SIAR [4]	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.	2.1.5.9.11 11.3.7
SPRG0–SPRG3 SPRG4–SPRG7 [11]	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 [19]	—	Segment registers (SR*n*). Note that the MPC7451 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.	PEM
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 MPC7451 sets one or more error bits in SRR1. Refer to the individual exceptions for individual SRR1 bit settings.	PEM, 2.1.3.4, 4.3

Table 2-1. Register Summary for the MPC7451 (continued)

Name	SPR	Description	Reference / Section
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 2.1.4.1 2.3.5.1
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.	2.1.5.7.1 5.5.5.1.1

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

[2] Register is defined by the AltiVec technology.

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

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

[5] 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.

[6] 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.

[7] For specific synchronization requirements on the register see Table 2-32.

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

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

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

[11] MPC7445-, MPC7447-, MPC7455-, and MPC7457-specific register.

[12] 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.

[13] MPC7451-, MPC7455-, MPC7457-specific, not supported on the MPC7441, MPC7445, and MPC7447

[14] MPC7457-specific, not supported on the MPC7441, MPC7445, MPC7447, MPC7451, and MPC7455

[15] A 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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 MPC7451 fully decodes the SPR field of the instruction. If the SPR specified is undefined, an illegal instruction program exception occurs.

2.1.3 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 MPC7451.

2.1.3.1 Processor Version Register (PVR)

For more information, see “Processor Version Register (PVR),” in Chapter 2, “PowerPC Register Set,” of *The Programming Environments Manual*.

Implementation Note—The processor version number is 0x8000,0x8001, 0x8002, for the MPC7451, MPC7455, and MPC7457 respectively. The processor revision level starts at 0x0200 for the MPC7451 and 0x0100 for the MPC7455 and MPC7457. The revision level is updated for each silicon revision. Table 2-2 describes the MPC7451 PVR bits that are not required by the PowerPC architecture.

Table 2-2. 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

2.1.3.2 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 MPC7451 provides write access to the PIR with **mtspr** using SPR 1023.

2.1.3.3 Machine State Register (MSR)

The MSR defines the state of the processor. When an exception occurs, MSR bits, as described in Table 2-3 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 MPC7451’s MSR is shown in Figure 2-3.

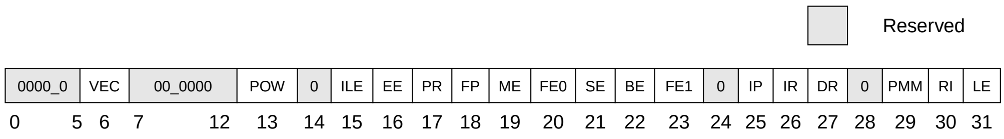


Figure 2-3. Machine State Register (MSR)

The MSR bits are defined in Table 2-3.

Table 2-3. 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 2-3. 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 2-4)
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 2-4)
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.

Table 2-3. MSR Bit Settings (continued)

Bit(s)	Name	Description
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 2-4, if either FE0 or FE1 are set, the MPC7451 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 2-4. IEEE Floating-Point Exception Mode Bits

FE0	FE1	Mode
0	0	Floating-point exceptions disabled
0	1	Imprecise nonrecoverable. For this setting, the MPC7451 operates in floating-point precise mode.
1	0	Imprecise recoverable. For this setting, the MPC7451 operates in floating-point precise mode.
1	1	Floating-point precise mode

2.1.3.4 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 2-4.

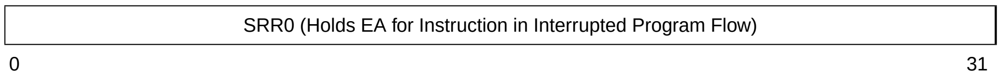


Figure 2-4. Machine Status Save/Restore Register 0 (SRR0)

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 2-5.

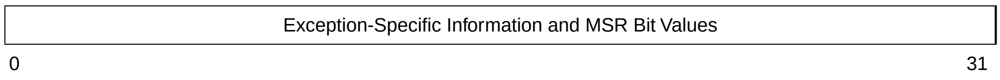


Figure 2-5. Machine Status Save/Restore Register 1 (SRR1)

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 2-3 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."

2.1.3.5 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 MPC7451 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 2-6 shows the format of the modified SDR1.

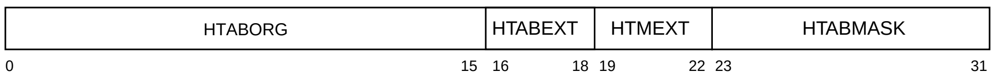


Figure 2-6. SDR1 Register Format—Extended Addressing

Bit settings for the SDR1 register are described in Table 2-5.

Table 2-5. 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 2.3.2.4, “Synchronization.”

2.1.4 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).

2.1.4.1 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*.

2.1.5 MPC7451-Specific Register Descriptions

The PowerPC architecture allows for implementation-specific SPRs. This section describes registers that are defined for the MPC7451 but are not included in the PowerPC architecture. Note that in the MPC7451, these registers are all supervisor-level registers. All the registers described in the *AltiVec Technology Programming Environments Manual* are implemented in MPC7451. See Chapter 2, “AltiVec Register Set,” in the *AltiVec Technology Programming Environments Manual* for details about these registers.

Note that while it is not guaranteed that the implementation of MPC7451-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. Then the implementation-specific registers for exception processing and memory management are presented, followed by the thermal management register. Finally the performance monitor registers are presented.

2.1.5.1 Hardware Implementation-Dependent Register 0 (HID0)

The hardware implementation-dependent register 0 (HID0) controls the state of several functions within the MPC7451. The HID0 register for the MPC7441 and the MPC7451 is shown in Figure 2-7.

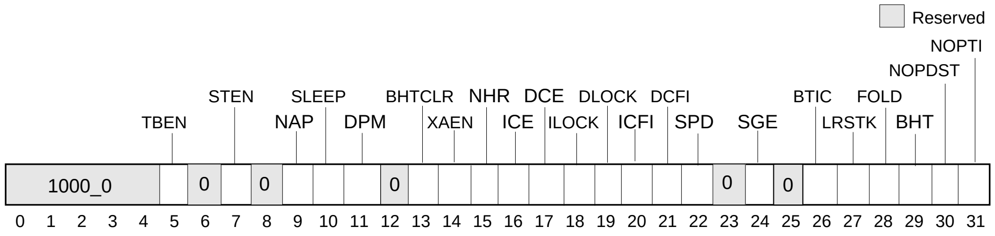


Figure 2-7. Hardware Implementation-Dependent Register 0 (HID0) for the MPC7441 and the MPC7451

The HID0 register for the MPC7445 and the MPC7455 is shown in Figure 2-8.

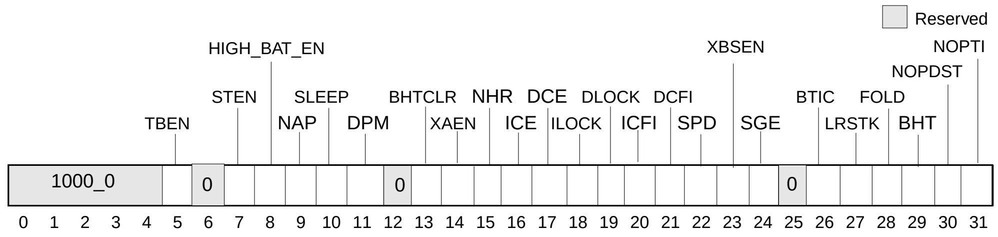


Figure 2-8. Hardware Implementation-Dependent Register 0 (HID0) for the MPC7445 and the MPC7455

The HID0 bits are described in Table 2-6.

Table 2-6. 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 MPC7451 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 MPC7451 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, 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 MPC7451 ignores the SLEEP bit.

Table 2-6. HID0 Field Descriptions (continued)

Bits	Name	Description
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[0:3] (for the MPC7447 and MPC7457, 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 MPC7451 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.
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 MPC7451 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 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.

Table 2-6. HID0 Field Descriptions (continued)

Bits	Name	Description
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.
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 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 MPC7451.

Table 2-6. HID0 Field Descriptions (continued)

Bits	Name	Description
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 MPC7451.
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	—	Reserved. Defined as IFTT or IFEM on some earlier processors.
	XBSEN	Extended BAT Block Size Enable. 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.
24	SGE [8]	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.

Table 2-6. HID0 Field Descriptions (continued)

Bits	Name	Description
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 MPC7451 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 MPC7451 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 [8]	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] 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 2.3.2.4, “Synchronization.”

2.1.5.2 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. The HID1 bits are shown in Figure 2-9.

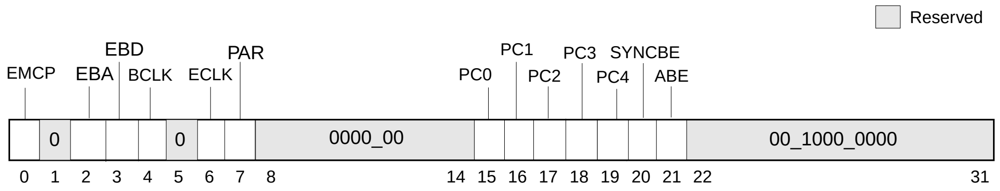


Figure 2-9. Hardware Implementation-Dependent Register 1 (HID1)

The HID1 bits are described in Table 2-7.

Table 2-7. 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 MPC7451 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 MPC7451 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 2-8.
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 2-8.

Table 2-7. HID1 Field Descriptions (continued)

Bits [1]	Name	Description
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.
8–14	—	Reserved
15	PC0	PLL configuration bit 0 (read-only). Reflects the state of PLL CFG[0].
16	PC1	PLL configuration bit 1 (read-only). Reflects the state of PLL CFG[1].
17	PC2	PLL configuration bit 2 (read-only). Reflects the state of PLL CFG[2].
18	PC3	PLL configuration bit 3 (read-only). Reflects the state of PLL CFG[3].
19	PC4	PLL configuration bit 4 (read-only). Reflects the state of PLL CFG[4].
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.

Table 2-8 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 2-8. 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 2.3.2.4, “Synchronization.”

2.1.5.3 Memory Subsystem Control Register (MSSCR0)

The memory subsystem control register (MSSCR0), shown in Figure 2-10, is used to configure and operate the memory subsystem for the MPC7451. It is accessed as SPR 1014. The MSSCR0 is initialized to all 0s except for the read-only bits.

Because MSSCR0 alters how the MPC7451 responds to snoop requests, it is important that changes to the value of MSSCR0 are handled correctly.

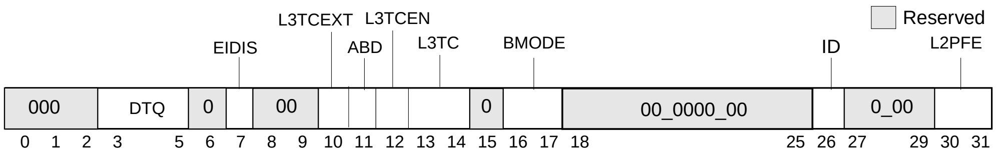


Figure 2-10. Memory Subsystem Control Register (MSSCR0)

Table 2-9 describes MSSCR0 fields.

Table 2-9. 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 MPC7451 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 MPC7451 performs external pushes instead of external interventions. External interventions are disabled.
8–9	—	Reserved

Table 2-9. MSSCR0 Field Descriptions (continued)

Bits	Name	Function
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 MPC7451 and is used as an L3 turnaround clock count only on the MPC7457.
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. 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 MPC7451. 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 MPC7451. 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.

Table 2-9. MSSCR0 Field Descriptions (continued)

Bits	Name	Function
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.
27–29	—	Reserved. Read as zeroes.
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 01One 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.

2.1.5.4 Memory Subsystem Status Register (MSSSR0)

The memory subsystem status register (MSSSR0), shown in Figure 2-11, is used to report parity in the L2 and L3 caches of the MPC7451. It is accessed as SPR 1015. The MSSSR0 is initialized to all 0s except for the read-only bits.

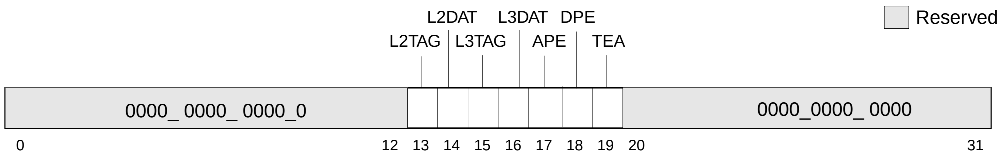


Figure 2-11. Memory Subsystem Status Register (MSSSR0)

Table 2-10 describes MSSSR0 fields.

Table 2-10. 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.

Table 2-10. MSSSR0 Field Descriptions (continued)

Bits	Name	Description
15	L3TAG	L3 tag parity error 0 L3 tag parity error not detected. 1 L3 tag parity error detected.
16	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

2.1.5.5 Instruction and Data Cache Registers

There are several registers 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.

2.1.5.5.1 L2 Cache Control Register (L2CR)

The L2 cache control register (L2CR), shown in Figure 2-12, is a supervisor-level, implementation-specific SPR used to configure and operate the L2 cache. It is cleared by a hard reset or power-on reset.

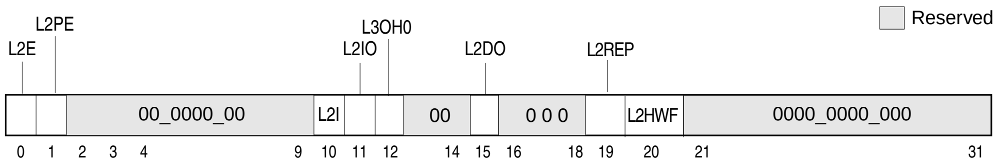


Figure 2-12. L2 Cache Control Register (L2CR)

The L2 cache interface is described in Chapter 3, “L1, L2, and L3 Cache Operation.” The L2CR bits are described in Table 2-11.

Table 2-11. L2CR Field Descriptions

Bits	Name	Description
0	L2E	L2 cache enable 0 L2 cache operation (including snooping) disabled 1 L2 cache operation (including snooping) enabled The L2 cache operation is enabled starting with the next transaction the L2 cache unit receives. Before enabling the L2 cache, all other L2CR bits must be set appropriately. The L2 cache may need to be invalidated globally.
1	L2PE	L2 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.
2–3	—	Reserved Must be set by software during initialization to ob00.
4–9	—	Reserved
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 mode 0 Instruction-only operation in the L2 cache disabled 1 Instruction-only operation in the L2 cache enabled Enables instruction-only operation in the L2 cache. For this operation, only instruction accesses cause new entries to be allocated in the L2 cache. Data addresses already in the cache still hit for the L1 data cache. When both L2CR[L2DO] and L2CR[L2IO] are set, the L2 cache is effectively locked.
12	L3OH0	L3 output hold 0. These bits configure 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 Hardware Specification* for specific output hold times.
13–14	—	Reserved
15	L2DO	L2 data-only mode 0 Data-only operation in the L2 cache disabled 1 Data-only operation in the L2 cache enabled Enables data-only operation in the L2 cache. When this bit is set, only data accesses can be cached in the L2 cache. Instruction accesses are serviced for instruction addresses already in the L2 cache; however, the L2 cache is not reloaded for L1 instruction cache misses. Note that setting both L2CR[L2D] and L2CR[L2IO] effectively locks the L2 cache.
16–18	—	Reserved
19	L2REP	L2 replacement algorithm 0 When this bit is cleared, the default replacement algorithm is used 1 When this bit is set, the secondary replacement algorithm is used See Section 3.6.4.4, "L2 Cache Line Replacement Algorithms," for more information.

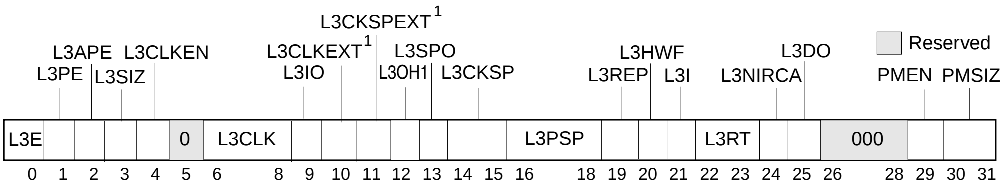
Table 2-11. L2CR Field Descriptions (continued)

Bits	Name	Description
20	L2HWF	L2 hardware flush. 0 L2 hardware flush disabled 1 L2 hardware flush enabled When L2CR[L2HWF] is set, the L2 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 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. See Section 3.6.3.1.5, "Flushing of L1, L2, and L3 Caches," for more information.
21–31	—	Reserved

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

2.1.5.5.2 L3 Cache Control Register (L3CR)

The L3 cache control register (L3CR), shown in Figure 2-20, 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.



[1]MPC7457-specific bit

Figure 2-13. 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 2-12.

Table 2-12. 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 *MPC7451 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 MPC7451 always generates L3 data parity.
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 MPC7451 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 MPC7451 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).

Table 2-12. L3CR Field Descriptions (continued)

Bits	Name	Description
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 *MPC7451 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 MPC7451: 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 MPC7451. 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 MPC7451 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.
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 MPC7451 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 MPC7451 and is used as an L3 clock sample point extension only on the MPC7457.
12	—	Reserved

Table 2-12. L3CR Field Descriptions (continued)

Bits	Name	Description
12	L3OH1	MPC7455: L3 output hold 1. These bits configure 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 Hardware Specification* 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 Section 3.7.3.8, "L3 Cache Clock and Timing Controls," and the *MPC7451 Hardware Specifications* for further clarification. The following values are correct for the MPC7451. 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 MPC7451. 00 6 clocks 01 7 clocks 10 8 clocks 11 9 clocks
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 *MPC7451 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 MPC7451. For the MPC7457, it is 6 clocks. 111 Reserved on the MPC7451. 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."

Table 2-12. L3CR Field Descriptions (continued)

Bits	Name	Description
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 MPC7451 and on each clock edge • Late-write SRAMs which are required by the MPC7451 to be of the pipelined (register-register) configurations • Pipeline burst SRAMs, referred to as PB2-type SRAMs For burst RAM selections, the MPC7451 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
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 MPC7451 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 MPC7451 to drive the L3_CLK[0:1] signals one-quarter of a core clock earlier at the expense of AC setup timing. See the *MPC7451 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

Table 2-12. L3CR Field Descriptions (continued)

Bits	Name	Description
29	PMEN	Private memory enable 0 Private memory disabled 1 Private memory enabled When this bit is set, the MPC7451 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 MB 1 2 MB Note that L3CR[30] bit is reserved on the MPC7451 and MPC7455. For the MPC7457, L3CR[30—31] is used: 00 1 MB 01 2 MB 10 4 MB 11 Reserved

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

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

The L3 cache output hold control register (L3OHCR), shown in Figure 2-20, 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 Hardware Specification*.

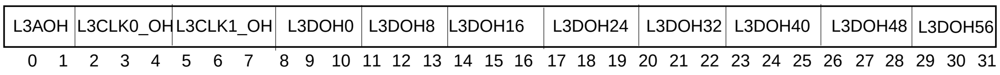


Figure 2-14. L3 Cache Output Hold Control Register (L3OHCR) for the MPC7457

The L3 cache interface is described in Chapter 3, "L1, L2, and L3 Cache Operation." The L3OHCR bits are described in Table 2-13.

Table 2-13. 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 Hardware Specification* 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 Hardware Specification* for specific output hold times.

Table 2-13. L3OHCR Field Descriptions (continued)

Bits	Name	Description
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 Hardware Specification* 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 Hardware Specification* 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 Hardware Specification* 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 Hardware Specification* 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 Hardware Specification* 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 Hardware Specification* 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 Hardware Specification* 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 Hardware Specification* 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 Hardware Specification* 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.

2.1.5.5.4 L3 Cache Input Timing Control (L3ITCR0)

The L3 cache input timing control register (L3ITCR0), shown in Figure 2-15, is a supervisor-level, implementation-specific SPR used to control the input AC timing of the L3 cache interface of the MPC7451. For the MPC7457, the L3ITCR0, shown in Figure 2-16, 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.

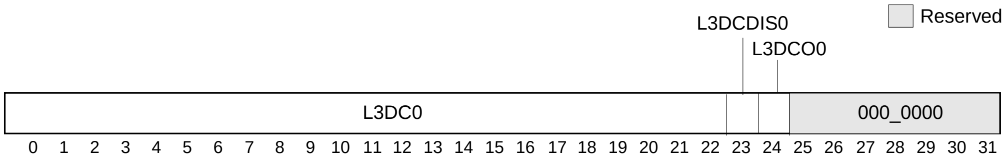


Figure 2-15. L3 Cache Control Register (L3ITCR0) for the MPC7451 and MPC7455

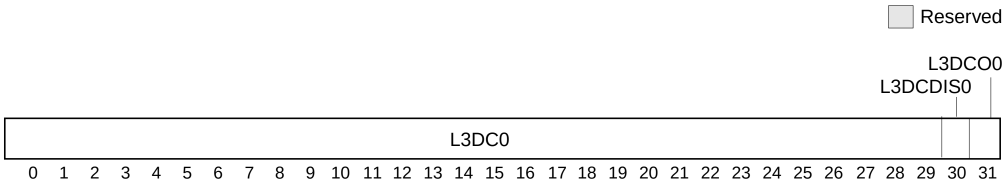


Figure 2-16. L3 Cache Control Register (L3ITCR0) for the MPC7457

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 2-14.

Table 2-14. 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 2-15.

Table 2-15. 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.

2.1.5.5.5 L3 Cache Input Timing Control (L3ITCR1)

The L3 cache input timing control register (L3ITCR1), shown in Figure 2-20, 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.

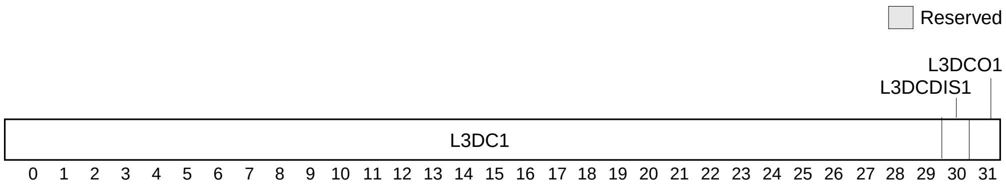


Figure 2-17. L3 Cache Control Register (L3ITCR1) for the MPC7457

The L3 cache interface is described in Chapter 3, “L1, L2, and L3 Cache Operation.” The L3ITCR0 bits for the MPC7457 are described in Table 2-16.

Table 2-16. L3ITCR1 Field Descriptions for the MPC7457

Bits	Name	Description
0-22	L3DC1	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	L3DCDIS1	L3 delay counter disable. Setting this bit disables the automic delay count configuration. Always read as 0.

Table 2-16. L3ITCR1 Field Descriptions for the MPC7457 (continued)

Bits	Name	Description
24	L3DCO1	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 1001.

2.1.5.5.6 L3 Cache Input Timing Control (L3ITCR2)

The L3 cache input timing control register (L3ITCR2), shown in Figure 2-18, 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.

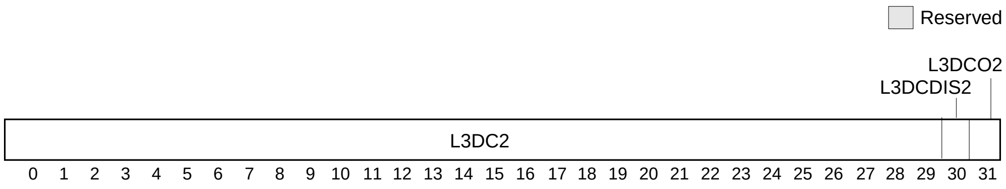


Figure 2-18. L3 Cache Control Register (L3ITCR2) for the MPC7457

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 2-16.

Table 2-17. L3ITCR2 Field Descriptions for the MPC7457

Bits	Name	Description
0-22	L3DC2	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	L3DCDIS2	L3 delay counter disable. Setting this bit disables the automic delay count configuration. Always read as 0.
24	L3DCO2	L3 delay counter override. Setting this bit overrides the automatic configuration value of the delay count. Always read as 0.
25-31		Reserved.

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

2.1.5.5.7 L3 Cache Input Timing Control (L3ITCR3)

The L3 cache input timing control register (L3ITCR3), shown in Figure 2-19, 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.

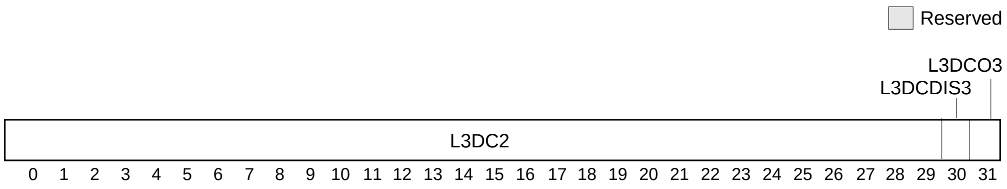


Figure 2-19. L3 Cache Control Register (L3ITCR3) for the MPC7457

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 2-18.

Table 2-18. 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.

2.1.5.5.8 Instruction Cache and Interrupt Control Register (ICTRL)

The instruction cache and interrupt control register (ICTRL), shown in Figure 2-20, 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.

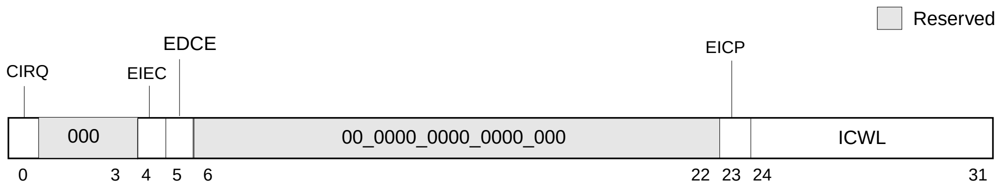


Figure 2-20. Instruction Cache and Interrupt Control Register (ICTRL)

Table 2-19 describes the bit fields for the ICTRL register.

Table 2-19. 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 MPC7451 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.

Table 2-19. ICTRL Field Descriptions (continued)

Bits	Name	Description
9–22	—	Reserved. Read as zeroes and ignores writes.
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, MPC7451 behaves the same as when HID0[ILOCK] is set. See Section 2.1.5.1, "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.

2.1.5.5.9 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 2-26.

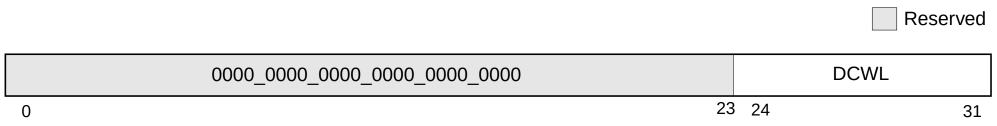


Figure 2-21. Load/Store Control Register (LDSTCR)

Table 2-25 describes the bit fields for the LDSTCR register.

Table 2-20. 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 MPC7451 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 2.3.2.4, "Synchronization."

2.1.5.5.10 L3 Private Memory Address Register (L3PM)

The L3 private address register (L3PM), shown in Figure 2-22, 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 MPC7451 private memory feature. Refer to Section 3.7.8, "L3 Private Memory Operation," for more details on the L3 private memory.

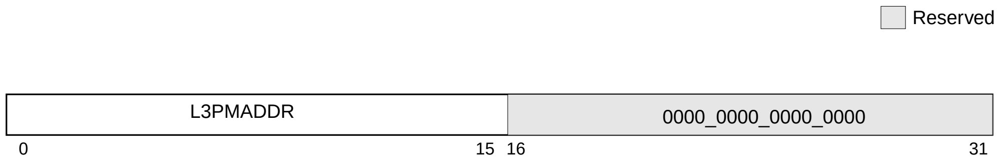


Figure 2-22. L3 Private Memory Address Register (L3PM)

The L3PM bits are described in Table 2-21.

Table 2-21. 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: 1MB L3PM[0–15] 2MB 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 2.3.2.4, "Synchronization."

2.1.5.6 Instruction Address Breakpoint Register (IABR)

The instruction address breakpoint register (IABR), shown in Table 2-23, 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 the SPR 1010. The MPC7451 requires that an **mtspr**[IABR] be followed by a context synchronizing instruction. The MPC7451 may not generate a breakpoint response for that context synchronizing instruction if the breakpoint was enabled by **mtspr**[IABR] immediately preceding it. The MPC7451 can not 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 2.3.2.4.1, “Context Synchronization.”

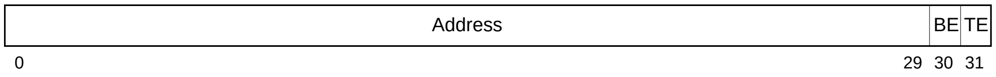


Figure 2-23. Instruction Address Breakpoint Register

The IABR bits are described in Table 2-22.

Table 2-22. Instruction Address Breakpoint Register 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 signalled. When IABR[TE] and MSR[IR] = 0 or when IABR[TE] and MSR[IR] = 1, then a match is signalled.

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

2.1.5.7 Memory Management Registers Used for Software Table Searching

This section describes the registers used by the MPC7451 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.”

2.1.5.7.1 TLB Miss Register (TLBMISS)

The TLBMISS register is automatically loaded by the MPC7451 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 MPC7451 always loads a big-endian address into the TLBMISS register. This register is read-only. The TLBMISS register has the format shown in Figure 2-24 for the MPC7451.

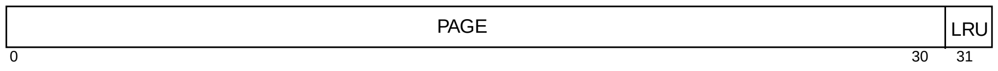


Figure 2-24. TLBMISS Register for MPC7451

Table 2-23 described the bits in the TLBMISS register.

Table 2-23. TLBMISS Register—Field and Bit Descriptions for the MPC7451

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.

2.1.5.7.2 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 2-25 shows the format for two supervisor registers, PTEHI and PTELO, respectively.

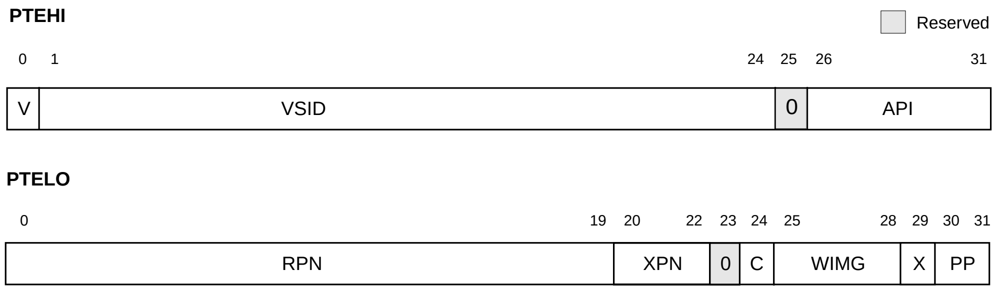


Figure 2-25. PTEHI and PTELO Registers—Extended Addressing

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 2-24 lists the corresponding bit definitions for the PTEHI and PTELO registers.

Table 2-24. 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.

2.1.5.8 Thermal Management Register

The MPC7451 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.

2.1.5.8.1 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 2-26. The overall junction temperature reduction comes from the dynamic power management of each functional unit when the MPC7451 is idle in between instruction fetches. Phase-locked loop (PLL) and delay-locked loop (DLL) configurations are unchanged.

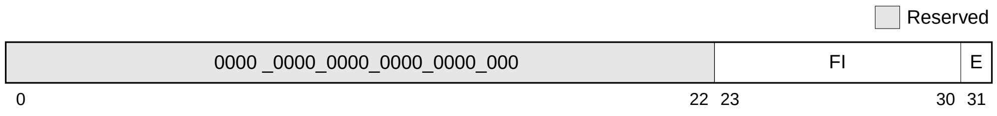


Figure 2-26. Instruction Cache Throttling Control Register (ICTC)

Table 2-25 describes the bit fields for the ICTC register.

Table 2-25. 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. (MPC7451 dispatches one instruction every INTERVAL cycle.) The minimum interval for throttling control is two cycles. 0x00, 0x01, 0x02 One instruction dispatches every 2 processor clocks.[1] 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.

2.1.5.9 Performance Monitor Registers

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

2.1.5.9.1 Monitor Mode Control Register 0 (MMCR0)

The monitor mode control register 0 (MMCR0), shown in Figure 2-27, 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 2.1.5.9.2, “User Monitor Mode Control Register 0 (UMMCR0).”

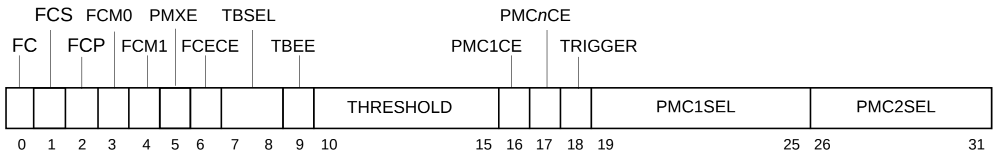


Figure 2-27. Monitor Mode Control Register 0 (MMCR0)

This register is automatically cleared at power-up. Reading this register does not change its contents. Table 2-26 describes MMCR0 fields.

Table 2-26. 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.

Table 2-26. 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 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.”
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]).

Table 2-26. MMCR0 Field Descriptions (continued)

Bits	Name	Description
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]).
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.

2.1.5.9.2 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.

2.1.5.9.3 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 2-28.

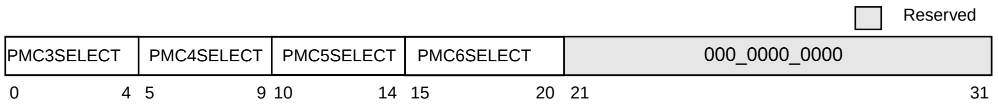


Figure 2-28. Monitor Mode Control Register 1 (MMCR1)

Bit settings for MMCR1 are shown in Table 2-27. The corresponding events are described in Section 2.1.5.9.8, “Performance Monitor Counter Registers (PMC1–PMC6).”

Table 2-27. MMCR1 Field Descriptions

Bits	Name	Description
0–4	PMC3SELECT	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	PMC4SELECT	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	PMC5SELECT	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	PMC6SELECT	PMC6 selector. Contains a code (one 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 2.1.5.9.4, “User Monitor Mode Control Register 1 (UMMCR1).”

2.1.5.9.4 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.

2.1.5.9.5 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 2-29.

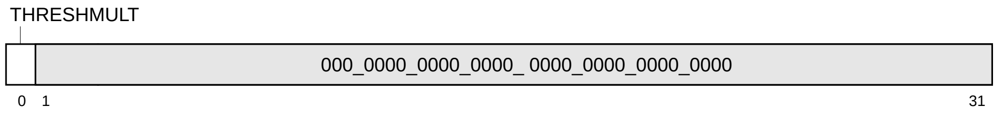


Figure 2-29. Monitor Mode Control Register 2 (MMCR2)

Table 2-28 describes MMCR2 fields.

Table 2-28. 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 2.1.5.9.6, “User Monitor Mode Control Register 2 (UMMCR2).”

2.1.5.9.6 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.

2.1.5.9.7 Breakpoint Address Mask Register (BAMR)

The breakpoint address mask register (BAMR), shown in Figure 2-30, is used in conjunction with the events that monitor IABR hits.

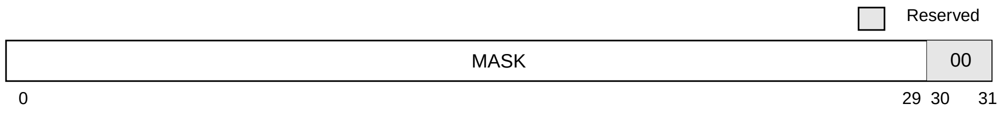


Figure 2-30. Breakpoint Address Mask Register (BAMR)

Table 2-29 describes BAMR fields.

Table 2-29. 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 2.3.2.4, “Synchronization.”

2.1.5.9.8 Performance Monitor Counter Registers (PMC1–PMC6)

PMC1–PMC6, shown in Figure 2-31, are 32-bit counters that can be programmed to generate a performance monitor exception when they overflow.

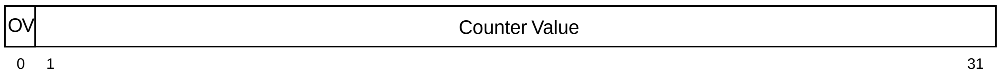


Figure 2-31. Performance Monitor Counter Registers (PMC1–PMC6)

The bits contained in the PMC registers are described in Table 2-30.

Table 2-30. 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

2.1.5.9.9 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

2.1.5.9.10 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 2-32.

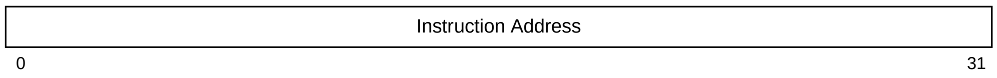


Figure 2-32. Sampled Instruction Address Registers (SIAR)

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.

2.1.5.9.11 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.

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

The MPC7451 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 MPC7451. A **mtspr** or **mfspr** SDAR or USDAR instruction causes a program exception.

2.1.6 Reset Settings

Table 2-31 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 2-31. 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

Table 2-31. Settings Caused by Hard Reset (Used at Power-On) (continued)

Resource	Setting
HID0	0x8000_0000
HID1	0x0000_0080 (note that bits 15–18 are set to match the settings of PLL_CFG[0:4] at reset)
IABR	0x0000_0000 (Breakpoint is disabled.)
ICTC	0x0000_0000
ICTRL	0x0000_0000
L2CR	0x0000_0000
L3CR	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 MPC7441, 0x8000_xxxx, where xxxx depends on the revision level, starting at 0200. For the MPC7445, 0x8001_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 MPC7451, 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 MPC7457, 0x8002_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

Table 2-31. Settings Caused by Hard Reset (Used at Power-On) (continued)

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

2.2 Operand Conventions

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

2.2.1 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 is one:

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

2.2.2 Data Organization in Memory and Data Transfers

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

Memory operands 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.

2.2.3 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 MPC7451 does not provide hardware support for floating-point memory that is not word-aligned. If a floating-point operand is not word-aligned, the MPC7451 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.

2.2.4 Floating-Point Operands

The MPC7451 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 MPC7451 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.

2.3 Instruction Set Summary

This chapter describes instructions and addressing modes defined for the MPC7451. These instructions are divided into the following functional categories:

- Integer instructions—These include arithmetic and logical instructions. For more information, see Section 2.3.4.1, “Integer Instructions.”
- Floating-point instructions—These include floating-point arithmetic instructions, as well as instructions that affect the floating-point status and control register (FPSCR). For more information, see Section 2.3.4.2, “Floating-Point Instructions.”
- Load and store instructions—These include integer and floating-point load and store instructions. For more information, see Section 2.3.4.3, “Load and Store Instructions.”
- Flow control instructions—These include branching instructions, condition register logical instructions, trap instructions, and other instructions that affect the instruction flow. For more information, see Section 2.3.4.4, “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 2.3.4.6, “Processor Control Instructions—UISA,” Section 2.3.5.1, “Processor Control Instructions—VEA,” and Section 2.3.6.2, “Processor Control Instructions—OEA.”
- Memory synchronization instructions—These instructions are used for memory synchronizing. See Section 2.3.4.7, “Memory Synchronization Instructions—UISA,” and Section 2.3.5.2, “Memory Synchronization Instructions—VEA,” for more information.
- Memory control instructions—These instructions provide control of caches and TLBs. For more information, see Section 2.3.5.3, “Memory Control Instructions—VEA,” and Section 2.3.6.3, “Memory Control Instructions—OEA.”
- External control instructions—These include instructions for use with special input/output devices. For more information, see Section 2.3.5.4, “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 MPC7451. Instructions that are

implementation specific are described in Section 2.6.2, “AltiVec Instructions with Specific Implementations for the MPC7451.”

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 four bytes long and word-aligned. It provides for byte, half-word, and word operand loads and stores between memory and a set of 32 general-purpose registers (GPRs). It 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.

2.3.1 Classes of Instructions

The MPC7451 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 MPC7451.

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.

2.3.1.1 Definition of Boundedly Undefined

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

2.3.1.2 Defined Instruction Class

Defined instructions are guaranteed to be supported in all implementations of the PowerPC architecture, except as stated in the instruction descriptions in Chapter 8, “Instruction Set,” of *The Programming Environments Manual*. The MPC7451 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 MPC7451 provides limited support for instructions represented in an invalid form.

2.3.1.3 Illegal Instruction Class

Illegal instructions can be grouped into the following categories:

- Instructions not defined in the PowerPC architecture.The following primary opcodes are defined as illegal, but 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 MPC7451.

  The following primary opcodes are defined for 64-bit implementations only and are illegal on the MPC7451:

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 2.3.1.4, “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 2.3.1.4, “Reserved Instruction Class.”

The MPC7451 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.

2.3.1.4 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 MPC7451)
- All other implementation-specific instructions
- Architecturally allowed extended opcodes

2.3.2 Addressing Modes

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

2.3.2.1 Memory Addressing

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

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

2.3.2.2 Memory Operands

Memory operands can be bytes, half words, words, 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*.

2.3.2.3 Effective Address Calculation

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

Effective address computations for both data and instruction accesses use 32-bit 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 2.3.4.3.2, “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

2.3.2.4 Synchronization

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

2.3.2.4.1 Context Synchronization

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

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

Modifying certain registers requires software synchronization to follow certain register dependencies. Table 2-32 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 2-32. 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.

Table 2-32. Control Registers Synchronization Requirements (continued)

Register	Bits	Synchronization Requirements
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	
	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	

Table 2-32. Control Registers Synchronization Requirements (continued)

Register	Bits	Synchronization Requirements
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.
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 **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.
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.

2.3.2.4.2 Execution Synchronization

An instruction is execution synchronizing if all previously initiated instructions appear to have completed before the instruction is initiated or, in the case of **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.

2.3.2.4.3 Instruction-Related Exceptions

There are two kinds of exceptions in the MPC7451—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 MPC7451 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.”

2.3.3 Instruction Set Overview

This section provides a brief overview of the PowerPC instructions implemented in the MPC7451 and highlights any special information with respect to how the MPC7451 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.

2.3.4 PowerPC UISA Instructions

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

2.3.4.1 Integer Instructions

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

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

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

2.3.4.1.1 Integer Arithmetic Instructions

Table 2-33 lists the integer arithmetic instructions for the processors that implement the PowerPC architecture.

Table 2-33. 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

Table 2-33. Integer Arithmetic Instructions (continued)

Name	Mnemonic	Syntax
Subtract from Minus One Extended	**subfme** (**subfme. subfmeo subfmeo.**)	**r**D,**r**A
Add to Zero Extended	**addze** (**addze. addzeo addzeo.**)	**r**D,**r**A
Subtract from Zero Extended	**subfze** (**subfze. subfzeo subfzeo.**)	**r**D,**r**A
Negate	**neg** (**neg. nego nego.**)	**r**D,**r**A
Multiply Low Immediate	**mulli**	**r**D,**r**A,SIMM
Multiply Low 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 MPC7451 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.

2.3.4.1.2 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 2-34 summarizes the integer compare instructions.

Table 2-34. 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*.

2.3.4.1.3 Integer Logical Instructions

The logical instructions shown in Table 2-35 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 2-35. 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	—

2.3.4.1.4 Integer Rotate and Shift Instructions

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

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

The integer rotate instructions are summarized in Table 2-36.

Table 2-36. 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 2-37.

Table 2-37. Integer Shift Instructions

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

2.3.4.2 Floating-Point Instructions

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

- Floating-point arithmetic instructions
- Floating-point multiply-add instructions

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

See Section 2.3.4.3, “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]).

2.3.4.2.1 Floating-Point Arithmetic Instructions

The floating-point arithmetic instructions are summarized in Table 2-38.

Table 2-38. 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.

2.3.4.2.2 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 2-39.

Table 2-39. 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

2.3.4.2.3 Floating-Point Rounding and Conversion Instructions

The Floating Round to Single-Precision (**frsp**) instruction is used to truncate a 64-bit double-precision number to a 32-bit single-precision floating-point number. The floating-point 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 2-40. 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

2.3.4.2.4 Floating-Point Compare Instructions

Floating-point compare instructions compare the contents of two floating-point registers. The comparison ignores the sign of zero (that is +0 = –0). The floating-point compare instructions are summarized in Table 2-41.

Table 2-41. 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

2.3.4.2.5 Floating-Point Status and Control Register Instructions

Every FPSCR instruction appears to synchronize the effects of all floating-point instructions executed by a given processor. Executing an FPSCR instruction ensures that all floating-point instructions previously initiated by the given processor appear to have completed before the FPSCR instruction is initiated and that no subsequent floating-point instructions appear to be initiated by the given processor until the FPSCR instruction has completed. The FPSCR instructions are summarized in Table 2-42.

Table 2-42. 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 MPC7451, there is no degradation of performance.

2.3.4.2.6 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 2-43 summarizes the floating-point move instructions.

Table 2-43. Floating-Point Move Instructions

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

2.3.4.3 Load and Store Instructions

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

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

Implementation Note—The following describes how the MPC7451 handles misalignment:

The MPC7451 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 MPC7451, 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 MPC7451, a TLB miss causes an instruction restart (as it causes a TLB miss exception)

2.3.4.3.1 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 MPC7451), 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*.

2.3.4.3.2 Integer Load and Store Address Generation

Integer load and store operations generate effective addresses using register indirect with immediate index mode, register indirect with index mode, or register indirect mode. See Section 2.3.2.3, "Effective Address Calculation," for information about calculating effective addresses. Note that 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.

2.3.4.3.3 Register Indirect Integer Load Instructions

For integer load instructions, the byte, half word, word, or double word addressed by the EA (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 MPC7451 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 MPC7451; 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 MPC7451. 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 MPC7451. 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 MPC7451, 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 2-44 summarizes the integer load instructions.

Table 2-44. 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

2.3.4.3.4 Integer Store Instructions

For integer store instructions, the contents of **r**S are stored into the byte, half word, word or double word in memory addressed by the 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 2-45 summarizes the integer store instructions.

Table 2-45. 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

2.3.4.3.5 Integer Store Gathering

The MPC7451 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 MPC7451 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.

2.3.4.3.6 Integer Load and Store with Byte-Reverse Instructions

Table 2-46 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 2-46. 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

2.3.4.3.7 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.

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 2-47. 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**)**

2.3.4.3.8 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 2-48 summarizes the integer load and store string instructions.

Table 2-48. 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 MPC7451 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 MPC7451 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.

2.3.4.3.9 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 MPC7451 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 MPC7451, 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 2-49 summarizes the floating-point load instructions.

Table 2-49. Floating-Point Load Instructions

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

2.3.4.3.10 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 2-50 summarizes the floating-point store instructions.

Table 2-50. 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 2-51 shows conversions the LSU makes when executing a Store Floating-Point Single instruction.

Table 2-51. Store Floating-Point Single Behavior

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

Table 2-52 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 2-52. 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 MPC7451. 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 MPC7451 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 MPC7451, there is also a case when execution of a store floating-point double (**stfd**, **stfdu**, **stfdx**, **stfdux**) instruction can require internal shifting of the mantissa. This case occurs when the operand of a store floating-point double instruction is a denormalized single-precision value. The value could be the result of a load floating-point single instruction, a single-precision arithmetic instruction, or a floating round to single-precision instruction. In these cases, shifting the mantissa takes from 1 to 23 clock cycles, depending upon the value to be stored. These cycles are incurred during the store.

2.3.4.4 Branch and Flow Control Instructions

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

2.3.4.4.1 Branch Instruction Address Calculation

Branch instructions can alter the sequence of instruction execution. Instruction addresses are always assumed to be word aligned; the 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 MPC7451, 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 MPC7451 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.

2.3.4.4.2 Branch Instructions

Table 2-53 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 2-53. 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

2.3.4.4.3 Condition Register Logical Instructions

Condition register logical instructions, shown in Table 2-54, and the Move Condition Register Field (**mcrf**) instruction are also defined as flow control instructions.

Table 2-54. 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.

2.3.4.4.4 Trap Instructions

The trap instructions shown in Table 2-55 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 2-55. 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.

2.3.4.5 System Linkage Instruction—UISA

The System Call (**sc**) instruction permits a program to call on the system to perform a service; see Table 2-56 and also Section 2.3.6.1, “System Linkage Instructions—OEA,” for additional information.

Table 2-56. 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).”

2.3.4.6 Processor Control Instructions—UISA

Processor control instructions are used to read from and write to the condition register (CR), machine state register (MSR), and special-purpose registers (SPRs). See Section 2.3.5.1, “Processor Control Instructions—VEA,” for the **mftb** instruction and Section 2.3.6.2, “Processor Control Instructions—OEA,” for information about the instructions used for reading from and writing to the MSR and SPRs.

2.3.4.6.1 Move to/from Condition Register Instructions

Table 2-57 summarizes the instructions for reading from or writing to the condition register.

Table 2-57. 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 MPC7451 is the same in both cases, if multiple fields are affected. Note that **mtcr**f 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.

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

Table 2-58 lists the **mtspr** and **mfspr** instructions.

Table 2-58. 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 2-59 lists the SPR numbers for user-level PowerPC SPR accesses.

Encodings for the MPC7451-specific user-level SPRs are listed in Table 2-60.

Table 2-59. 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
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 five bits appearing in bits 16–20 of the instruction and the low-order five bits in bits 11–15.

2 The TB registers are referred to as TBRs rather than SPRs and can be written to using the **mtsp**r 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 2-60. User-level SPR Encodings for MPC7451-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.

2.3.4.7 Memory Synchronization Instructions—UISA

Memory synchronization instructions control the order in which memory operations are completed with respect to asynchronous events, and the order in which memory operations are seen by other processors or memory access mechanisms. SeeSection 3.3.3.6, “Atomic Memory References,” for additional information about these instructions and about related aspects of memory synchronization. See Table 2-61 for a summary.

Table 2-61. 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 MPC7451 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 MPC7451 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 2.3.5.2, “Memory Synchronization Instructions—VEA,” 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 MPC7451 encounters one of these invalid instruction forms, it sets CR0 to an undefined value.

2.3.5 PowerPC VEA Instructions

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

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

2.3.5.1 Processor Control Instructions—VEA

In addition to the move to condition register instructions (specified by the UISA), the VEA defines the **mftb** instruction (user-level instruction) for reading the contents of the time base register; see Chapter 3, "L1, L2, and L3 Cache Operation," for more information. Table 2-62 shows the **mftb** instruction.

Table 2-62. 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 MPC7451, note the following:

- The MPC7451 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 MPC7451 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.

2.3.5.2 Memory Synchronization Instructions—VEA

Memory synchronization instructions control the order in which memory operations are completed with respect to asynchronous events, and the order in which memory operations are seen by other processors or memory access mechanisms. See Chapter 3, "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 2-63 describes the memory synchronization instructions defined by the VEA.

Table 2-63. 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 MPC7451 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 MPC7451 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**.

2.3.5.3 Memory Control Instructions—VEA

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 2.3.6.3, "Memory Control Instructions—OEA," 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."

2.3.5.3.1 User-Level Cache Instructions—VEA

The instructions summarized in this section help user-level programs manage on-chip caches if they are implemented. See Chapter 3, "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 MPC7451'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 MPC7451 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 2-64 summarizes the cache instructions defined by the VEA. Note that these instructions are accessible to user-level programs.

Table 2-64. 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 MPC7451 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 MPC7451 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 MPC7451. • **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, MPC7451 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 2-64. 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 2-64. 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.

2.3.5.4 Optional External Control Instructions

The PowerPC architecture defines an optional external control feature that, if implemented, is supported by the two external control instructions, **eciwx** and **ecowx**. These instructions allow a user-level program to communicate with a special-purpose device. These instructions are provided in the MPC7451 and are summarized in Table 2-65.

Table 2-65. 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.”

2.3.6 PowerPC OEA Instructions

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

2.3.6.1 System Linkage Instructions—OEA

This section describes the system linkage instructions (see Table 2-66). 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 2-66. 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 MPC7451, this means the **rfi** instruction works its way to the final stage of the execution pipeline, updates architected registers, and redirects the instruction flow.

2.3.6.2 Processor Control Instructions—OEA

The instructions listed in Table 2-67 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, “PowerPC Register Set,” of *The Programming Environments Manual* for serialization requirements and other recommended precautions to observe when manipulating the segment registers.

Table 2-67. 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 is discussed in this section. Table 2-68 lists instructions for accessing the MSR.

Table 2-68. 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 2-69.

Table 2-69. 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 2-59. Encodings for MPC7451-specific, supervisor-level SPRs are listed in Table 2-60. Simplified mnemonics are provided for **mtspr** and **mfspr** in Appendix F, “Simplified Mnemonics,” in *The Programming Environments Manual*.

Table lists the SPR numbers for supervisor-level PowerPC SPR accesses.

Table 2-70. 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

Table 2-70. Supervisor-level PowerPC SPR Encodings (continued)

Register Name	SPR [1]			Access	mfspr/mtspr
	Decimal	spr[5–9]	spr[0–4]		
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
MMCR0 [2]	952	11101	11000	Supervisor	Both
MMCR1 [2]	956	11101	11100	Supervisor	Both
PIR [2]	1023	11111	11111	Supervisor (OEA)	Both
PMC1 [2]	953	11101	11001	Supervisor	Both
PMC2 [2]	954	11101	11010	Supervisor	Both
PMC3 [2]	957	11101	11101	Supervisor	Both
PMC4 [2]	958	11101	11110	Supervisor	Both
PMC5 [2]	945	11101	10001	Supervisor	Both
PMC6 [2]	946	11101	10010	Supervisor	Both
PVR	287	01000	11111	Supervisor (OEA)	**mfspr**
SDR1	25	00000	11001	Supervisor (OEA)	Both
SIAR [2]	955	11101	11011	Supervisor	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 MPC7451-specific SPRs are listed in Table 2-60.

Table 2-71. Supervisor-level SPR Encodings for MPC7451-Defined Registers

Register Name	SPR [1]			Access	mfspr/mtspr
	Decimal	spr[5–9]	spr[0–4]		
BAMR	951	11101	10111	Supervisor	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	Both
HID1	1009	11111	10001	Supervisor	Both
IABR	1010	11111	10010	Supervisor	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	Both
ICTRL	1011	11111	10011	Supervisor	Both

Table 2-71. Supervisor-level SPR Encodings for MPC7451-Defined Registers (continued)

Register Name	SPR [1]			Access	mfspr/mtspr
	Decimal	spr[5–9]	spr[0–4]		
L2CR	1017	11111	11001	Supervisor	Both
L3CR [3]	1018	11111	11010	Supervisor	Both
L3ITCR0 [3]	984	11111	11010	Supervisor	Both
L3ITCR1 [4]	1001	11111	11010	Supervisor	Both
L3ITCR2 [4]	1002	11111	11010	Supervisor	Both
L3ITCR3 [4]	1003	11111	11010	Supervisor	Both
L3OHCR [4]	1000	11111	11010	Supervisor	Both
L3PM [3]	983	11110	10111	Supervisor	Both
LDSTCR	1016	11111	11000	Supervisor	Both
MMCR2	944	11101	10000	Supervisor	Both
MSSCR0	1014	11111	10110	Supervisor	Both
MSSSR0	1015	11111	10111	Supervisor	Both
PTEHI	981	11110	10101	Supervisor	Both
PTELO	982	11110	10110	Supervisor	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
TLBMISS	980	11110	10100	Supervisor	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 only, register may not be supported on other processors that implement the PowerPC architecture

3 MPC7451-, MPC7455-, MPC7457-specific register, not supported on the MPC7441, MPC7445, and MPC7447

4 MPC7457-specific register, not supported on the MPC7441, MPC7445, MPC7447, MPC7451,and MPC7455

2.3.6.3 Memory Control Instructions—OEA

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 2.3.5.3, “Memory Control Instructions—VEA,” describes user-level memory control instructions.

2.3.6.3.1 Supervisor-Level Cache Management Instruction—(OEA)

Table 2-72 lists the only supervisor-level cache management instruction.

Table 2-72. 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 2.3.5.3.1, “User-Level Cache Instructions—VEA.”

See Section 2.3.5.3.1, “User-Level Cache Instructions—VEA,” 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.

2.3.6.3.2 Translation Lookaside Buffer Management Instructions—OEA

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 MPC7451 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 2.3.8, “Implementation-Specific Instructions.”

See Chapter 7, “Memory Management,” for more information about TLB operations. Table 2-73 summarizes the operation of the TLB instructions in the MPC7451. Note that the broadcast of **tlbie** and **tlbsync** instructions is enabled by the setting of HID1[SYNCBE].

Table 2-73. 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].

Table 2-73. Translation Lookaside Buffer Management Instruction

Name	Mnemonic	Syntax	Implementation Notes
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 MPC7451. As described above, **tlbie** can be used to invalidate a particular index of the TLB based on EA[14–19]—a sequence of 64 **tlbie** instructions followed by a **tlbsync** instruction invalidates all the TLB structures (for EA[14–19] = 0, 1, 2, . . . , 63). Attempting to execute **tlbia** causes an illegal instruction program exception.

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

2.3.7 Recommended Simplified Mnemonics

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 *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.

2.3.8 Implementation-Specific Instructions

This section provides the details for the two MPC7451 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 MPC7451-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

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

```
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 MPC7451-specific instruction, and not part of the PowerPC instruction set.

Other registers altered:

- None

2.4 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 2.3.4.1, “Integer Instructions.”
- Vector floating-point arithmetic instructions—These floating-point arithmetic instructions and floating-point modes are described in Section 2.3.4.2, “Floating-Point Instructions.”
- Vector load and store instructions—These load and store instructions for vector registers are described in Section 2.5.3, “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 2.5.5, “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 2.3.4.6, “Processor Control Instructions—UISA.”
- Memory control instructions—These instructions are used for managing caches (user level and supervisor level), and are described in Section 2.6.1, “AltiVec Vector Memory Control Instructions—VEA.”

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.

2.5 AltiVec UISA Instructions

This section describes the instructions defined in the AltiVec user instruction set architecture (UISA).

2.5.1 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 2.5.1.2, “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.

2.5.1.1 Vector Integer Arithmetic Instructions

Table 2-74 lists the integer arithmetic instructions for the processors that implement the PowerPC architecture.

Table 2-74. Vector Integer Arithmetic Instructions

Name	Mnemonic	Syntax
Vector Add Unsigned Integer [b,h,w] Modulo1	**vaddubm** **vadduhm** **vadduwm**	**v**D,**v**A,**v**B
Vector Add Unsigned Integer [b,h,w] Saturate	**vaddubs** **vadduhs** **vadduws**	**v**D,**v**A,**v**B
Vector Add Signed Integer [b.h.w] Saturate	**vaddsbs** **vaddshs** **vaddsws**	**v**D,**v**A,**v**B
Vector Add and Write Carry-out Unsigned Word	**vaddcuw**	**v**D,**v**A,**v**B
Vector Subtract Unsigned Integer Modulo	**vsububm** **vsubuhm** **vsubuwm**	**v**D,**v**A,**v**B

Table 2-74. Vector Integer Arithmetic Instructions (continued)

Name	Mnemonic	Syntax
Vector Subtract Unsigned Integer Saturate	**vsububs** **vsubuhs** **vsubuws**	**v**D,**v**A,**v**B
Vector Subtract Signed Integer Saturate	**vsubsbs** **vsubshs** **vsubsws**	**v**D,**v**A,**v**B
Vector Subtract and Write Carry-out Unsigned Word	**vsubcuw**	**v**D,**v**A,**v**B
Vector Multiply Odd Unsigned Integer [b,h] Modulo	**vmuloub** **vmulouh**	**v**D,**v**A,**v**B
Vector Multiply Odd Signed Integer [b,h] Modulo	**vmulosb** **vmulosh**	**v**D,**v**A,**v**B
Vector Multiply Even Unsigned Integer [b,h] Modulo	**vmuleub** **vmuleuh**	**v**D,**v**A,**v**B
Vector Multiply Even Signed Integer [b,h] Modulo	**vmulesb** **vmulesh**	**v**D,**v**A,**v**B
Vector Multiply-High and Add Signed Half-Word Saturate	**vmhaddshs**	**v**D,**v**A,**v**B, **v**C
Vector Multiply-High Round and Add Signed Half-Word Saturate	**vmhraddshs**	**v**D,**v**A,**v**B,**v**C
Vector Multiply-Low and Add Unsigned Half-Word Modulo	**vmladduhm**	**v**D,**v**A,**v**B,**v**C
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

Table 2-74. Vector Integer Arithmetic Instructions (continued)

Name	Mnemonic	Syntax
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

2.5.1.2 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 2-76) 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 2-75.

Table 2-75. 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 2-76 summarizes the vector integer compare instructions.

Table 2-76. 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

2.5.1.3 Vector Integer Logical Instructions

The vector integer logical instructions shown in Table 2-77 perform bit-parallel operations on the operands.

Table 2-77. 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

2.5.1.4 Vector Integer Rotate and Shift Instructions

The vector integer rotate instructions are summarized in Table 2-78.

Table 2-78. 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 2-79.

Table 2-79. 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

2.5.2 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*.

2.5.2.1 Vector Floating-Point Arithmetic Instructions

The floating-point arithmetic instructions are summarized in Table 2-80.

Table 2-80. 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

2.5.2.2 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 2-81.

Table 2-81. 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

2.5.2.3 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 2-82.

Table 2-82. Vector Floating-Point Rounding and Conversion Instructions

Name	Mnemonic	Syntax
Vector Round to Floating-Point Integer Nearest	**vrfin**	**v**D,**v**B
Vector Round to Floating-Point Integer toward Zero	**vrfiz**	**v**D,**v**B
Vector Round to Floating-Point Integer toward Positive Infinity	**vrfip**	**v**D,**v**B
Vector Round to Floating-Point Integer toward Minus Infinity	**vrfim**	**v**D,**v**B
Vector Convert from Unsigned Fixed-Point Word	**vcfux**	**v**D,**v**B,UIMM

Table 2-82. Vector Floating-Point Rounding and Conversion Instructions

Name	Mnemonic	Syntax
Vector Convert from Signed Fixed-Point Word	**vcfsx**	**v**D,**v**B,UIMM
Vector Convert to Unsigned Fixed-Point Word Saturate	**vctuxs**	**v**D,**v**B,UIMM
Vector Convert to Signed Fixed-Point Word Saturate	**vctsxs**	**v**D,**v**B,UIMM

2.5.2.4 Vector Floating-Point Compare Instructions

The floating-point compare instructions are summarized in Table 2-83.

Table 2-83. Vector Floating-Point Compare Instructions

Name	Mnemonic	Syntax
Vector Compare Greater Than Floating-Point [Record]	**vcmpgtfp[.]**	**v**D,**v**A,**v**B
Vector Compare Equal to Floating-Point [Record]	**vcmpeqfp[.]**	**v**D,**v**A,**v**B
Vector Compare Greater Than or Equal to Floating-Point [Record]	**vcmpgeqfp[.]**	**v**D,**v**A,**v**B
Vector Compare Bounds Floating-Point [Record]	**vcmpbfp[.]**	**v**D,**v**A,**v**B

2.5.2.5 Vector Floating-Point Estimate Instructions

The floating-point estimate instructions are summarized in Table 2-84.

Table 2-84. Vector Floating-Point Estimate Instructions

Name	Mnemonic	Syntax
Vector Reciprocal Estimate Floating-Point	**vrefp**	**v**D,**v**B
Vector Reciprocal Square Root Estimate Floating-Point	**vrsqrtefp**	**v**D,**v**B
Vector Log2 Estimate Floating-Point	**vlogefp**	**v**D,**v**B
Vector 2 Raised to the Exponent Estimate Floating-Point	**vexptefp**	**v**D,**v**B

2.5.3 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.

2.5.3.1 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 2-85 summarizes the vector load instructions.

Table 2-85. 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 MPC7451, **lvxl** and **stvxl** are interpreted to be transient. See Section 7.1.2.3, "Data Stream Touch Instructions."

2.5.3.2 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 2-86 summarizes the vector alignment instructions.

Table 2-86. 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

2.5.3.3 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 2-87 provides a summary of the vector store instructions.

Table 2-87. 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 MPC7451, **lvxl**, **stvxl** are interpreted to be transient. See Section 7.1.2.3, “Data Stream Touch Instructions.”

2.5.4 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.

2.5.5 Vector Permutation and Formatting Instructions

Vector pack, unpack, merge, splat, permute, and select can be used to accelerate various vector math operations and vector formatting. Details of these instructions follow.

2.5.5.1 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 2-88 describes the vector pack instructions.

Table 2-88. 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
Vector Pack Signed Integer [h,w] Unsigned Saturate	**vpkshus** **vpkswus**	**v**D, **v**A, **v**B

Table 2-88. Vector Pack Instructions (continued)

Name	Mnemonic	Syntax
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

2.5.5.2 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 2-89 describes the unpack instructions.

Table 2-89. 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

2.5.5.3 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 2-90 describes the merge instructions.

Table 2-90. 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

2.5.5.4 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 2-91. 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

2.5.5.5 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 2-88 describes the vector permute instruction.

Table 2-92. Vector Permute Instruction

Name	Mnemonic	Syntax
Vector Permute	**vperm**	**v**D**,** **v**A**,v**B**,v**C

2.5.5.6 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 2-93 describes the **vsel** instruction.

Table 2-93. Vector Select Instruction

Name	Mnemonic	Syntax
Vector Select	**vsel**	**v**D,**v**A,**v**B,**v**C

2.5.5.7 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 ($0 \leq \text{count} \leq 127$). 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 2-94 describes the vector shift instructions.

Table 2-94. 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

2.5.5.8 Vector Status and Control Register Instructions

Table 2-95 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 2-95. 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

2.6 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.

2.6.1 AltiVec Vector Memory Control Instructions—VEA

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 MPC7451 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 2-96 summarizes the directed prefetch cache instructions defined by the AltiVec VEA. Note that these instructions are accessible to user-level programs.

Table 2-96. 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 MPC7451
Data Stream Touch for Store Transient	**dststt**	**r**A,**r**B,STRM	Not recommended for use in MPC7451
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.”

2.6.2 AltiVec Instructions with Specific Implementations for the MPC7451

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 MPC7451 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.”